**Lab Exercise 1: Creating a Simple Index**

**Task**

Create a table employees with columns id, name, department, and salary. Then, create an index on the name column.

**Solution**

CREATE TABLE employees (

id INT PRIMARY KEY AUTO\_INCREMENT,

name VARCHAR(100),

department VARCHAR(50),

salary DECIMAL(10,2)

);

CREATE INDEX idx\_name ON employees(name);

**Lab Exercise 2: Creating a Composite Index**

**Task**

Create a composite index on department and salary in the employees table.

**Solution**

CREATE INDEX idx\_dept\_salary ON employees(department, salary);

**Lab Exercise 3: Using EXPLAIN to Analyze Index Usage**

**Task**

Use EXPLAIN to check if MySQL is using the index for the following query:

SELECT \* FROM employees WHERE name = 'John Doe';

**Solution**

EXPLAIN SELECT \* FROM employees WHERE name = 'John Doe';

The output should show idx\_name in the possible\_keys column, indicating that the index is being used.

**Lab Exercise 4: Creating a UNIQUE Index**

**Task**

Ensure that employee names are unique by creating a unique index.

**Solution**

CREATE UNIQUE INDEX idx\_unique\_name ON employees(name);

If name should allow duplicates, a unique index should **not** be used.

**Lab Exercise 5: Dropping an Index**

**Task**

Drop the index idx\_dept\_salary from the employees table.

**Solution**

DROP INDEX idx\_dept\_salary ON employees;

**Lab Exercise 6: Creating a FULLTEXT Index**

**Task**

Modify the employees table and add a description column. Then, create a FULLTEXT index on name and description.

**Solution**

ALTER TABLE employees ADD COLUMN description TEXT;

CREATE FULLTEXT INDEX idx\_fulltext\_name\_desc ON employees(name, description);

**Lab Exercise 7: Testing a FULLTEXT Index with MATCH() AGAINST()**

**Task**

Write a query to find employees whose name or description contains the word "developer".

**Solution**

SELECT \* FROM employees

WHERE MATCH(name, description) AGAINST('developer');

**Lab Exercise 8: Using a Prefix Index**

**Task**

Create an index on the first 10 characters of the name column.

**Solution**

CREATE INDEX idx\_name\_prefix ON employees(name(10));

**Lab Exercise 9: Creating an Index for ORDER BY Optimization**

**Task**

Create an index that helps optimize the following query:

SELECT \* FROM employees ORDER BY salary DESC;

**Solution**

CREATE INDEX idx\_salary ON employees(salary);

**Lab Exercise 10: Viewing Indexes in a Table**

**Task**

List all indexes on the employees table.

**Solution**

SHOW INDEX FROM employees;

**Explanation of All Index Types in MySQL**

Indexes in MySQL are used to improve query performance by allowing the database to find rows faster. However, they also take up space and can slow down INSERT, UPDATE, and DELETE operations because indexes need to be updated. Below are different types of indexes in MySQL.

**1. PRIMARY KEY Index**

* A **primary key** is a unique identifier for each row in a table.
* It **automatically creates a unique index**.
* A table can have only **one** primary key.

**Example:**

CREATE TABLE employees (

id INT PRIMARY KEY, -- Automatically indexed

name VARCHAR(100),

department VARCHAR(50),

salary DECIMAL(10,2)

);

✅ **Use Case:** Ensures uniqueness and fast lookups on the primary key.

**2. UNIQUE Index**

* Ensures that **all values in a column are unique**.
* Unlike a primary key, a table can have **multiple unique indexes**.

**Example:**

CREATE UNIQUE INDEX idx\_unique\_name ON employees(name);

✅ **Use Case:** Enforce uniqueness on columns like email or phone\_number.

**3. INDEX (Normal Index)**

* The simplest form of index.
* Speeds up searches using WHERE clauses.

**Example:**

CREATE INDEX idx\_name ON employees(name);

✅ **Use Case:** If you frequently search by name, this index improves query performance.

**4. COMPOSITE Index (Multi-column Index)**

* Indexing **multiple columns** together to speed up queries that filter by multiple conditions.

**Example:**

CREATE INDEX idx\_dept\_salary ON employees(department, salary);

✅ **Use Case:**  
This index helps **queries using both department and salary**, like:

SELECT \* FROM employees WHERE department = 'IT' AND salary > 50000;

⚠️ **Note:**

* The order of columns **matters** in a composite index.
* The index is **useful for queries using the first column (department) or both**, but **not just salary alone**.

**5. FULLTEXT Index**

* Used for **text searching** (e.g., searching inside articles or descriptions).
* Works with MATCH() and AGAINST() functions.

**Example:**

CREATE FULLTEXT INDEX idx\_fulltext\_name\_desc ON employees(name, description);

✅ **Use Case:**  
Used for **searching words** in name and description, e.g.:

SELECT \* FROM employees

WHERE MATCH(name, description) AGAINST('developer');

⚠️ **Note:**

* Only works with **CHAR, VARCHAR, and TEXT** columns.
* Best for **long text searches** like blogs, articles, or product descriptions.

**6. PREFIX Index**

* Indexes only the **first few characters** of a column (useful for large VARCHAR or TEXT fields).

**Example:**

CREATE INDEX idx\_name\_prefix ON employees(name(10));

✅ **Use Case:**  
Useful when **you only search the first few characters** of name instead of the whole field.

⚠️ **Note:**

* Only works for **VARCHAR and TEXT** columns.
* Helps save space but may **reduce efficiency compared to full-column indexes**.

**7. CLUSTERED Index**

* In **MySQL InnoDB**, the **PRIMARY KEY is the clustered index**.
* It means the **table's physical order** follows the **primary key** order.

✅ **Use Case:**  
Optimizes **range-based queries** on the primary key.

⚠️ **Note:**

* Only **one clustered index per table** (usually the **primary key**).
* Slower inserts if the key is not **sequential** (e.g., UUIDs vs. AUTO\_INCREMENT).

**8. SECONDARY Index**

* Any index that is **not a primary key or clustered index**.
* Used to speed up lookups for non-primary key columns.

✅ **Use Case:**  
When you frequently search using a **non-primary key** column.

**9. SPATIAL Index**

* Used for **geometric** data types (POINT, LINESTRING, POLYGON).
* Helps with **GIS (Geographical Information System) searches**.

**Example:**

CREATE TABLE locations (

id INT PRIMARY KEY AUTO\_INCREMENT,

coordinates POINT NOT NULL,

SPATIAL INDEX (coordinates)

);

✅ **Use Case:**  
Used in **mapping applications** to quickly find locations.

⚠️ **Note:**

* Only works with **MyISAM or InnoDB** tables.
* Queries use ST\_Contains(), ST\_Within(), etc.

**10. FUNCTIONAL Index (MySQL 8.0+)**

* Indexes are based on an **expression or function output**.

**Example:**

CREATE INDEX idx\_lowercase\_name ON employees((LOWER(name)));

✅ **Use Case:**  
If you often **search for case-insensitive names**, like:

SELECT \* FROM employees WHERE LOWER(name) = 'john doe';

⚠️ **Note:**

* Available only in **MySQL 8.0+**.

**📌 Summary of Index Types in MySQL**

| **Index Type** | **Purpose** |
| --- | --- |
| **PRIMARY KEY** | Uniquely identifies rows; automatically indexed |
| **UNIQUE** | Ensures uniqueness of values in a column |
| **INDEX** (Normal) | Speeds up searches on a single column |
| **COMPOSITE** | Indexes multiple columns together |
| **FULLTEXT** | Efficient text-based search |
| **PREFIX** | Indexes only part of a column (saves space) |
| **CLUSTERED** | Physically orders the table by the primary key |
| **SECONDARY** | Any non-primary key index |
| **SPATIAL** | Indexes geometric data for location-based searches |
| **FUNCTIONAL** | Index based on computed expressions |

**📌 Best Practices for Using Indexes**

✅ **Use indexes for columns used in WHERE, JOIN, ORDER BY, or GROUP BY**  
✅ **Use composite indexes carefully; order matters**  
✅ **Use EXPLAIN to check if queries benefit from indexes**  
✅ **Avoid too many indexes (they slow down inserts/updates)**  
✅ **Use FULLTEXT for long text searches instead of LIKE '%text%'**