Worksheet No- 2

Name - Sahil Gupta **UID – 25MCI10266**

Branch – MCA **Section – MAM-1(A)**

Semester – 2nd **Date of performance – 12th Jan,2026**

Subject - Technical Training **Subject Code -**

Aim / Overview of the Practical

To understand and implement the CASE statement in PostgreSQL for conditional data aggregation and analysis.

The CASE statement in PostgreSQL is used to perform conditional logic similar to if-else conditions. It is commonly used in SELECT queries to apply conditions while retrieving or aggregating data.

Software Requirements

- PostgreSQL
  - PostgreSQL (Database Server)
  - pgAdmin (Graphical User Interface)
  - SQL Queries (DDL, DML, DCL commands)

Objectives

- 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

Procedure of the experiments

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.

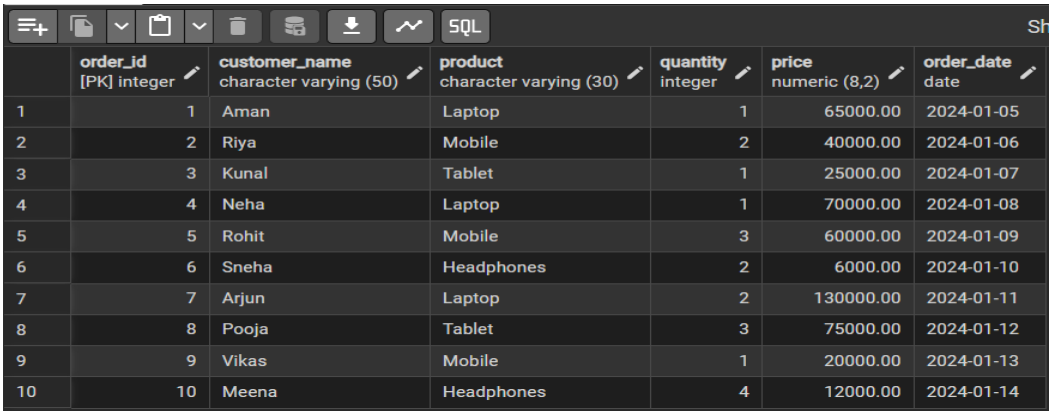
Practical / Experiment Steps

```
-- Create Orders table representing customer orders
CREATE TABLE Orders (
    order_id INT PRIMARY KEY,
    customer_name VARCHAR(50),
    product VARCHAR(30),
    quantity INT,
    price DECIMAL(8,2),
    order_date DATE
);

-- Insert sufficient sample records
INSERT INTO Orders VALUES
(1, 'Aman', 'Laptop', 1, 65000, '2024-01-05'),
(2, 'Riya', 'Mobile', 2, 40000, '2024-01-06'),
(3, 'Kunal', 'Tablet', 1, 25000, '2024-01-07'),
(4, 'Neha', 'Laptop', 1, 70000, '2024-01-08'),
(5, 'Rohit', 'Mobile', 3, 60000, '2024-01-09'),
(6, 'Sneha', 'Headphones', 2, 6000, '2024-01-10'),
(7, 'Arjun', 'Laptop', 2, 130000, '2024-01-11'),
(8, 'Pooja', 'Tablet', 3, 75000, '2024-01-12'),
(9, 'Vikas', 'Mobile', 1, 20000, '2024-01-13'),
(10, 'Meena', 'Headphones', 4, 12000, '2024-01-14');
```

-- View table data

```
SELECT * FROM Orders;
```



	order_id [PK] integer	customer_name character varying (50)	product character varying (30)	quantity integer	price numeric (8,2)	order_date date
1	1	Aman	Laptop	1	65000.00	2024-01-05
2	2	Riya	Mobile	2	40000.00	2024-01-06
3	3	Kunal	Tablet	1	25000.00	2024-01-07
4	4	Neha	Laptop	1	70000.00	2024-01-08
5	5	Rohit	Mobile	3	60000.00	2024-01-09
6	6	Sneha	Headphones	2	6000.00	2024-01-10
7	7	Arjun	Laptop	2	130000.00	2024-01-11
8	8	Pooja	Tablet	3	75000.00	2024-01-12
9	9	Vikas	Mobile	1	20000.00	2024-01-13
10	10	Meena	Headphones	4	12000.00	2024-01-14

-- Step 2: Filtering Data Using Conditions

-- Retrieve orders with price greater than 50,000

```
SELECT * FROM Orders
WHERE price > 50000;
```

	order_id [PK] integer	customer_name character varying (50)	product character varying (30)	quantity integer	price numeric (8,2)	order_date date
1	1	Aman	Laptop	1	65000.00	2024-01-05
2	4	Neha	Laptop	1	70000.00	2024-01-08
3	5	Rohit	Mobile	3	60000.00	2024-01-09
4	7	Arjun	Laptop	2	130000.00	2024-01-11
5	8	Pooja	Tablet	3	75000.00	2024-01-12

-- Step 3: Sorting Query Results

-- Sort orders by price in ascending order

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

	customer_name character varying (50)	product character varying (30)	price numeric (8,2)
1	Sneha	Headphones	6000.00
2	Meena	Headphones	12000.00
3	Vikas	Mobile	20000.00
4	Kunal	Tablet	25000.00
5	Riya	Mobile	40000.00
6	Rohit	Mobile	60000.00
7	Aman	Laptop	65000.00
8	Neha	Laptop	70000.00
9	Pooja	Tablet	75000.00
10	Arjun	Laptop	130000.00

--Sorting in descending order

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

	customer_name character varying (50)	product character varying (30)	price numeric (8,2)
1	Arjun	Laptop	130000.00
2	Pooja	Tablet	75000.00
3	Neha	Laptop	70000.00
4	Aman	Laptop	65000.00
5	Rohit	Mobile	60000.00
6	Riya	Mobile	40000.00
7	Kunal	Tablet	25000.00
8	Vikas	Mobile	20000.00
9	Meena	Headphones	12000.00
10	Sneha	Headphones	6000.00

-- Sort using multiple attributes (product, then price)

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

	customer_name character varying (50)	product character varying (30)	price numeric (8,2)
1	Meena	Headphones	12000.00
2	Sneha	Headphones	6000.00
3	Arjun	Laptop	130000.00
4	Neha	Laptop	70000.00
5	Aman	Laptop	65000.00
6	Rohit	Mobile	60000.00
7	Riya	Mobile	40000.00
8	Vikas	Mobile	20000.00
9	Pooja	Tablet	75000.00
10	Kunal	Tablet	25000.00

-- Step 4: Grouping Data for Aggregation

-- Calculate total sales for each product

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

FROM Orders
```

GROUP BY product;

	product character varying (30)	total_sales numeric
1	Mobile	120000.00
2	Tablet	100000.00
3	Laptop	265000.00
4	Headphones	18000.00

-- Step 5: Applying Conditions on Aggregated Data

-- Retrieve products having total sales greater than 80,000

```
SELECT product,
SUM(price) AS total_sales
FROM Orders
GROUP BY product
HAVING SUM(price) > 80000;
```

	product character varying (30)	total_sales numeric
1	Mobile	120000.00
2	Tablet	100000.00
3	Laptop	265000.00

-- Step 6: Conceptual Understanding of Filtering vs Aggregation Conditions

-- Incorrect query: using WHERE with aggregate function (logical error)

```
SELECT product,
SUM(price) AS total_sales
FROM Orders
WHERE price > 30000        -- Row-level filtering
GROUP BY product
HAVING SUM(price) > 80000;   -- Group-level filtering
```

	product character varying (30)	total_sales numeric
1	Mobile	100000.00
2	Laptop	265000.00

5. Learning Outcome

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