“SQL Queries with Examples”

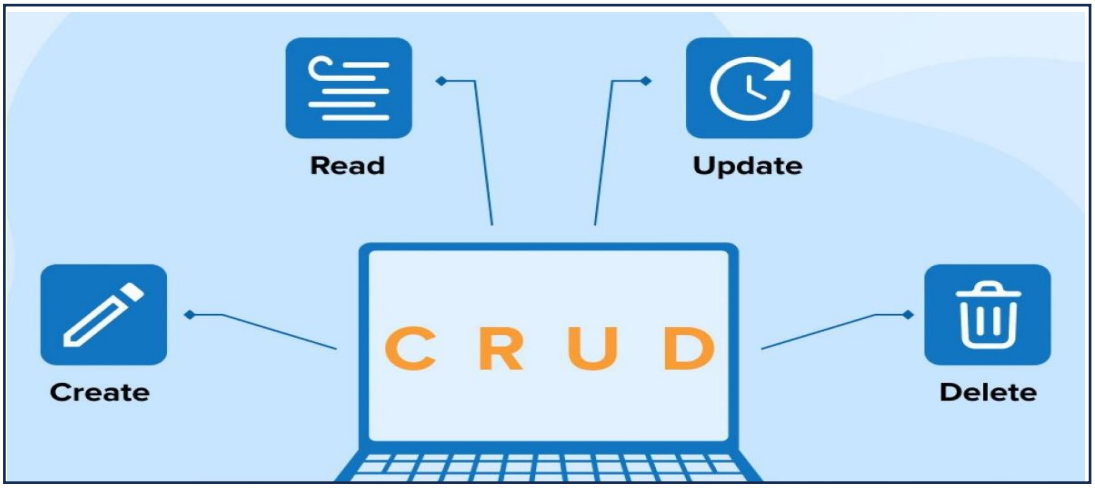
Documented By: HARSHA K

1. Domain-Driven Schema Design:

Design your database based on the core business domain. For example, in an education system, the main entities might be: Student, Teacher, Course, etc.

2. CRUD Operations in SQL

At the core of all SQL operations lie the four fundamental actions known as CRUD — Create, Read, Update, and Delete. These operations form the building blocks for managing data in any relational database. Whether we are inserting a new row (Create), retrieving specific records using SELECT (Read), modifying data with UPDATE, or removing entries with DELETE, every SQL query is essentially one or a combination of these operations. Understanding CRUD helps simplify complex queries and provides a strong foundation for database interaction.



3. Basic SQL Operations:

Table Creation

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

Insert Sample Data

```
INSERT INTO students VALUES
 (1, 'Anbu', 'Data Analysis', '2025-07-17'),
 (2, 'Bala', 'Data Engineering', '2025-07-15'),
 (3, 'Campbell', 'Data Science', '2025-07-18'),
 (4, 'David', 'Data Analyst', '2025-07-17'),
```

SELECT Queries

```
SELECT * FROM students;
SELECT name, course FROM students;
```

WHERE Queries

```
SELECT * FROM students WHERE course = 'Data Engineering';
SELECT * FROM students WHERE join_date > '2025-07-15';
SELECT * FROM students WHERE course = 'Data Engineering' AND join_date > '2025-07-18';
SELECT * FROM students WHERE course IN ('Data Science', 'Data Analyst');
SELECT * FROM students WHERE join_date BETWEEN '2025-07-15' AND '2025-07-17';
SELECT * FROM students WHERE name LIKE 'A%';
SELECT * FROM students WHERE name LIKE '%a';
SELECT * FROM students WHERE name LIKE '%a%';
```

UPDATE Statements

```
UPDATE students SET course = 'Advanced Data Engineering' WHERE student_id = 1;
UPDATE students SET join_date = '2025-09-20' WHERE name = 'Bala';
UPDATE students SET join_date = ADDDATE("2025-06-15", INTERVAL 1 DAY);
```

DELETE Statements

```
DELETE FROM students WHERE student_id = 2;
DELETE FROM students WHERE join_date < '2025-09-16';
```

4. Subqueries:

Inline Subquery Example

```
CREATE TABLE employees (
  emp_id INT PRIMARY KEY,
  emp_name VARCHAR(100),
  department VARCHAR(50),
  salary INT,
  age INT);

INSERT INTO employees VALUES
  (1, 'Amit', 'HR', 30000, 25),
  (2, 'Neha', 'IT', 45000, 28),
  (3, 'Rahul', 'IT', 50000, 30),
  (4, 'Divya', 'Sales', 40000, 26),
  (5, 'Kiran', 'Sales', 35000, 24),
  (6, 'Meena', 'HR', 32000, 29);
```

Analytic Function: RANK()

```
SELECT emp_name, department, salary,
  RANK() OVER (ORDER BY salary DESC) AS salary_rank
FROM employees;
```

5. SQL Joins:

Creating Tables

```
CREATE TABLE customers (
  customer_id INT PRIMARY KEY,
  customer_name VARCHAR(100),
  city VARCHAR(50));
```

```
INSERT INTO customers VALUES
 (1, 'Amit Sharma', 'Delhi'),
 (2, 'Neha Reddy', 'Hyderabad'),
 (3, 'Rahul Iyer', 'Mumbai'),
 (4, 'Divya Mehta', 'Chennai');

CREATE TABLE orders (
 order_id INT PRIMARY KEY,
 customer_id INT,
 product_name VARCHAR(100),
 order_amount INT,
 FOREIGN KEY (customer_id) REFERENCES customers(customer_id));

INSERT INTO orders VALUES
 (101, 1, 'Laptop', 55000),
 (102, 2, 'Mouse', 500),
 (103, 1, 'Keyboard', 1500),
 (104, 3, 'Monitor', 7000),
 (105, 2, 'Printer', 8500);
```

INNER JOIN

```
SELECT c.customer_name, o.product_name, o.order_amount
FROM customers c
INNER JOIN orders o ON c.customer_id = o.customer_id;
```

LEFT JOIN

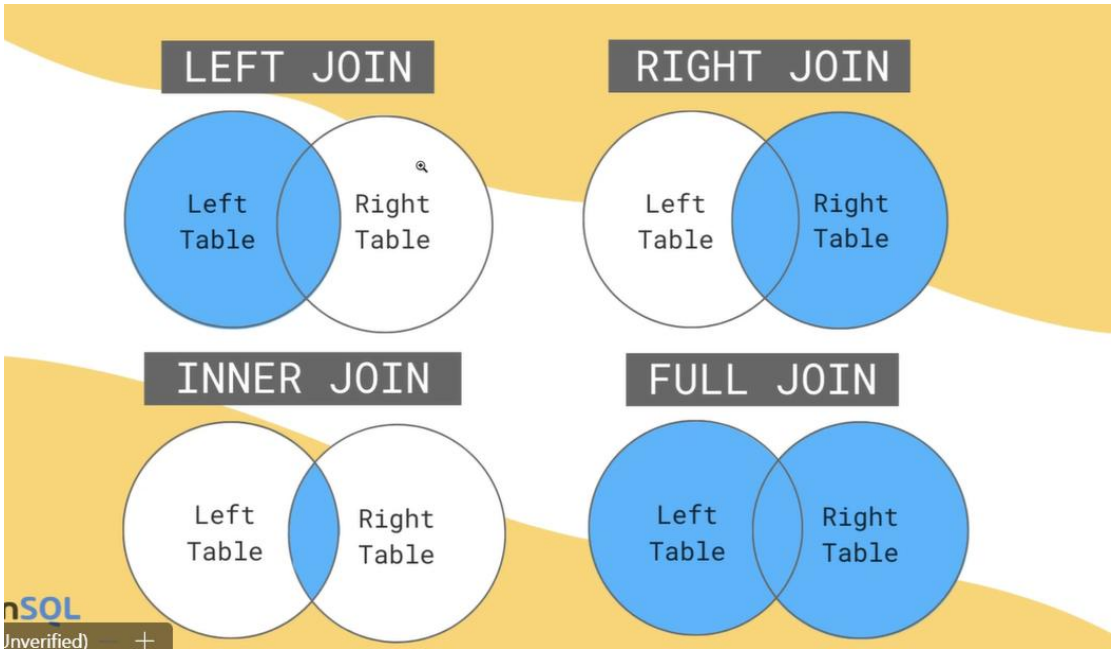
```
SELECT c.customer_name, o.product_name
FROM customers c
LEFT JOIN orders o ON c.customer_id = o.customer_id;
```

RIGHT JOIN

SELECT o.product_name, c.customer_name

FROM customers c

RIGHT JOIN orders o ON c.customer_id = o.customer_id;



6. Join Filters and Grouping:

Filtered Join

SELECT c.customer_name, o.product_name, o.order_amount

FROM customers c

JOIN orders o ON c.customer_id = o.customer_id

WHERE o.order_amount > 5000;

Grouping: Total Orders by Customer

SELECT c.customer_name, COUNT(o.order_id) AS total_orders

FROM customers c

JOIN orders o ON c.customer_id = o.customer_id

GROUP BY c.customer_name

HAVING total_orders > 1;

Total Amount Spent by Customer

```
SELECT c.customer_name, SUM(o.order_amount) AS total_spent
FROM customers c
JOIN orders o ON c.customer_id = o.customer_id
GROUP BY c.customer_name;
```

Customers with No Orders

```
SELECT c.customer_name
FROM customers c
LEFT JOIN orders o ON c.customer_id = o.customer_id
WHERE o.order_id IS NULL;
```

Order Count by City

```
SELECT c.city, COUNT(o.order_id) AS order_count
FROM customers c
JOIN orders o ON c.customer_id = o.customer_id
GROUP BY c.city;
```