What is Indexing in SQL?

Indexing is a technique in SQL used to improve the performance of queries by creating a data structure that allows the database to locate rows faster, similar to how an index in a book helps you find topics quickly.

Why Do We Need Indexing?

- **Faster Query Execution**: Reduces the time it takes to search for rows in large datasets.
- **Efficient Filtering**: Speeds up operations like `WHERE`, `JOIN`, and `GROUP BY`.
- **Optimized Sorting**: Improves performance for queries with `ORDER BY` clauses.

How Does Indexing Work?

1. An index creates a separate data structure (often a B-tree or hash table) for the indexed column(s).
2. When a query is executed, the database checks the index for matching values instead of scanning the entire table.
3. If a match is found, the database retrieves the associated rows faster.

> **Example**: Without an index, searching for a specific name in a table with 1 million rows would require scanning all rows. With an index, it only searches through the indexed structure.

Types of Indexes in SQL

1. **Primary Index**
    - Automatically created for the primary key column.

Example:

```
CREATE TABLE students (
    student_id INT PRIMARY KEY,
    name VARCHAR(100)
);
```

    -
        - The primary key `student_id` is indexed automatically.
2. **Unique Index**
    - Ensures all values in the indexed column are unique.

Example:

```
CREATE UNIQUE INDEX idx_email
ON users(email);
```

    -
3. **Non-Unique Index**
    - Used to improve performance without uniqueness constraints.

Example:

```
CREATE INDEX idx_department
```

```
ON employees(department_id);
```

4. **Composite Index (Multi-Column Index)**
    - Indexes multiple columns together for compound conditions.

Example:

```
CREATE INDEX idx_name_dept
ON employees(last_name, department_id);
```

5. **Full-Text Index**
    - Used for text-based searches.

Example (MySQL):

```
CREATE FULLTEXT INDEX idx_fulltext_name
ON articles(content);
```

6. **Clustered Index** (database-specific)
    - Sorts and stores data rows in the table based on the key.
    - Automatically created for primary keys in databases like SQL Server.
7. **Non-Clustered Index**
    - Creates a pointer to the actual data without sorting the table rows.
    - Common in most SQL databases.

Examples to Understand Indexing

Basic Index Creation

```
CREATE TABLE employees (
    employee_id INT PRIMARY KEY,
    first_name VARCHAR(50),
    last_name VARCHAR(50),
    department_id INT
);

CREATE INDEX idx_department_id ON employees(department_id);
```

1.

Using an Index in a Query

```
SELECT *
FROM employees
WHERE department_id = 101;
```

2.
    - Without an index: Scans all rows.
    - With an index: Quickly locates rows matching `department_id = 101`.

Composite Index

```
CREATE INDEX idx_name_dept
ON employees(last_name, department_id);


SELECT *
FROM employees
WHERE last_name = 'Smith' AND department_id = 5;
```

3.

When Not to Use Indexing

- On small tables (full table scans are faster).
- Frequently updated columns (indexes slow down `INSERT`/`UPDATE`/`DELETE`).
- Columns with high duplication or low cardinality (e.g., `gender`).

Best Practices

- Index columns frequently used in `WHERE`, `JOIN`, and `ORDER BY`.
- Avoid over-indexing (too many indexes can degrade performance during writes).
- Regularly analyze and optimize indexes based on query patterns.