

# Hash Index

*by Sophia Tutorial*



## WHAT'S COVERED

This tutorial explores the use of hash indexes in databases in two parts:

1. Hash Index Explanation
2. Examples

## 1. Hash Index Explanation

Hash indexes are unique in that they are really only used for equality operators = and single-value lookup scenarios. They aren't used for comparison operators that find a range of values. Basically, a hash index has an array of N number of buckets where each one has a pointer to a row in the table. The hash index uses a function that takes a key and the N number of buckets and maps the key to the corresponding bucket of the hash index. The hash function uses an algorithm that maps data of a variable length to data of a fixed length in a specific but random way.

One simple example of a hash algorithm could be that it takes a string and returns the length of a string. Let us say that we have 10 buckets, as the length of the column is 10, and we pass into the hash function "Bob". Since the length of "Bob" is equal to 3, it would be placed in the third bucket. If we passed into the hash function "Jennifer", we would have the value of 8 as the length, so it would go in the 8th bucket. So far, that seems to be fairly easy as a hash function.

However, what happens in the case that you have a value hashed to the same place? For example, if we have "Ron", the length is 3 and it would point to the third bucket. In this case, we have two different keys ("Bob" and "Ron"), but they have the same hash value based on our function. In this case, we have a collision, which is very common in hash functions. The more collisions a hash function has, the worse it can be, as it can have a performance impact when reading values. This can also occur if the number of buckets is too small compared to the number of distinct keys.

There are many ways to solve this collision problem. One approach is to have the pointer of the first item point to the next item and so forth. This way, you would group all of the items that hash to the same bucket linked to one another. Remember that the hash algorithm that we used is just a simple approach, but there can be more complex types of hash algorithms that will evenly distribute the data into buckets. For example, if we hashed the data based on the first name and had 100 buckets, most of the data would only be in the first 10 buckets or so. The remaining buckets would be mostly empty. Having an algorithm that would distribute the names across all 100 buckets would make it much more even and quicker to find data.

## 2. 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. Queries like this 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

The hash index uses a hashing function to speed up queries that use the equality operator.

Source: Authored by Vincent Tran