@B-Tree

<https://www.geeksforgeeks.org/introduction-of-b-tree-2/>

@Insert Operation in B-Tree

<https://www.geeksforgeeks.org/insert-operation-in-b-tree/>

@Delete Operation in B-Tree

<https://www.geeksforgeeks.org/delete-operation-in-b-tree/>

@Introduction of B+ Tree

<https://www.geeksforgeeks.org/introduction-of-b-tree/>

@why indexing in dbms

1. Indexing is used to optimise the performance of a database by minimising the number of disk accesses required when a query is

processed.

2. The index is a type of data structure. It is used to locate and access the data in a database table quickly.

3. Speeds up operation with read operations like SELECT queries, WHERE clause etc.

Search Key: Contains copy of primary key or candidate key

of the table or something else.

5. Data Reference: Pointer holding the address of disk block

where the value of the corresponding key is stored.

6. Indexing is optional, but increases access speed. It is not the

primary mean to access the tuple, it is the secondary mean.

7. Index file is always sorted.

<https://www.geeksforgeeks.org/indexing-in-databases-set-1/>

@Indexing Methods

<https://www.geeksforgeeks.org/sql-queries-on-clustered-and-non-clustered-indexes/>

**Database indexes** work in a similar manner. They guide the database to the exact location of the data, enabling faster and more efficient data retrieval.

In this article, we'll explore:

* What are database indexes?
* How do they work?
* Benefits of using them.
* Different types of indexes.
* Which data structure they use?
* How to use them smartly?

**What are Database Indexes?**

**Key Pointer 
INDEX TABLE 
C 
E 
A 
B 
newsletter. ashishps. 
囗 吅 山 囗 
com 
DATABASE TABLE **

A database index is a super-efficient lookup table that allows a database to find data much faster.

It holds the indexed column values along with pointers to the corresponding rows in the table.

Without an index, the database might have to scan every single row in a massive table to find what you want – a painfully slow process.

But, with an index, the database can zero in on the exact location of the desired data using the index’s pointers.

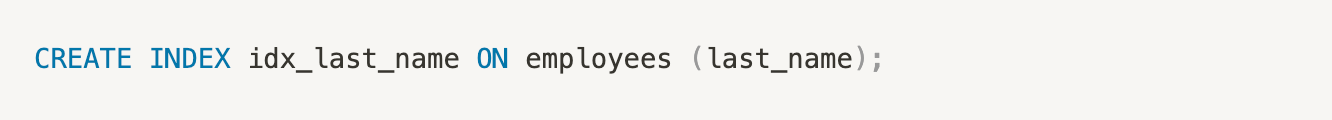
**How to create Indexes?**

Here's an example of creating an index in a MySQL database.

Let's say we have a table named employees with the following structure:

CREATE TABLE employees 
id INT PRIMARY KEY, 
first name VARCHAR(5Ø) 
last_name VARCHAR(5Ø) 
email VARCHAR(IØO) 
department VARCHAR(5Ø) 

Now, let's create an index on the last\_name column to improve the performance of queries that frequently search or sort based on the last name.



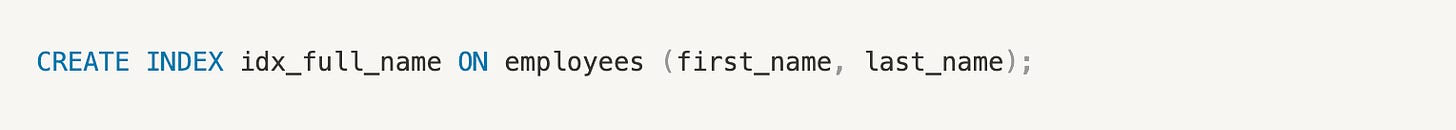
In this example, we use the CREATE INDEX statement to create an index named idx\_last\_name on the employees table. The index is created on the last\_name column.

After creating the index, queries that involve conditions or sorting on the last\_name column will be optimized. For example:

SELECT * FROM employees WHERE last_name 
= I Smith I; 

This query will use the idx\_last\_name index to quickly locate the rows where the last\_name is 'Smith', avoiding a full table scan.

You can also create indexes on multiple columns (composite indexes) if your queries frequently involve conditions on multiple columns together. For example:



This creates a composite index on the first\_name and last\_name columns, which can be useful for queries that search or sort based on both columns.

**📣** [**CodeCrafters.io**](https://app.codecrafters.io/join?via=ashishps1)

If you want to take your **Software Engineering** skills to the next level, check out [**CodeCrafters**](https://app.codecrafters.io/join?via=ashishps1).

**CodeCrafters** is a **YC backed** platform that offers a unique, **hands-on** approach to practice **complex** **programming challenges** like building your own **Redis, Git, SQLite** and **Bittorrent** client from scratch.

[**Sign up and get 40% off if you upgrade.**](https://app.codecrafters.io/join?via=ashishps1)

**How do Database Indexes Work?**

Here's a step-by-step explanation of how database indexes work:

1. **Index Creation**: The database administrator creates an index on a specific column or set of columns.
2. **Index Building**: The database management system builds the index by scanning the table and storing the values of the indexed column(s) along with a pointer to the corresponding data.
3. **Query Execution**: When a query is executed, the database engine checks if an index exists for the requested column(s).
4. **Index Search**: If an index exists, the database searches the index for the requested data, using the pointers to quickly locate the data.
5. **Data Retrieval**: The database retrieves the requested data, using the pointers from the index.

**Benefits of Database Indexes**

Database indexes offer several benefits, including:

* **Faster Query Performance**: Indexes can significantly improve query performance especially for large datasets by reducing the amount of data that needs to be scanned.
* **Reduced CPU Usage**: By reducing the number of rows that need to be scanned, indexes can decrease CPU usage and optimize resource utilization.
* **Rapid Data Retrieval**: Indexes enable quick data retrieval for queries that involve equality or range conditions on the indexed columns.
* **Efficient Sorting**: Indexes can also be used to efficiently sort data based on the indexed columns, eliminating the need for expensive sorting operations.
* **Better Data Organization**: Indexes can help maintain data organization and structure, making it easier to manage and maintain the database.

**Types of Database Indexes**

**Indexes based on Structure and Key Attributes:**

* **Primary Index:** Automatically created when a primary key constraint is defined on a table. Ensures uniqueness and helps with super-fast lookups using the primary key.
* **Clustered Index:** Determines the order in which data is physically stored in the table. A clustered index is most useful when we’re searching in a range. Only one clustered index can exist per table.
* **Non-clustered or Secondary Index:** This index does not store data in the order of the index. Instead, it provides a list of virtual pointers or references to the location where the data is actually stored.

**Indexes based on Data Coverage:**

* **Dense index:** Has an entry for every search key value in the table. Suitable for situations where the data has a small number of distinct search key values or when fast access to individual records is required.
* **Sparse index:** Has entries only for some of the search key values. Suitable for situations where the data has a large number of distinct search key values.

**Specialized Index Types:**

* **Bitmap Index:** Excellent for columns with low cardinality (few distinct values). Common in data warehousing.
* **Hash Index:** A index that uses a hash function to map values to specific locations. Great for exact match queries.
* **Filtered Index:** Indexes a subset of rows based on a specific filter condition. Useful to improve query speed on commonly filtered columns.
* **Covering Index:** Includes all the columns required by a query in the index itself, eliminating the need to access the underlying table data.
* **Function-based index:** Indexes that are created based on the result of a function or expression applied to one or more columns of a table.
* **Full-Text Index**: A index designed for full-text search, allowing for efficient searching of text data.
* **Spatial Index:** Used for indexing geographical data types.

Subscribe to receive new articles every week.

Subscribed

**What Data Structure do Indexes use?**

Most commonly used data structures that power indexes are B-Trees, Hash Tables and Bitmaps.

**B-Tree (Balanced Tree)**

**newsletter. ashishps. 
com 
12 
24 
27 **

Most database engines use either a B-Tree or a variation of B-Trees like B+ Trees.

B-Trees have a hierarchical structure with a root node, internal nodes (index nodes), and leaf nodes.

Each node in a B-Tree contains a sorted array of keys and pointers to child nodes.

Here's why they are so well-suited:

* **Self-Balancing:** B-trees ensure that the 'height' of the tree stays balanced even when inserting or deleting data. This ensures logarithmic time complexity for insertion, deletion, and searching.
* **Ordered:** B-trees keep the data sorted, making range queries ("find all orders between date X and Y") and inequality comparisons very fast.
* **Disk-Friendly:** B-trees are designed to work well with disk-based storage. A single node of a B-tree often corresponds to a disk block, minimizing disk access operations.

Many databases use a slightly modified B-tree variant called the B+ tree.

In a B+ tree, all data values are stored only in the leaf nodes, which can further improve performance for certain use cases like range queries.

**Hash Tables**

**Search Key 
Hash Function 
newsletter. ashishps. 
com 
Pointer to 
Row 5 
Pointer to 
Row 6 
Pointer to 
Address 7 
Buckets 
("D3", 1, "2024") 
Data 
Row 1 
Row 2 
Row 3 
Row 4 
Row 5 
Row 6 
Row 7 **

Hash tables are used for hash indexes, which are based on a hash function.

A hash table consists of an array of buckets, with each bucket containing the addresses for rows in the data.

Hash indexes employ a hash function to map keys to their corresponding bucket in the hash table, enabling constant-time lookup operations.

Hash indexes provide fast equality lookups, as the hash function determines the exact location of the data based on the key.

However, hash indexes do not support range queries or sorting efficiently.

**Bitmaps**

**АО 110 
в 1001 
со 101 **

Each bit in the bitmap corresponds to a row, and the value of the bit indicates whether the key value exists in that row.

Bitmap indexes use a bitmap (a binary array) to represent the presence or absence of a specific key value in each row of a table.

Bitmap indexes are well-suited for columns with low cardinality (a small number of distinct values) and for performing complex queries involving multiple conditions.

Bitmap operations like AND, OR, and NOT are performed efficiently using bitwise operations, making bitmap indexes suitable for analytical queries involving multiple columns.

**How to use Database Indexes Smartly?**

To get the most out of database indexes, consider these best practices:

* **Identify Query Patterns**: Analyze the most frequent and critical queries executed against your database to determine which columns to index and which type of index to use.
* **Index Frequently Used Columns**: Consider indexing columns that are frequently used in WHERE, JOIN, and ORDER BY clauses.
* **Index Selective Columns:** Indexes are most effective on columns with a good spread of data values (high cardinality). Indexing a gender column might be less beneficial than one with a unique customer\_id.
* **Use Appropriate Index Types**: Choose the right index type for your data and queries.
* **Consider Composite Indexes**: For queries involving multiple columns, consider creating composite indexes that encompass all relevant columns. This reduces the need for multiple single-column indexes and improves query performance.
* **Monitor Index Performance**: Regularly monitor index performance, remove unused indexes and adjust your indexing strategy as the database workload evolves.
* **Avoid Over-Indexing**: Avoid creating too many indexes, as this can lead to increased storage requirements and slower write performance.
  + Indexes take up extra disk space since they're additional data structures that need to be stored alongside your tables.
  + Every time you insert, update, or delete data in a table with an index, the index needs to update too. This can slightly slow down write operations.

To summarize, indexes are a powerful tool to optimize database query performance.

But remember to choose the right column and index type, monitor performance, and avoid over-indexing to get the most out of them.

**How DBMS Indexing done to improve search query performance? Explained**

**VI. How Table data is actually stored? 
Emp ID 
1 
2 
3 
4 
Name 
c 
This is just logical epresentation. 
Actual Data is not 
stored in this way. 
Address 
City A 
City B 
City C 
City D 
- DBMS creates Data Pages (generally its 8KB but depends upon DB to DB) 
- Each Data Page can store multiple table rows in it. **

Just an example of Data page. 
8KB 
(8192bytes) 
96 bytes 
8060 bytes 
36 bytes 
Header 
Data 
Records 
Offset 
PageNo 
Free space 
checksum etc. 
Actual Data is stored in here 
Contains an array, each index of an array 
holds a pointer to 
corresponding data in the Data records. 

able row is lets say 4Byte 
Emp ID 
Nyne 
so in 1 data page we can have N 125 DB rows 
PagelD: 1 
64Bytes 
City B 
City c 
Rowl 
Row2 
Row3 
ow 
Row5 

DBMS creates and manage these data pages. As for storing 1 table data, it can 
create many data pages. 
These data pages ultimately gets stored in th Data Block in physical memory 
like disk. 
What is Data block? 
- Data Block is the minimum amount of data which can be read/write by an 
I/O operation. 
- It is managed by underlying storage system like disk. Data Block size can 
range from 4kb to 32kb (common size is 8KB). 
- So based on the data block size, it can hold 1 or many Data page. 