Branch: MCA (Data Science)	**Semester:** 2nd
Student Name: Chaitanya Sharma	**UID:** 25MCD10056
Subject Name: Technical Training - I Lab	**Subject Code**: 25CAP-652
Section/Group: 25MCD-1(A)	**Date of Performance:** 21-Jan-2026

Experiment No.: 2

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. **S/W Requirement:** Oracle Database Express Edition and PGAdmin

3. **Objectives:** Implementation of SELECT Queries with Filtering, Grouping and Sorting in PostgreSQL
   - 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. **Task to be done:**

   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.
   - Purpose: To create a realistic dataset for performing analytical queries.

Step 2: Filtering Data Using Conditions

- Execute data retrieval operations to display only those records that satisfy specific conditions, such as higher-priced orders.
- Observe how filtering limits the number of rows returned.

Observation: Filtering reduces unnecessary data processing and improves query efficiency.

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.

Observation: Sorting is essential for reports, rankings, and ordered displays.

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.

Observation: Grouping transforms transactional data into analytical insights.

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.

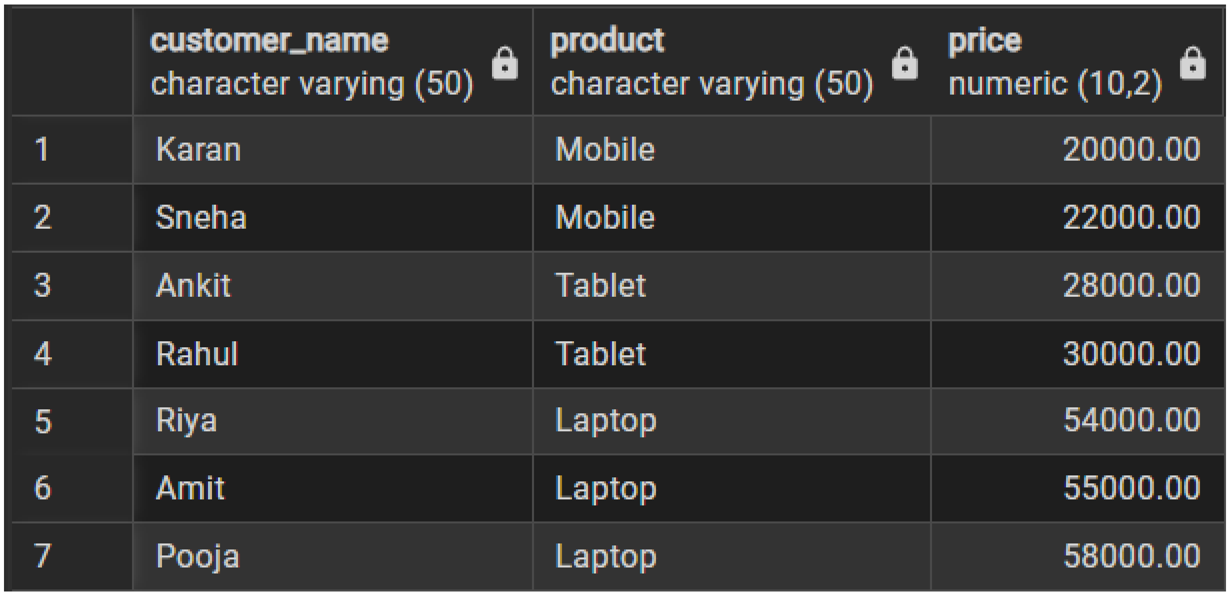
Observation: Conditions applied after grouping allow refined analytical reporting.

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.

Observation: Understanding execution order prevents common SQL mistakes frequently tested in interviews.

4. Dataset:



	customer_name character varying (50)	product character varying (50)	price numeric (10,2)
1	Karan	Mobile	20000.00
2	Sneha	Mobile	22000.00
3	Ankit	Tablet	28000.00
4	Rahul	Tablet	30000.00
5	Riya	Laptop	54000.00
6	Amit	Laptop	55000.00
7	Pooja	Laptop	58000.00

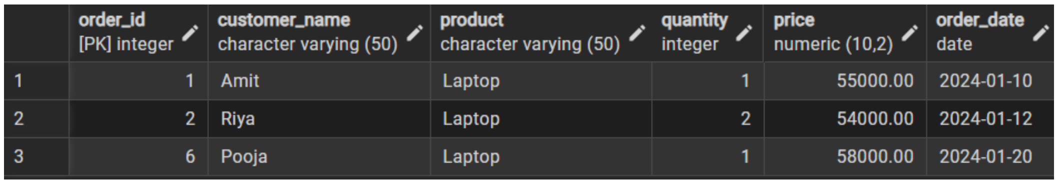
5. Code:

```
-- Step 1

   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, 55000, '2024-01-10'),

('Riya', 'Laptop', 2, 54000, '2024-01-12'),

('Karan', 'Mobile', 3, 20000, '2024-01-15'),

('Sneha', 'Mobile', 1, 22000, '2024-01-16'),

('Rahul', 'Tablet', 2, 30000, '2024-01-18'),

('Pooja', 'Laptop', 1, 58000, '2024-01-20'),

('Ankit', 'Tablet', 3, 28000, '2024-01-22');

-- Step 2:

SELECT *

FROM customer_orders

WHERE price > 30000;

--Step 3:

SELECT customer_name, product, price

FROM customer_orders

WHERE price > 20000

ORDER BY price ASC;

SELECT customer_name, product, price

FROM customer_orders

WHERE price > 20000

ORDER BY price DESC;
```

```
-- Step 4:
SELECT *
FROM customer_orders
WHERE price > 30000;
-- Step 5:
SELECT product,
    SUM(quantity * price) AS total_sales
FROM customer_orders
GROUP BY product
HAVING SUM(quantity * price) > 100000;
-- Step 6:
select product, sum(quantity*price) as total_revenue
from customer_orders
where order_date >= '2024-01-15'
group by product
having sum(quantity*price) > 50000;
```

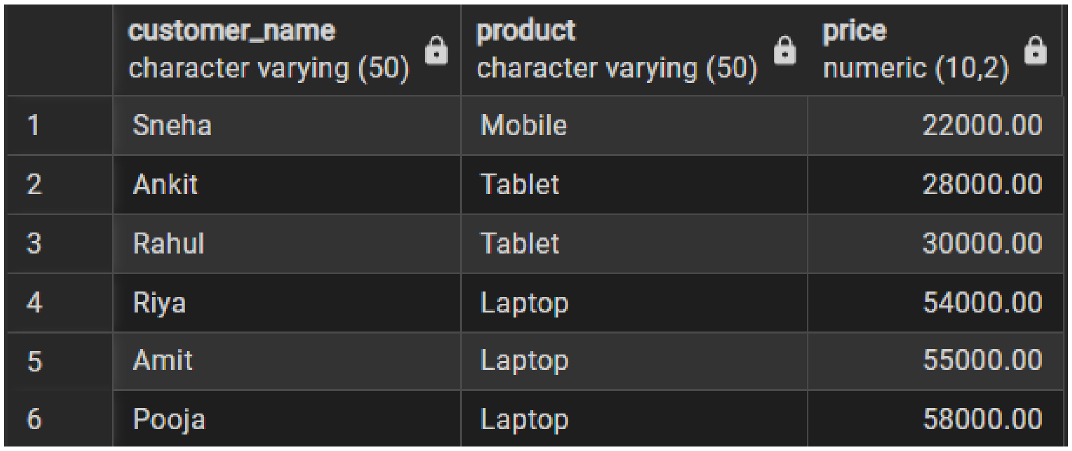
- **Output:**
  - **Step-2:**



	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	55000.00	2024-01-10
2	2	Riya	Laptop	2	54000.00	2024-01-12
3	6	Pooja	Laptop	1	58000.00	2024-01-20

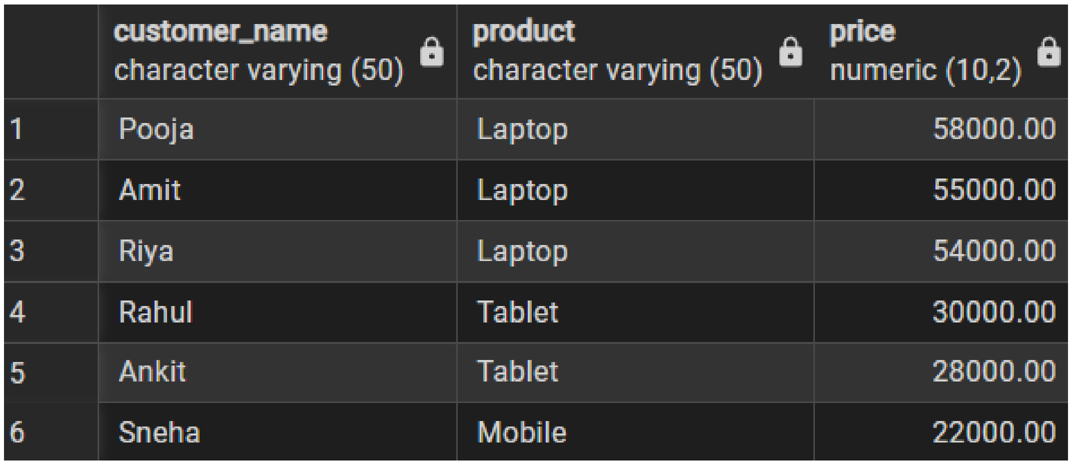
  - **Step-3:**

Order by ASC:



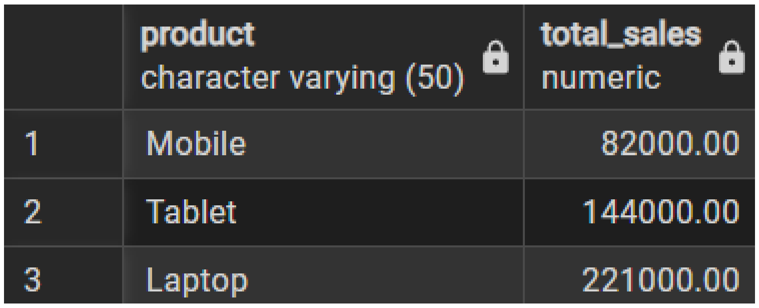
	customer_name character varying (50)	product character varying (50)	price numeric (10,2)
1	Sneha	Mobile	22000.00
2	Ankit	Tablet	28000.00
3	Rahul	Tablet	30000.00
4	Riya	Laptop	54000.00
5	Amit	Laptop	55000.00
6	Pooja	Laptop	58000.00

Order by DESC:



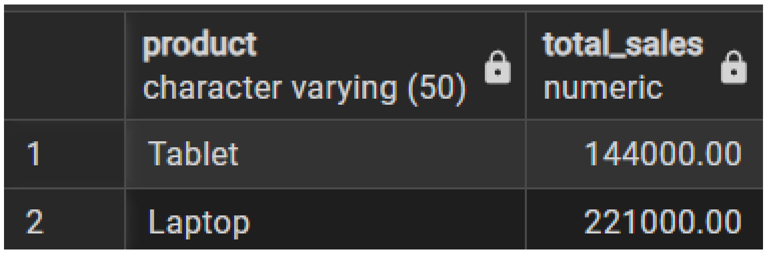
	customer_name character varying (50)	product character varying (50)	price numeric (10,2)
1	Pooja	Laptop	58000.00
2	Amit	Laptop	55000.00
3	Riya	Laptop	54000.00
4	Rahul	Tablet	30000.00
5	Ankit	Tablet	28000.00
6	Sneha	Mobile	22000.00

- **Step-4:**



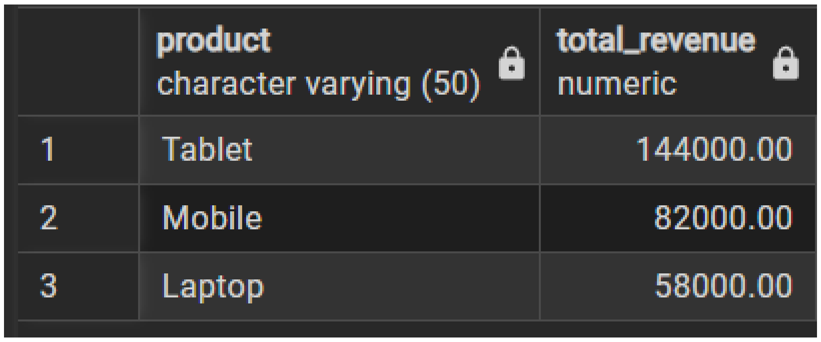
	product character varying (50)	total_sales numeric
1	Mobile	82000.00
2	Tablet	144000.00
3	Laptop	221000.00

- **Step-5:**



	product character varying (50)	total_sales numeric
1	Tablet	144000.00
2	Laptop	221000.00

- **Step-6:**



	product character varying (50)	total_revenue numeric
1	Tablet	144000.00
2	Mobile	82000.00
3	Laptop	58000.00

- **Learning Outcomes:**
  - Students understand how data can be filtered to retrieve only relevant records from a database.
  - Students learn how sorting improves readability and usefulness of query results in reports.
  - Students gain the ability to group data for analytical purposes.
  - Students clearly differentiate between row-level conditions and group-level conditions.
  - Students develop confidence in writing analytical SQL queries used in real-world scenarios.
  - Students are better prepared to answer SQL-based placement and interview questions related to filtering, grouping, and aggregation.