Hash Index

by Sophia

WHAT'S COVERED

This lesson explores the use of hash indexes in databases, in three parts. Specifically, this lesson will cover:

1. Understanding Hash Indexes
2. Uses for a Hash Index
3. Examples

1. Understanding Hash Indexes

A **hash index** offers fast lookup times for exact match queries but is not suitable for other types of queries. That means when you are choosing an index type when indexing a particular column, you must consider the type of queries you will most often run when deciding if a hash index is the best choice.

A **hash function** is a math algorithm that takes an input (such as a data value stored in a specific row) and produces a fixed size string of characters called a **hash value**. The same input will always produce the same hash value.

In hash indexing, every time you enter a value in the column being indexed, the database runs that value through the hash algorithm for that index and stores the resulting hash value in the hash table, along with a pointer to the corresponding original value in the table.

When you want to look up a value in the indexed column, the database uses your query input to compute the hash value for it, using the same hash function. Then it looks up the hash value in the hash table and uses the associated pointer to quickly jump to the matching row in the dataset.

HINT

One of the nice things about a hash index is that it offers constant-time retrieval. That means the time it takes to find an item doesn't increase when the size of the collection increases.

TERMS TO KNOW

Hash Index

An index that stores the hashes of the values of the indexed fields and looks up data by hashing the query's request and comparing it to the hash values.

Hash Function

An algorithm that computes the hash value of data by performing math operations on it.

Hash Value

The result of running data through a hash function.

2. Uses for a Hash Index

Hash indexing can greatly speed retrieval times for exact match searches because it directly maps the search key to the data location.

However, hash indexes are valuable only when you are running an equality search—that is, a query with a WHERE clause that uses an equality operator (an = sign).

The main drawback of a hash index is its specific focus. Hash indexes are not suitable for queries involving ranges because the hash value represents a single value; it can't represent a range.

It's also not good in situations where multiple data values happen to produce the same hash value, and the index returns erroneous results. This situation is known as a **hash collision**.

EXAMPLE Suppose the hash function did something as simple as counting the number of characters in the value. For example, the name Bob would have a hash value of 3. The problem with that is that every name with three letters would also have a hash value of 3, like Ann and May. There are internal techniques for handling collisions, but multiple collisions can degrade the search speed and negate the value of using the index.

TERM TO KNOW

Hash Collision

A situation in which multiple data values produce the same hash value, so the index returns erroneous results.

3. Examples

Let us take a look at instances where a hash index would be ideal:

```
SELECT *
```

```
FROM track
WHERE name = 'Walk On';

SELECT *
FROM track
WHERE album_id = 5;

SELECT *
FROM customer
WHERE country = 'USA';
```

What do these queries have in common? They all use the equality operator. On the other hand, queries like the following ones would not use the hash index, as they don't use the = operator:

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

SELECT *
FROM track
WHERE album_id >=5;

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

SUMMARY

In this lesson, you developed an **understanding of the basics of hash indexes**, including how they work and what they are good for. You learned that a hash index is created by running data values through a hash function, which is an algorithm that generates a fixed length value and places it in a table cell. That cell also contains a pointer to the actual data that the hash value represents. When a search is run, the search request itself is hashed, and then that value is compared to the hash values in the hash table to locate the matching entry and direct the query to the associated row in the table. When multiple data values happen to produce the same hash value, it is known as a hash collision. Collisions can be worked around, but they slow things down, partially or fully negating the value of the index.

You also learned to identify **uses for a hash index**. Hash indexes are specialized and beneficial only when running equality queries—that is, queries that use the WHERE clause and an = sign. They are not good for queries involving ranges. Lastly, you saw some **examples** of queries that are good or not so good for hash indexes.

Source: THIS TUTORIAL WAS AUTHORED BY DR. VINCENT TRAN, PHD (2020) AND FAITHE WEMPEN (2024) FOR SOPHIA LEARNING. PLEASE SEE OUR **TERMS OF USE**.

TERMS TO KNOW

Hash Collision

A situation in which multiple data values produce the same hash value, so the index returns erroneous results.

Hash Function

An algorithm that computes the hash value of data by performing math operations on it.

Hash Index

An index that stores the hashes of the values of the indexed fields and looks up data by hashing the query's request and comparing it to the hash values.

Hash Value

The result of running data through a hash function.