Indexing for Query Optimization in MySQL

SPEED UP
YOUR QUERIES
WITH THE
POWER OF
INDEXES!

WHAT IS AN INDEX?

An index is a database object that helps speed up data retrieval by creating a quick lookup reference for rows in a table.

Indexes store a copy of selected columns and provide a pointer to the rows where the data is located. Think of it as a table of contents in a book, helping you find what you're looking for faster!

WHY SHOULD YOU USE INDEXES?

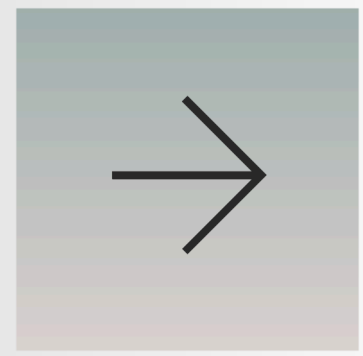
Indexes help with:

Faster query performance: They allow the database to skip scanning through the entire table.

Optimized search results: Especially for large datasets where you frequently search, filter, or join tables.

Note:

While indexing speeds up queries, it may slow down insertions, updates, and deletions. So, use indexes wisely!



HOW DOES INDEXING WORK?

Indexes work like a sorted list that the database uses to find data quickly.

Without an index: MySQL has to scan every row to find the matching results (Full Table Scan).

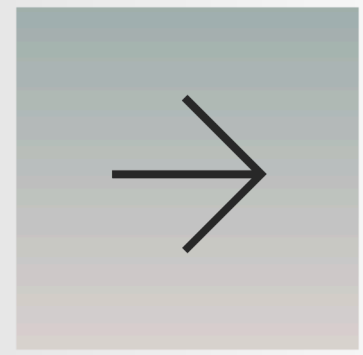
With an index: MySQL can jump directly to the matching rows using the index (Indexed Scan).

Indexes use B-trees and other structures to store data in a way that’s efficient for search and retrieval.

TYPES OF INDEXES IN MYSQL

MySQL offers several types of indexes:

1. Primary Index: Automatically created for the primary key.
2. Unique Index: Ensures that no duplicate values exist in the indexed columns.
3. Composite Index: An index on multiple columns.
4. Full-Text Index: Optimized for text searches in large datasets.



WHEN TO USE INDEXES?

Indexes should be used when:

- Searching or filtering data frequently.
- Sorting or ordering results with ORDER BY.
- Joining tables where indexed columns improve join performance.

Avoid over-indexing as it can slow down write operations (inserts, updates, and deletes).

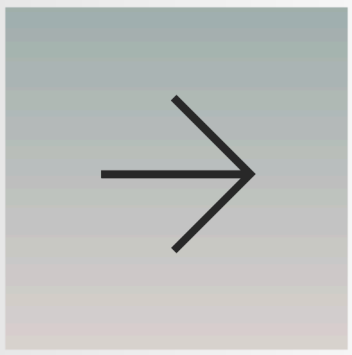
CREATING AN INDEX IN MYSQL

```
CREATE INDEX index_name
ON table_name (column_name);
```

Example:

```
CREATE INDEX idx_employee_last_name
ON employees (last_name);
```

Here, we create an index on the last_name column
in the employees table. This improves performance
when searching for employees by last name.



DROPPING AN INDEX

```
DROP INDEX index_name
ON table_name;
```

Example:

```
DROP INDEX idx_employee_last_name
ON employees;
```

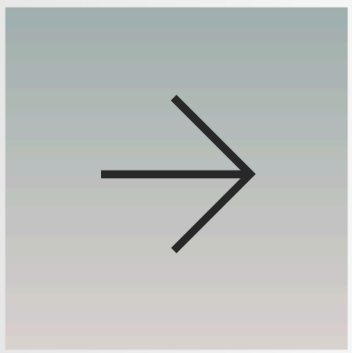
This removes the idx_employee_last_name index from the employees table. You might drop an index if it’s no longer needed or if it’s slowing down write operations.

COMPOSITE INDEX EXAMPLE

Example:

```
CREATE INDEX idx_employee_name
ON employees (first_name, last_name);
```

A composite index on both first_name and last_name helps optimize searches that involve bo columns. MySQL will first search by first_name, the narrow down the search using last_name.



BEST PRACTICES FOR INDEXING

Limit the number of indexes: Too many indexes can slow down write operations.
Index frequently used columns: Especially those in WHERE, JOIN, and ORDER BY clauses.
Use composite indexes: For queries involving multiple columns.
Monitor index usage: Use the EXPLAIN command to see how MySQL uses your indexes.

PRACTICE SQL CODE FOR INDEXING

Test these commands on your own table to understand how indexing works:

```
-- Create an example employees table
CREATE TABLE employees (
employee_id INT PRIMARY KEY,
first_name VARCHAR(50),
last_name VARCHAR(50),
hire_date DATE
);

-- Add an index on last_name
CREATE INDEX idx_last_name
ON employees (last_name);

-- Create a composite index on first_name and last_name
CREATE INDEX idx_name
ON employees (first_name, last_name);

-- Drop an index
DROP INDEX idx_last_name
ON employees;
```

Get hands-on with indexing and feel the power of optimized queries!

Test indexing strategies on your own database and watch your query performance soar