7/17/25

Role Specific Training- Data Engineering

Day-01

Advanced SQL

1.Domain Driven Design:

Designing a schema based on Domain. e.g., educational system- Student, Teacher, Course….

Basic SELECT SQL Queries:

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

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'),
(5, 'Esabella', 'Data Engineering', '2025-07-18')
--SELECT clause
SELECT * FROM students;

SELECT name, course FROM students;

SELECT * FROM students WHERE course = 'Data Engineering';
-- WHERE clause variations
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 '2025-07-17' AND '2025-07-15';


-- Pattern matching – LIKE

SELECT * FROM students WHERE name LIKE 'A%';


SELECT * FROM students WHERE name LIKE '%a';


SELECT * FROM students WHERE name LIKE '%a%';


-- UPDATE clause


UPDATE students

SET course = 'Advanced Data Engineering'

WHERE student_id = 1;


UPDATE students

SET join_date = '2025-09-20'

WHERE name = 'Bala';

-- Updating the date by 1

UPDATE students

SET join_date = ADDDATE ("2017-06-15", INTERVAL 1 DAY);
```

```
-- DELETE clause
DELETE FROM students
WHERE student_id = 2;

DELETE FROM students
WHERE join_date < '2025-09-16';
--isactive – false:
```

Make a retired employee inactive from the database.

2. SUBQUERY:

2.1. Inline Query:

```
CREATE DATABASE simple_sql;
USE simple_sql;
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);
```

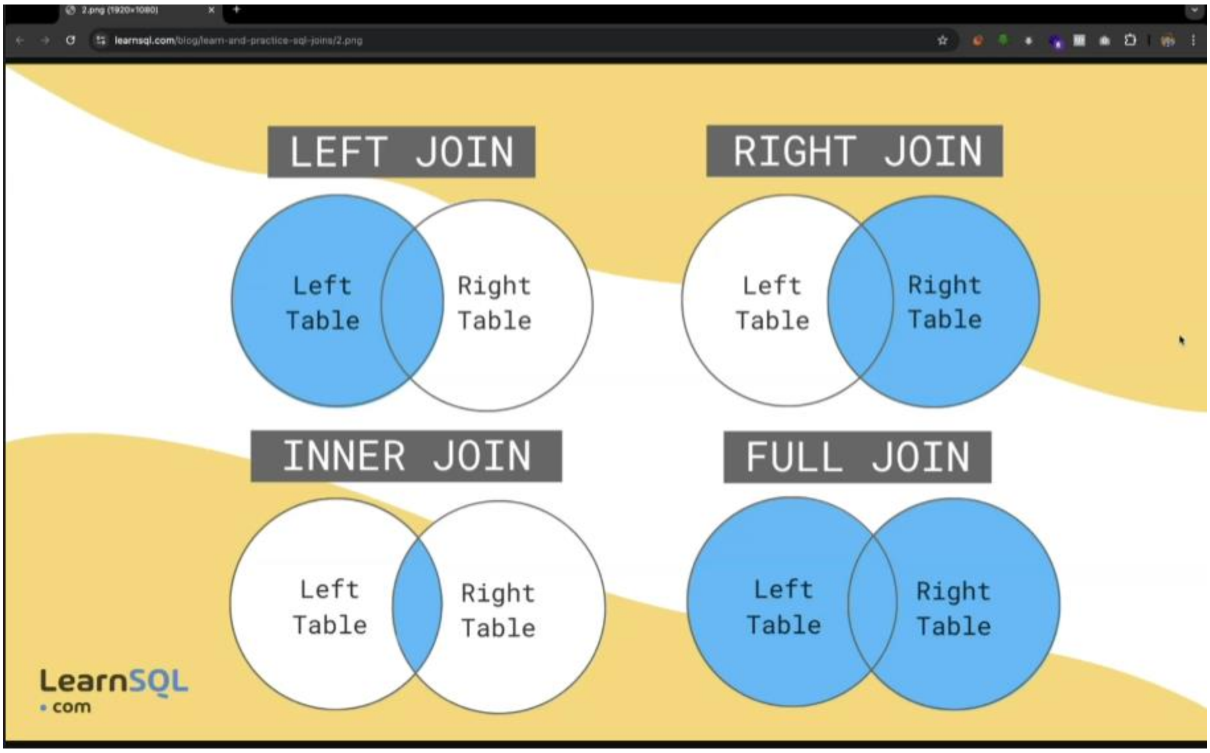
```
SELECT * FROM employees

WHERE salary > (

        SELECT AVG(salary) FROM employees

);


--- Show department-wise average salary using a derived table:

SELECT dept_avg.department , dept_avg.avg_salary

FROM (

        SELECT department, AVG(salary) AS avg_salary

        FROM employees

        GROUP BY department

) AS dept_avg;


---Analytic function: RANK()

--Show employees with their rank based on salary (highest first)

SELECT emp_name, department, salary,

        RANK() OVER (ORDER BY salary DESC) AS salary_rank

FROM employees;
```

JOINS OPERATIONS

```
-- creating customer table to play with joins

use analytics_practice;

CREATE TABLE customers (

  customer_id INT PRIMARY KEY,

  customer_name VARCHAR(100),

  city VARCHAR(50)

);


INSERT INTO customers VALUES
```

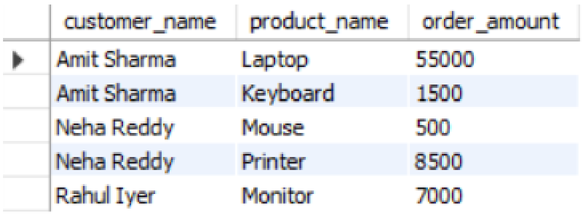
```
(1, 'Amit Sharma', 'Delhi'),
(2, 'Neha Reddy', 'Hyderabad'),
(3, 'Rahul Iyer', 'Mumbai'),
(4, 'Divya Mehta', 'Chennai');

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);
```

JOINS TYPES:



i. INNER JOIN:

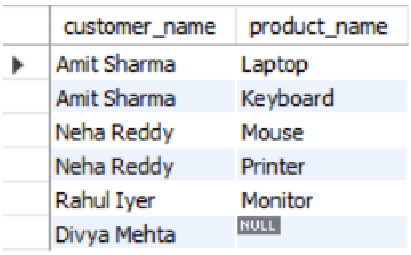
```
SELECT c.customer_name, o.product_name, o.order_amount

FROM customers c

INNER JOIN orders o

ON c.customer_id = o.customer_id;
```



customer_name	product_name	order_amount
Amit Sharma	Laptop	55000
Amit Sharma	Keyboard	1500
Neha Reddy	Mouse	500
Neha Reddy	Printer	8500
Rahul Iyer	Monitor	7000

ii. LEFT JOIN:

```
SELECT c.customer_name, o.product_name

FROM customers c

LEFT JOIN orders o

ON c.customer_id = o.customer_id;
```



customer_name	product_name
Amit Sharma	Laptop
Amit Sharma	Keyboard
Neha Reddy	Mouse
Neha Reddy	Printer
Rahul Iyer	Monitor
Divya Mehta	NULL

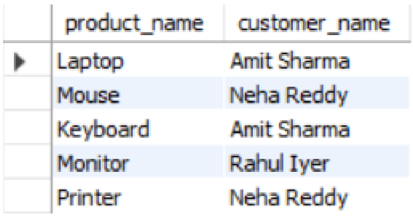
iii. RIGHT JOIN

SELECT o.product_name, c.customer_name

FROM customers c

RIGHT JOIN orders o

ON c.customer_id = o.customer_id;



product_name	customer_name
Laptop	Amit Sharma
Mouse	Neha Reddy
Keyboard	Amit Sharma
Monitor	Rahul Iyer
Printer	Neha Reddy

-- Applying Filtering on JOIN Operations

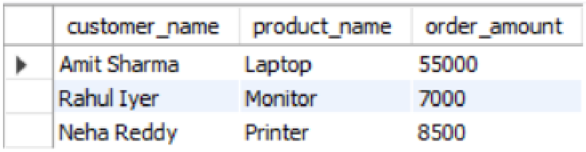
SELECT c.customer_name, o.product_name, o.order_amount

FROM customers c

JOIN orders o

ON c.customer_id = o.customer_id

WHERE o.order_amount > 5000;



customer_name	product_name	order_amount
Amit Sharma	Laptop	55000
Rahul Iyer	Monitor	7000
Neha Reddy	Printer	8500

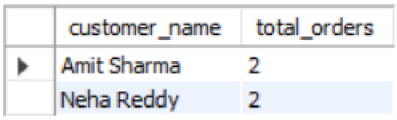
SELECT c.customer_name, COUNT(o.order_id) AS total_orders

FROM customers c

JOIN orders o ON c.customer_id = o.customer_id

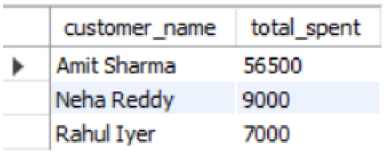
GROUP BY c.customer_name

```
HAVING total_orders > 1;
```



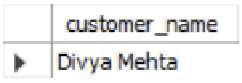
customer_name	total_orders
Amit Sharma	2
Neha Reddy	2

```
-- Total amount 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 ;
```



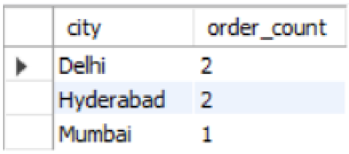
customer_name	total_spent
Amit Sharma	56500
Neha Reddy	9000
Rahul Iyer	7000

```
-- Customers who havent placed any 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;
```



customer_name
Divya Mehta

```
-- Group data based on city with order_count

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 city;
```



city	order_count
Delhi	2
Hyderabad	2
Mumbai	1