Experiment 2

Student Name: Suyash
UID: 25MCI10054
Branch: MCA(AI/ML)
Section/Group: 25MAM-1
Semester: 2nd
Date of Performance: 13/1/2026
Subject Name: Technical Training
Subject Code: 25CAP-652

1. Aim :

To implement and analyze SQL SELECT queries using filtering, sorting, grouping, and aggregation concepts in PostgreSQL for efficient data retrieval and analytical reporting

2. Tools Used

- PostgreSQL

3. Objective:

- To retrieve specific data using filtering conditions
- To sort query results using single and multiple attributes
- To perform aggregation using grouping techniques
- To apply conditions on aggregated data
- To understand real-world analytical queries commonly asked in placement interviews

4. Practical / Experiment Steps

Step 1: Database and Table Preparation

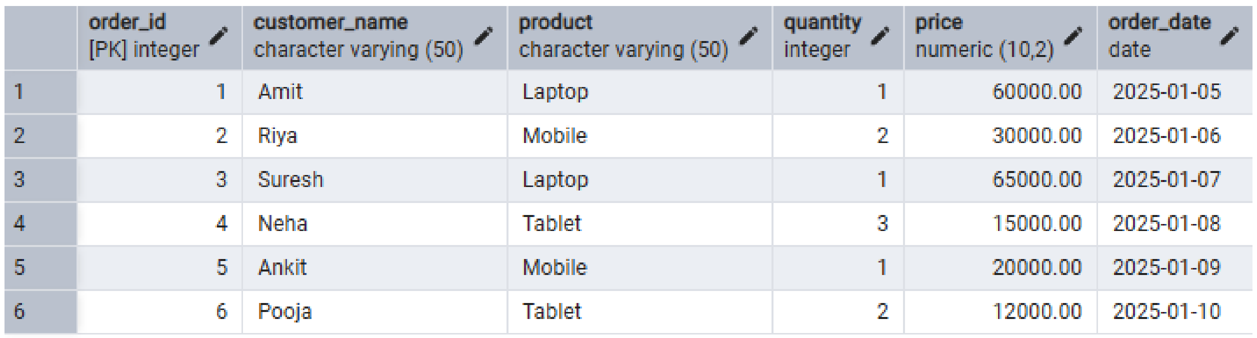
- Start the PostgreSQL server.
- Open the PostgreSQL client tool.
- Create a database for the experiment.
- Prepare a sample table representing customer orders containing details such as customer name, product, quantity, price, and order date.

- Insert sufficient sample records to allow meaningful analysis.

Query :

```
CREATE TABLE customer_orders (
    order_id SERIAL PRIMARY KEY,
    customer_name VARCHAR(50),
    product VARCHAR(50),
    quantity INT,
    price NUMERIC(10,2),
    order_date DATE
);
INSERT INTO customer_orders (customer_name, product, quantity, price, order_date) VALUES
('Amit', 'Laptop', 1, 60000, '2025-01-05'),
('Riya', 'Mobile', 2, 30000, '2025-01-06'),
('Suresh', 'Laptop', 1, 65000, '2025-01-07'),
('Neha', 'Tablet', 3, 15000, '2025-01-08'),
('Ankit', 'Mobile', 1, 20000, '2025-01-09'),
('Pooja', 'Tablet', 2, 12000, '2025-01-10');
```

Output



	order_id [PK] integer	customer_name character varying (50)	product character varying (50)	quantity integer	price numeric (10,2)	order_date date
1	1	Amit	Laptop	1	60000.00	2025-01-05
2	2	Riya	Mobile	2	30000.00	2025-01-06
3	3	Suresh	Laptop	1	65000.00	2025-01-07
4	4	Neha	Tablet	3	15000.00	2025-01-08
5	5	Ankit	Mobile	1	20000.00	2025-01-09
6	6	Pooja	Tablet	2	12000.00	2025-01-10

Step 2: Filtering Data Using Conditions

- Execute data retrieval operations to display only those records that satisfy specific conditions, such as higher-priced orders.

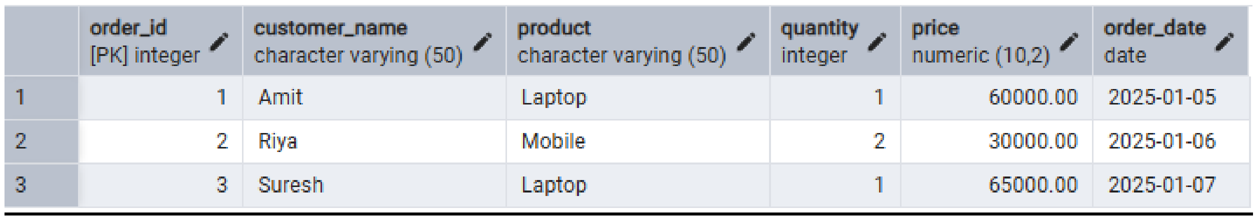
<u>Query(Without case Statment)</u>

```
SELECT *

FROM customer_orders

WHERE price > 25000;
```

<u>Query with case statement:</u>

```
SELECT *

FROM customer_orders

WHERE

   CASE

      WHEN Price > 25000 THEN 1

      ELSE 0

   END = 1;
```

<u>Output</u>



	order_id [PK] integer	customer_name character varying (50)	product character varying (50)	quantity integer	price numeric (10,2)	order_date date
1	1	Amit	Laptop	1	60000.00	2025-01-05
2	2	Riya	Mobile	2	30000.00	2025-01-06
3	3	Suresh	Laptop	1	65000.00	2025-01-07

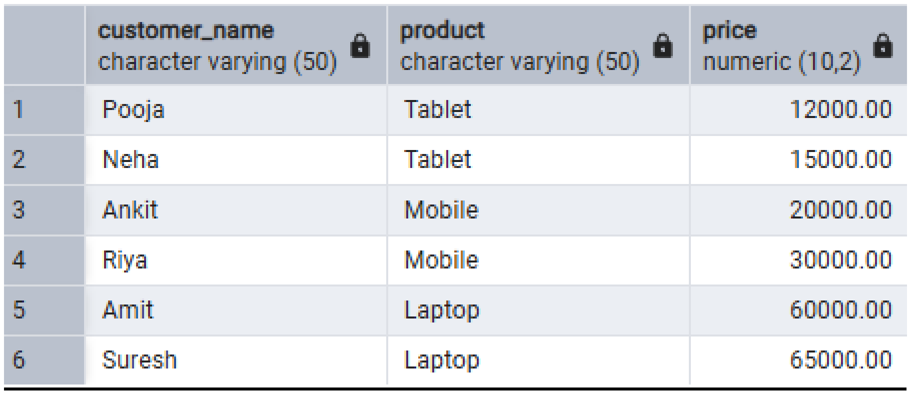
Step 3: Sorting Query Results

- Retrieve selected columns from the table and arrange the output based on numerical values such as price.
- Perform sorting using both ascending and descending order.
- Apply sorting on more than one attribute to understand priority-based ordering.

Sort by Price (Ascending)

```
SELECT customer_name, product, price
FROM customer_orders
ORDER BY price ASC;
```

Output

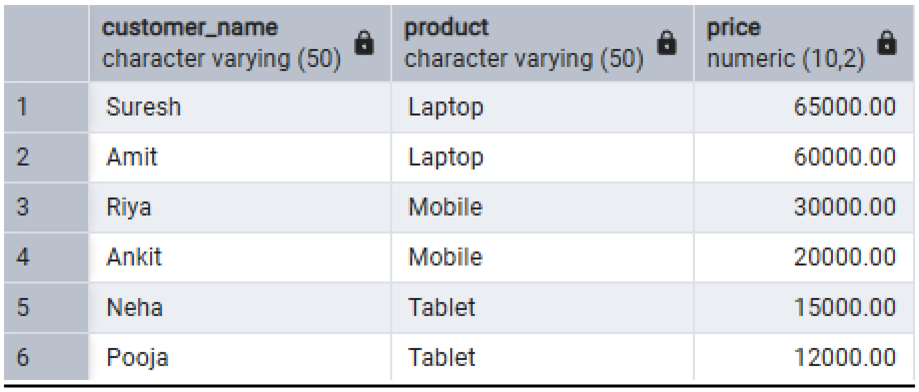


	customer_name character varying (50)	product character varying (50)	price numeric (10,2)
1	Pooja	Tablet	12000.00
2	Neha	Tablet	15000.00
3	Ankit	Mobile	20000.00
4	Riya	Mobile	30000.00
5	Amit	Laptop	60000.00
6	Suresh	Laptop	65000.00

Sort by Price (Descending)

```
SELECT customer_name, product, price
FROM customer_orders
ORDER BY price DESC;
```

Output

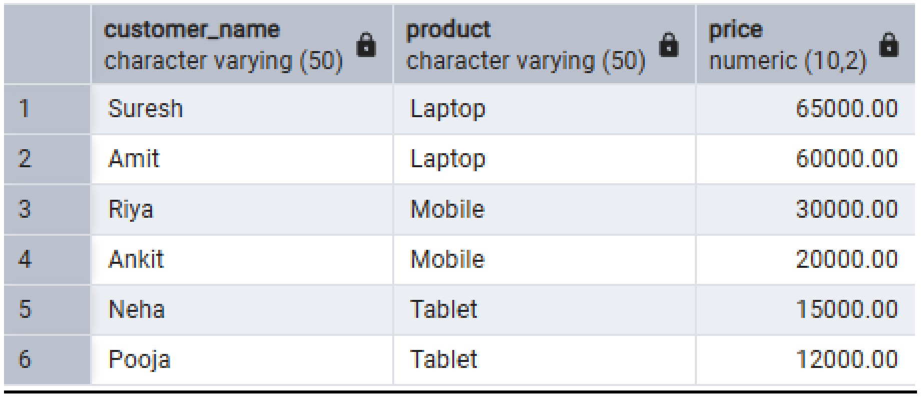


	customer_name character varying (50)	product character varying (50)	price numeric (10,2)
1	Suresh	Laptop	65000.00
2	Amit	Laptop	60000.00
3	Riya	Mobile	30000.00
4	Ankit	Mobile	20000.00
5	Neha	Tablet	15000.00
6	Pooja	Tablet	12000.00

Sort by Product, then Price

```
SELECT customer_name, product, price
FROM customer_orders
ORDER BY product ASC, price DESC;
```

Output



	customer_name character varying (50)	product character varying (50)	price numeric (10,2)
1	Suresh	Laptop	65000.00
2	Amit	Laptop	60000.00
3	Riya	Mobile	30000.00
4	Ankit	Mobile	20000.00
5	Neha	Tablet	15000.00
6	Pooja	Tablet	12000.00

Step 4: Grouping Data for Aggregation

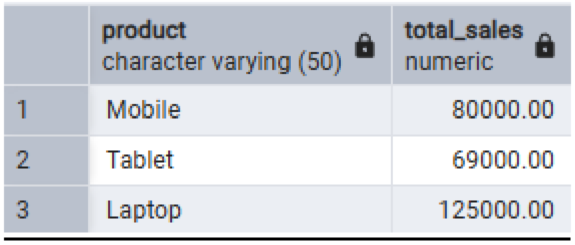
- Group records based on a common attribute such as product.
- Calculate aggregate values like total sales for each group.
- Analyze how multiple rows are combined into summarized results.

Total Sales Per Product

Query:

```
SELECT product, SUM(price * quantity) AS total_sales

FROM customer_orders

GROUP BY product;
```

Output



	product character varying (50)	total_sales numeric
1	Mobile	80000.00
2	Tablet	69000.00
3	Laptop	125000.00

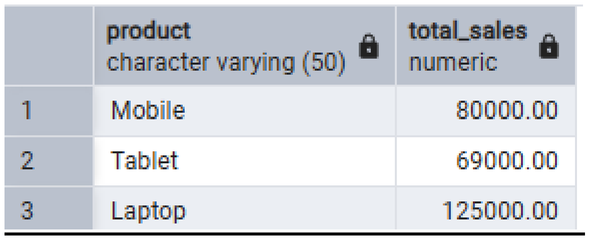
Step 5: Applying Conditions on Aggregated Data

- Apply conditions on grouped results to retrieve only those groups that satisfy specific aggregate criteria.
- Compare the difference between row-level filtering and group-level filtering.

Query:

```
SELECT product, SUM(price * quantity) AS total_sales
FROM customer_orders
GROUP BY product
HAVING SUM(price * quantity) > 50000;
```

Output



	product character varying (50)	total_sales numeric
1	Mobile	80000.00
2	Tablet	69000.00
3	Laptop	125000.00

Step 6: Conceptual Understanding of Filtering vs Aggregation Conditions

- Analyze scenarios where conditions are incorrectly applied before grouping.
- Correctly apply conditions after grouping to avoid logical errors.

Incorrect usage:

Querry:

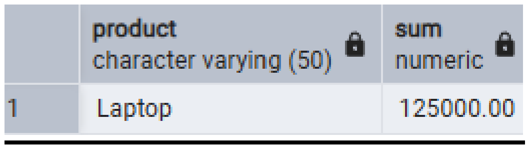
```
SELECT product, SUM(price)
FROM customer_orders
WHERE SUM(price) > 50000
GROUP BY product;
```

Correct Usage:

Querry:

```
SELECT product, SUM(price)
FROM customer_orders
GROUP BY product
HAVING SUM(price) > 50000;
```

Output



	product character varying (50)	sum numeric
1	Laptop	125000.00

8. Learning Output

- Understand how to create relational database tables using appropriate data types and constraints
- Learn to retrieve required data from a table using **row-level filtering** with the WHERE clause.
- Gain the ability to apply **column-level (group-level) filtering** using the HAVING clause.
- Develop practical knowledge of using **CASE statements** for conditional logic in SQL queries.
- Understand the use of **aggregate functions** such as SUM(), AVG(), and COUNT() for analytical reporting.
- Clearly differentiate between **row-level filtering and group-level filtering**, and apply them correctly in real-world SQL scenarios.