**INDEXES IN SQL**

**What is an Index in SQL?**

An **index** is like the index of a book.

* In a book → you use the index to find a topic quickly, without reading every page.
* In a database → an **index helps the SQL engine find rows faster**, without scanning the entire table.

**Why Indexes are Needed**

Let’s say we have a table employees with **10 lakh rows**.

SELECT \* FROM employees WHERE emp\_id = 10045;

Without an index → SQL must **scan all 10 lakh rows** → **Full Table Scan (slow)**

With an index on emp\_id → SQL jumps directly to the row (like searching in a dictionary) → **Index Seek (fast)**

So, indexes are mainly for:

* Faster **SELECT** queries
* Faster **WHERE**, **JOIN**, **ORDER BY**, and **GROUP BY** operations

**How to Create an Index**

**Example 1 Single column index**

CREATE INDEX idx\_employee\_id

ON employees(emp\_id);

Now queries filtering by emp\_id will be faster.

**Example 2 Composite index (multiple columns)**

CREATE INDEX idx\_dept\_salary

ON employees(department\_id, salary);

Used when your query filters by **both** columns, e.g.:

SELECT \* FROM employees

WHERE department\_id = 5 AND salary > 50000;

Tip: Order of columns matters in composite index!

**Example 3: Unique index**

CREATE UNIQUE INDEX idx\_email\_unique

ON employees(email);

This not only speeds up search but also ensures **no duplicate emails** (like UNIQUE constraint).

A **UNIQUE INDEX** does **not remove** duplicates that already exist.  
Instead, it **prevents** new duplicates from being inserted.

Here’s what happens internally 👇

1. When you **create a UNIQUE INDEX**, the database checks existing data first:
   * ✅ If all existing values are unique → it creates the index successfully.
   * ❌ If duplicates already exist → it throws an error and **fails to create** the index.
2. Once the index is created:
   * Any future **INSERT** or **UPDATE** that tries to create a duplicate value will **fail** automatically.

**Real-Time Use Cases**

| **Use Case** | **Example Query** | **Solution** |
| --- | --- | --- |
| Searching for users by ID | WHERE user\_id = ? | Index on user\_id |
| Searching by email/username | WHERE email = ? | Unique Index on email |
| Filtering big tables by date | WHERE order\_date BETWEEN '2024-01-01' AND '2024-01-31' | Index on order\_date |
| Joining large tables | JOIN customers c ON orders.cust\_id = c.id | Index on cust\_id in both tables |
| Sorting results | ORDER BY created\_at DESC | Index on created\_at |
| Frequent aggregations | GROUP BY department\_id | Index on department\_id |

**When NOT to Use Indexes**

Indexes make SELECT faster but **slow down**:

* INSERT
* UPDATE
* DELETE

Because the database must update the index **every time data changes**.

So, **don’t** create indexes on:

* Small tables
* Columns with lots of duplicate values (e.g. gender, status)
* Tables frequently updated but rarely queried

**Checking Index Usage**

You can analyze whether your query uses an index with:



**If it shows Using index → your index is working.**

**What happens internally:**

The database engine creates a separate data structure (a B-tree) for that column:

B-Tree (sorted by emp\_id)

**[5000]**

**/ \**

**[1-4999] [5001-10000]**

**Quick Formula to Decide**

Ask yourself 4 questions:

1️⃣ Is the column used in WHERE / JOIN / GROUP BY / ORDER BY?  
→ If yes → Consider indexing

2️⃣ Does it have many unique values?  
→ If yes → Good index candidate

3️⃣ Is the table large (10k+ rows)?  
→ If yes → Index helps

4️⃣ Is the column updated often?  
→ If yes → Avoid index

**Indexes are stored permanently in the database**

…until you drop them manually or drop the table.

They’re not temporary structures.

When you create an index:

**The database engine (like MySQL, PostgreSQL, etc.) physically creates a separate file or structure on disk to store the index data.**It stays there even after:

* You restart the DB server
* You shut down your system
* You insert/update/delete data (index auto-updates)

**Example Proof**

1️⃣ Create table:

CREATE TABLE employees (

emp\_id INT PRIMARY KEY,

name VARCHAR(50),

department VARCHAR(50)

);

2️⃣ Create index:

CREATE INDEX idx\_dept ON employees(department);

3️⃣ Restart MySQL → Query:

SHOW INDEX FROM employees;

You’ll still see idx\_dept listed — meaning it’s stored permanently.

**When Index Is Lost**

Indexes are dropped only when:

* You manually remove it:
* DROP INDEX idx\_dept ON employees;
* Or when you drop the entire table:
* DROP TABLE employees;

Otherwise, they remain there forever — just like the data itself.