



**Student Name:** Aayush Raj  
**Branch:** M.C.A(A.I & M.L)  
**Semester:** 2<sup>nd</sup>  
**Subject Name:** TECHINICAL SKILLS

**UID:** 25MCI10066  
**Section/Group:** 25MAM-1 A  
**Date of Performance:** 12/01/2026  
**Subject Code:** 25CAP-652

## WORKSHEET 2

**AIM:** To design and implement a sample database system using DDL, DML, and DCL commands, including database creation, data manipulation, schema modification, and role-based access control to ensure data integrity and secure, read-only access for authorized users.

**S/W Requirement:** Oracle Database Express Edition and pgAdmin

### **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

Given:

### **Practical / Experiment Steps**

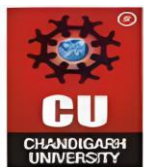
#### **Step 1: Database and Table Preparation**

- 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:

Table Creation:

```
CREATE TABLE CustomerOrders (  
    OrderID SERIAL PRIMARY KEY,  
    CustomerName VARCHAR(50),  
    Product VARCHAR(50),
```



Quantity INT,

Price DECIMAL(10,2),

OrderDate DATE

);

Insertion Of Records

INSERT INTO CustomerOrders

(CustomerName, Product, Quantity, Price, OrderDate)

VALUES

('Aarav', 'Laptop', 1, 65000, '2025-01-10'),

('Neha', 'Mobile', 2, 40000, '2025-01-12'),

('Rohit', 'Laptop', 1, 70000, '2025-01-15'),

('Priya', 'Tablet', 3, 45000, '2025-01-18'),

('Karan', 'Mobile', 1, 20000, '2025-01-20'),

('Simran', 'Laptop', 2, 130000, '2025-01-22'),

('Aman', 'Tablet', 2, 30000, '2025-01-25'),

('Riya', 'Mobile', 3, 60000, '2025-01-26'),

('Vikas', 'Laptop', 1, 68000, '2025-01-28'),

('Pooja', 'Tablet', 1, 15000, '2025-01-30');

Output:

	orderid [PK] integer	customername character varying (50)	product character varying (50)	quantity integer	price numeric (10,2)	orderdate date
1	1	Aarav	Laptop	1	65000.00	2025-01-10
2	2	Neha	Mobile	2	40000.00	2025-01-12
3	3	Rohit	Laptop	1	70000.00	2025-01-15
4	4	Priya	Tablet	3	45000.00	2025-01-18
5	5	Karan	Mobile	1	20000.00	2025-01-20
6	6	Simran	Laptop	2	130000.00	2025-01-22
7	7	Aman	Tablet	2	30000.00	2025-01-25
8	8	Riya	Mobile	3	60000.00	2025-01-26
9	9	Vikas	Laptop	1	68000.00	2025-01-28
10	10	Pooja	Tablet	1	15000.00	2025-01-30

## Step 2: Filtering Data Using Conditions

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

Query(Without case Statment)

```
SELECT *  
  
FROM CustomerOrders  
  
WHERE Price > 50000;
```

Query with case statement:

```
SELECT *  
  
FROM CustomerOrders  
  
WHERE  
  
CASE  
  
    WHEN Price > 50000 THEN 1  
  
    ELSE 0  
  
END = 1;
```

Output:

	orderid [PK] integer	customername character varying (50)	product character varying (50)	quantity integer	price numeric (10,2)	orderdate date
1	1	Aarav	Laptop	1	65000.00	2025-01-10
2	3	Rohit	Laptop	1	70000.00	2025-01-15
3	6	Simran	Laptop	2	130000.00	2025-01-22
4	8	Riya	Mobile	3	60000.00	2025-01-26
5	9	Vikas	Laptop	1	68000.00	2025-01-28

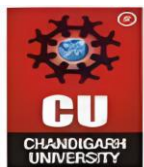
## Step 3: Sorting Query Results

- 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.

Query:

**Sort orders by price in ascending order**

```
SELECT *
```



FROM CustomerOrders

ORDER BY Price ASC;

Output:

	orderid [PK] integer	customename character varying (50)	product character varying (50)	quantity integer	price numeric (10,2)	orderdate date
1	10	Pooja	Tablet	1	15000.00	2025-01-30
2	5	Karan	Mobile	1	20000.00	2025-01-20
3	7	Aman	Tablet	2	30000.00	2025-01-25
4	2	Neha	Mobile	2	40000.00	2025-01-12
5	4	Priya	Tablet	3	45000.00	2025-01-18
6	8	Riya	Mobile	3	60000.00	2025-01-26
7	1	Aarav	Laptop	1	65000.00	2025-01-10
8	9	Vikas	Laptop	1	68000.00	2025-01-28
9	3	Rohit	Laptop	1	70000.00	2025-01-15
10	6	Simran	Laptop	2	130000.00	2025-01-22

Sort orders by price in descending order

Query:

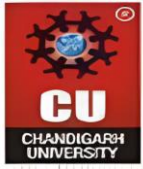
SELECT \*

FROM CustomerOrders

ORDER BY Price DESC;

Output

	orderid [PK] integer	customename character varying (50)	product character varying (50)	quantity integer	price numeric (10,2)	orderdate date
1	6	Simran	Laptop	2	130000.00	2025-01-22
2	3	Rohit	Laptop	1	70000.00	2025-01-15
3	9	Vikas	Laptop	1	68000.00	2025-01-28
4	1	Aarav	Laptop	1	65000.00	2025-01-10
5	8	Riya	Mobile	3	60000.00	2025-01-26
6	4	Priya	Tablet	3	45000.00	2025-01-18
7	2	Neha	Mobile	2	40000.00	2025-01-12
8	7	Aman	Tablet	2	30000.00	2025-01-25
9	5	Karan	Mobile	1	20000.00	2025-01-20
10	10	Pooja	Tablet	1	15000.00	2025-01-30



## Sort using multiple columns

### Query:

```
SELECT *
```

```
FROM CustomerOrders ORDER BY Product ASC, Price DESC;
```

### Output:

	orderid [PK] integer	customername character varying (50)	product character varying (50)	quantity integer	price numeric (10,2)	orderdate date
1	6	Simran	Laptop	2	130000.00	2025-01-22
2	3	Rohit	Laptop	1	70000.00	2025-01-15
3	9	Vikas	Laptop	1	68000.00	2025-01-28
4	1	Aarav	Laptop	1	65000.00	2025-01-10
5	8	Riya	Mobile	3	60000.00	2025-01-26
6	2	Neha	Mobile	2	40000.00	2025-01-12
7	5	Karan	Mobile	1	20000.00	2025-01-20
8	4	Priya	Tablet	3	45000.00	2025-01-18
9	7	Aman	Tablet	2	30000.00	2025-01-25
10	10	Pooja	Tablet	1	15000.00	2025-01-30

## 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.

### Total sales for each product

#### Query:

```
SELECT
```

```
Product,
```

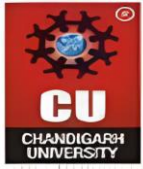
```
SUM(Price) AS TotalSales
```

```
FROM CustomerOrders
```

```
GROUP BY Product;
```

### Output

	product character varying (50)	totalsales numeric
1	Mobile	120000.00
2	Tablet	90000.00
3	Laptop	333000.00



### Average price of each product

#### Query:

```
SELECT
```

```
    Product,
```

```
    AVG(Price) AS AveragePrice
```

```
FROM CustomerOrders
```

```
GROUP BY Product;
```

#### Output:

	product character varying (50)	averageprice numeric
1	Mobile	40000.000000000000
2	Tablet	30000.000000000000
3	Laptop	83250.000000000000

### 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.

#### Column-Level / Group-Level Filtering (Using HAVING)

#### Query:

```
SELECT
```

```
    Product,
```

```
    SUM(Price) AS TotalSales
```

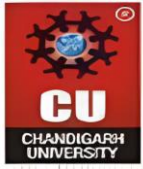
```
FROM CustomerOrders
```

```
GROUP BY Product
```

```
HAVING SUM(Price) > 100000;
```

#### Output:

	product character varying (50)	totalsales numeric
1	Mobile	120000.00
2	Laptop	333000.00



### Row-Level Filtering (Using WHERE)

#### Query:

```
SELECT
```

```
    Product,
```

```
    SUM(Price) AS TotalSales
```

```
FROM CustomerOrders
```

```
WHERE Price > 50000
```

```
GROUP BY Product;
```

#### Output:

	product character varying (50)	totalsales numeric
1	Mobile	60000.00
2	Laptop	333000.00

### 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:

#### Query:

```
SELECT Product, SUM(Price)
```

```
FROM CustomerOrders
```

```
WHERE SUM(Price) > 100000
```

```
GROUP BY Product;
```

#### Correct Usage:

#### Query:

```
SELECT Product, SUM(Price) AS TotalSales
```

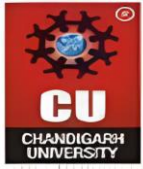
```
FROM CustomerOrders
```

```
GROUP BY Product
```

```
HAVING SUM(Price) > 100000;
```

#### Output:

	product character varying (50)	totalsales numeric
1	Mobile	120000.00
2	Laptop	333000.00



### **Learning Outcomes:**

- 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.