Index

What is an Index?

An index in a database is a data structure that improves the speed of data retrieval operations on a table at the cost of additional writes and storage space to maintain it. Indexes are used to quickly locate data without having to search every row in a database table every time a database table is accessed.

Types of Indexes

1. **Primary Index**:
    - Automatically created when a primary key is defined.
    - Ensures that the values in the key are unique and not null.
2. **Unique Index**:
    - Ensures that all the values in the indexed column are unique.
    - Created using the UNIQUE keyword.
3. **Non-Unique Index**:
    - Allows duplicate values in the indexed column.
    - Created without the UNIQUE keyword.
4. **Composite Index**:
    - An index on multiple columns of a table.
    - Useful for queries involving multiple columns in the WHERE clause.
5. **Full-Text Index**:
    - Special type of index used for full-text searches.
    - Suitable for searching large text fields.
6. **Spatial Index**:

- Used for spatial data types.
- Enhances performance for spatial queries.

How Indexes Work

Indexes are typically implemented using B-trees or B+ trees. Here's how they work in general:

- **B-tree Structure**: The B-tree structure keeps data sorted and allows searches, sequential access, insertions, and deletions in logarithmic time.
- **B+ Tree**: A variation of the B-tree, used in databases to improve the efficiency of range queries.

When a query is executed, the database engine uses the index to quickly locate the data, reducing the number of rows that need to be scanned.

Benefits of Using Indexes

1. **Improved Query Performance**:

    - Speeds up the retrieval of rows by reducing the number of disk I/O operations.
    - Efficient for large tables.

2. **Faster Sorting**:

    - Helps in quickly sorting rows using the indexed columns.

3. **Quick Access to Rows**:

    - Enhances performance for SELECT, JOIN, and WHERE clauses.

Costs of Using Indexes

1. **Increased Storage**:

    - Indexes consume additional disk space.

2. **Slower Write Operations**:

   - INSERT, UPDATE, and DELETE operations can be slower because the index must be updated whenever the data changes.

3. **Maintenance Overhead**:

   - Requires ongoing maintenance, especially for frequently updated tables.

When to Use Indexes

1. **Frequent Searches**:

   - Use indexes on columns that are frequently searched.

2. **Sorting**:

   - Useful for columns involved in ORDER BY clauses.

3. **Joins**:

   - Indexes on foreign keys can speed up join operations.

4. **Unique Constraints**:

   - Enforce uniqueness on columns.

When Not to Use Indexes

1. **Small Tables**:

   - For small tables, full table scans are usually more efficient than using indexes.

2. **Frequent Updates**:

   - Tables with frequent insertions, updates, or deletions may suffer from the overhead of maintaining indexes.

3. **Columns with High Cardinality**:

- Columns with a high number of unique values might not benefit much from indexing.

Step 1: Create the Table

First, let's create a table named Employees.

```
CREATE TABLE Employees (
    EmployeeID INT PRIMARY KEY AUTO_INCREMENT,
    FirstName VARCHAR(50),
    LastName VARCHAR(50),
    Position VARCHAR(50),
    Salary DECIMAL(10, 2),
    HireDate DATE
);
```

Step 2: Insert Records

Next, let's insert some records into the Employees table.

```
INSERT INTO Employees (FirstName, LastName, Position, Salary,
HireDate) VALUES
('Rahul', 'Sharma', 'Manager', 60000, '2020-01-15'),
('Anjali', 'Verma', 'Developer', 50000, '2019-02-10'),
('Raj', 'Kumar', 'Designer', 45000, '2021-03-22'),
('Priya', 'Patel', 'Tester', 40000, '2018-07-30');
```

Step 3: Create Indexes

Indexes are used to speed up the retrieval of data from a database table. Here are examples of creating different types of indexes:

Creating an Index on a Single Column

```
CREATE INDEX idx_lastname ON Employees (LastName);
```

Creating a Composite Index on Multiple Columns

```
CREATE INDEX idx_lastname_position ON Employees (LastName, Position);
```

Complete SQL Script

Here is the complete SQL script combining all the steps:

```
-- Step 1: Create the Employees table
CREATE TABLE Employees (
    EmployeeID INT PRIMARY KEY AUTO_INCREMENT,
    FirstName VARCHAR(50),
    LastName VARCHAR(50),
    Position VARCHAR(50),
    Salary DECIMAL(10, 2),
    HireDate DATE
);

-- Step 2: Insert records into the Employees table
INSERT INTO Employees (FirstName, LastName, Position, Salary, HireDate) VALUES
('Rahul', 'Sharma', 'Manager', 60000, '2020-01-15'),
('Anjali', 'Verma', 'Developer', 50000, '2019-02-10'),
('Raj', 'Kumar', 'Designer', 45000, '2021-03-22'),
('Priya', 'Patel', 'Tester', 40000, '2018-07-30');

-- Step 3: Create indexes
-- Create an index on the LastName column
CREATE INDEX idx_lastname ON Employees (LastName);

-- Create a composite index on the LastName and Position columns
CREATE INDEX idx_lastname_position ON Employees (LastName, Position);
```

Verifying the Indexes

To verify that the indexes have been created, you can use the following SQL statement:

```
SHOW INDEX FROM Employees;
```

This will show a list of all indexes created on the Employees table, including their types and the columns they cover.

```
-- Drop the index on the LastName column

DROP INDEX idx_lastname ON Employees;

-- Drop the composite index on the LastName and Position columns
DROP INDEX idx_lastname_position ON Employees;
```