SQL Basics & CRUD

What is SQL

- **SQL (Structured Query Language)** is the standard language for interacting with relational databases (like MySQL, PostgreSQL, SQLite, SQL Server, Oracle).
- **Uses:**
    - Create databases & tables
    - Insert, update, delete data
    - Query (retrieve) data
    - Manage permissions & transactions

Relational databases organize data in **tables (rows & columns)**.

- **Row (record):** A single entry in the table.
- **Column (field):** An attribute of the data.

CRUD Operations

CRUD = **Create, Read, Update, Delete**

- The 4 fundamental operations in databases.
- Most apps (e.g., Instagram, banking apps) rely on CRUD behind the scenes.

CREATE (Insert Data)

- Create new databases, tables, or rows.

-- Create a table

```
CREATE TABLE Customers (
    CustomerID INT PRIMARY KEY,
    Name VARCHAR(100),
    Email VARCHAR(100),
    Age INT
);

-- Insert rows into table
INSERT INTO Customers (CustomerID, Name, Email, Age)
VALUES (1, 'Alice', 'alice@gmail.com', 25);

INSERT INTO Customers (CustomerID, Name, Email, Age)
VALUES (2, 'Bob', 'bob@yahoo.com', 30);
```

Adds new data to the database.

READ (Select Data)

- Retrieve data with **SELECT**.

```
-- Get all records
SELECT * FROM Customers;

-- Get only names and emails
SELECT Name, Email FROM Customers;

-- Filter with WHERE
SELECT * FROM Customers WHERE Age > 25;

-- Sort data
SELECT * FROM Customers ORDER BY Age DESC;
```

Used for queries, reports, and analytics.

UPDATE (Modify Data)

- Change existing records.

```
-- Update one record
```

```
UPDATE Customers
SET Email = 'alice123@gmail.com'
WHERE CustomerID = 1;

-- Increase age of all customers by 1
UPDATE Customers
SET Age = Age + 1;
```

Updates specific or multiple rows.

DELETE (Remove Data)

- Delete records from a table.

```
-- Delete one customer
DELETE FROM Customers
WHERE CustomerID = 2;

-- Delete all records (⚠ dangerous!)
DELETE FROM Customers;
```

Removes unnecessary or invalid data.

Other Important SQL Basics

Constraints

- **PRIMARY KEY** → Unique identifier per row.
- **FOREIGN KEY** → Links tables together.
- **NOT NULL** → Field must have a value.
- **UNIQUE** → No duplicates allowed.

Aggregate Functions

```
SELECT COUNT(*) FROM Customers;        -- total rows
SELECT AVG(Age) FROM Customers;        -- average age
SELECT MIN(Age), MAX(Age) FROM Customers; -- youngest & oldest
```

Example :CRUD in Python

```
import sqlite3

# Connect to database (creates file if not exists)
conn = sqlite3.connect("mydb.db")
cursor = conn.cursor()

# Create table
cursor.execute("""
CREATE TABLE IF NOT EXISTS Customers (
    CustomerID INTEGER PRIMARY KEY,
    Name TEXT,
    Email TEXT,
    Age INTEGER
)
""")

# CREATE (Insert)
cursor.execute("INSERT INTO Customers (Name, Email, Age) VALUES (?, ?, ?)",
               ("Alice", "alice@gmail.com", 25))
conn.commit()

# READ (Select)
cursor.execute("SELECT * FROM Customers")
print(cursor.fetchall())

# UPDATE
cursor.execute("UPDATE Customers SET Age = ? WHERE Name = ?", (26, "Alice"))
conn.commit()

# DELETE
cursor.execute("DELETE FROM Customers WHERE Name = ?", ("Alice",))
conn.commit()

conn.close()
```

JOINs in SQL

Purpose: Combine data from two or more tables based on a related column.

Types of JOINs:

INNER JOIN – Returns only the matching rows from both tables.

```
SELECT employees.name, departments.department_name
FROM employees
INNER JOIN departments
ON employees.department_id = departments.id;
```

LEFT JOIN (or LEFT OUTER JOIN) – Returns all rows from the left table and matching rows from the right table. Non-matching rows from the right table show NULL.

```
SELECT employees.name, departments.department_name
FROM employees
LEFT JOIN departments
ON employees.department_id = departments.id;
```

RIGHT JOIN (or RIGHT OUTER JOIN) – Returns all rows from the right table and matching rows from the left table. Non-matching rows from the left table show NULL.

```
SELECT employees.name, departments.department_name
FROM employees
RIGHT JOIN departments
ON employees.department_id = departments.id;
```

FULL OUTER JOIN – Returns all rows from both tables. Non-matching rows show NULL.

```
SELECT employees.name, departments.department_name
FROM employees
FULL OUTER JOIN departments
ON employees.department_id = departments.id;
```

CROSS JOIN – Returns the Cartesian product of two tables.

```
SELECT employees.name, departments.department_name
FROM employees
CROSS JOIN departments;
```

GROUP BY in SQL

Purpose: Aggregate data based on one or more columns.

Basic syntax:

```
SELECT column1, aggregate_function(column2)
FROM table
GROUP BY column1;
```

Common Aggregate Functions:

- `COUNT()` – Count of rows
- `SUM()` – Sum of values
- `AVG()` – Average

- `MIN()` – Minimum
- `MAX()` – Maximum

Example: Count employees per department

```
SELECT department_id, COUNT(*) AS num_employees
FROM employees
GROUP BY department_id;
```

Example: Average salary per department

```
SELECT department_id, AVG(salary) AS avg_salary
FROM employees
GROUP BY department_id;
```

GROUP BY with HAVING – Filter aggregated results

```
SELECT department_id, COUNT(*) AS num_employees
FROM employees
GROUP BY department_id
HAVING COUNT(*) > 5;  -- Only departments with more than 5 employees
```