Experiment 2

Student Name: Nikhil Kumar
UID: 25MCI10036
Branch: MCA (AI/ML)
Section/Group: 25MAM-1(A)
Semester: 2-SEM
Date of Performance: 13/01/2026
Subject Name: Technical Training - 1
Subject Code: 25CAP-652

1. Aim of the Session:

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

2. Software Requirements:

- PostgreSQL
- pgAdmin
- Windows Operating System

3. Objectives of the Session:

- 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. Procedure of the practical

- Create a sample table representing Employee details
- Insert realistic records into the table

- Retrieve filtered data using WHERE clause
- Sort query results using ORDER BY clause
- Group records using GROUP BY clause
- Apply conditions on grouped data using HAVING clause
- Analyze execution order of WHERE and HAVING clauses

5. 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.

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.

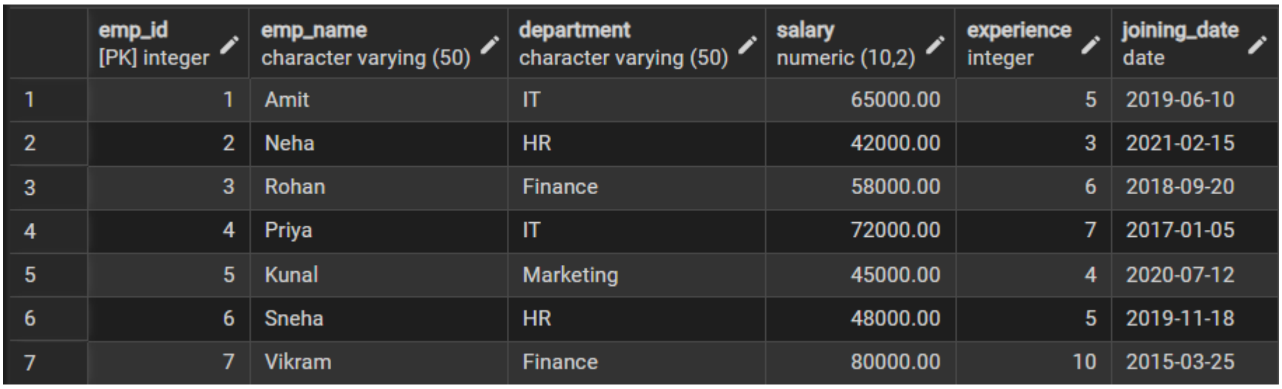
6. Input / Output details and Screenshots:

i. Creating sample table and inserting records.

Input:

```
CREATE TABLE employee_details (
    emp_id SERIAL PRIMARY KEY,
    emp_name VARCHAR(50),
    department VARCHAR(50),
    salary NUMERIC(10,2),
    experience INT,
    joining_date DATE
);
INSERT INTO employee_details (emp_name, department, salary, experience, joining_date) VALUES
('Amit', 'IT', 65000, 5, '2019-06-10'),
('Neha', 'HR', 42000, 3, '2021-02-15'),
('Rohan', 'Finance', 58000, 6, '2018-09-20'),
('Priya', 'IT', 72000, 7, '2017-01-05'),
('Kunal', 'Marketing', 45000, 4, '2020-07-12'),
('Sneha', 'HR', 48000, 5, '2019-11-18'),
('Vikram', 'Finance', 80000, 10, '2015-03-25');
SELECT * FROM employee_details;
```

Output:

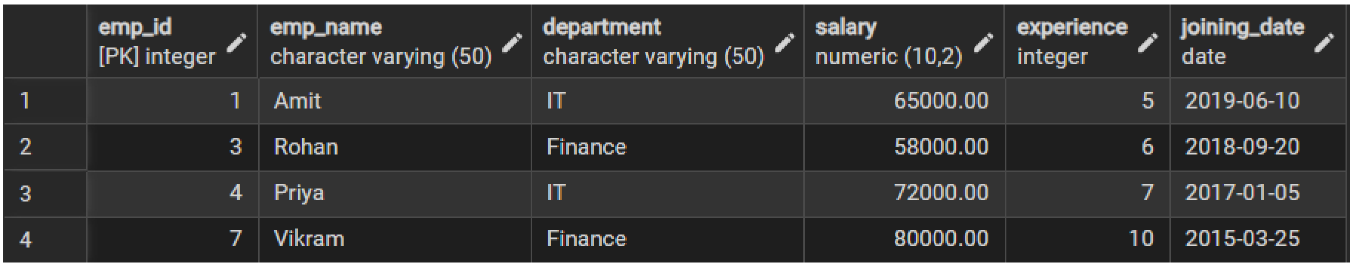


	emp_id [PK] integer	emp_name character varying (50)	department character varying (50)	salary numeric (10,2)	experience integer	joining_date date
1	1	Amit	IT	65000.00	5	2019-06-10
2	2	Neha	HR	42000.00	3	2021-02-15
3	3	Rohan	Finance	58000.00	6	2018-09-20
4	4	Priya	IT	72000.00	7	2017-01-05
5	5	Kunal	Marketing	45000.00	4	2020-07-12
6	6	Sneha	HR	48000.00	5	2019-11-18
7	7	Vikram	Finance	80000.00	10	2015-03-25

Displaying all records from employee_details table.

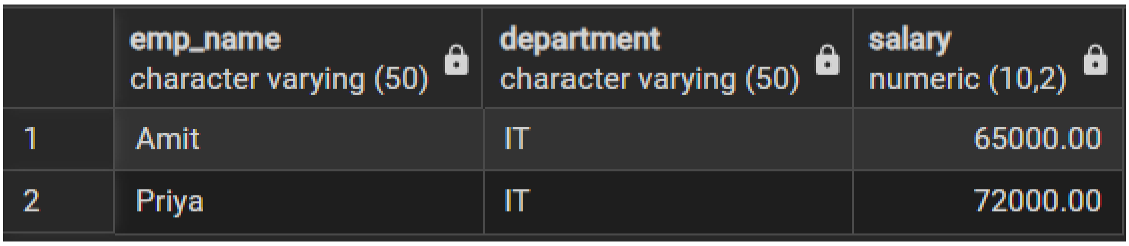
ii. Filtering data using WHERE clause

```
-- Employees with salary greater than 50000
SELECT *
FROM employee_details
WHERE salary > 50000;
```



	emp_id [PK] integer	emp_name character varying (50)	department character varying (50)	salary numeric (10,2)	experience integer	joining_date date
1	1	Amit	IT	65000.00	5	2019-06-10
2	3	Rohan	Finance	58000.00	6	2018-09-20
3	4	Priya	IT	72000.00	7	2017-01-05
4	7	Vikram	Finance	80000.00	10	2015-03-25

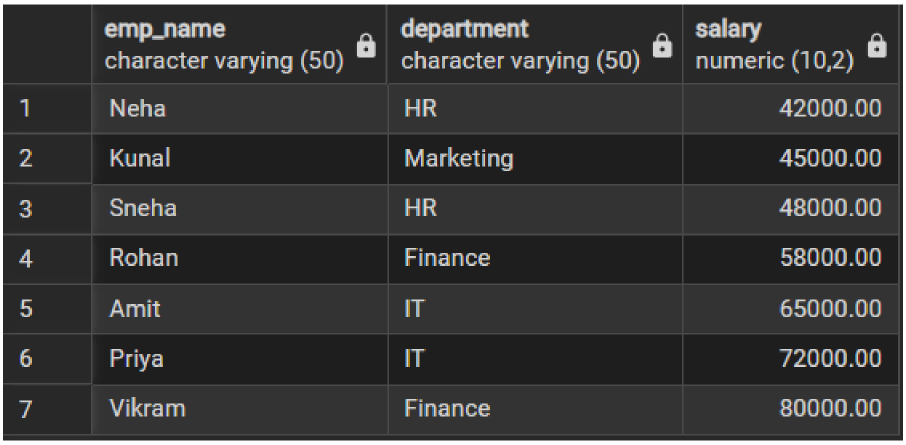
```
-- Employees from IT department
SELECT emp_name, department, salary
FROM employee_details
WHERE department = 'IT';
```



	emp_name character varying (50)	department character varying (50)	salary numeric (10,2)
1	Amit	IT	65000.00
2	Priya	IT	72000.00

iii. Sorting query results using ORDER BY clause

```
-- Sort employees by salary (ascending)
SELECT emp_name, department, salary
FROM employee_details
ORDER BY salary ASC;
```



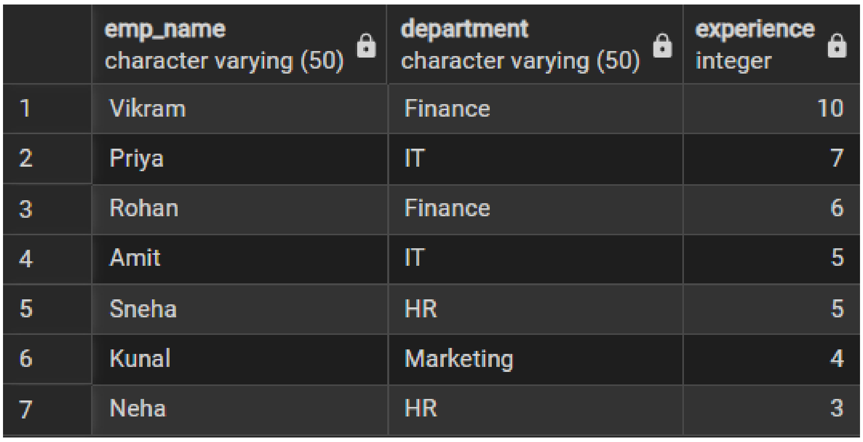
	emp_name character varying (50)	department character varying (50)	salary numeric (10,2)
1	Neha	HR	42000.00
2	Kunal	Marketing	45000.00
3	Sneha	HR	48000.00
4	Rohan	Finance	58000.00
5	Amit	IT	65000.00
6	Priya	IT	72000.00
7	Vikram	Finance	80000.00

-- Sort employees by experience (descending)

SELECT emp_name, department, experience

FROM employee_details

ORDER BY experience DESC;



	emp_name character varying (50)	department character varying (50)	experience integer
1	Vikram	Finance	10
2	Priya	IT	7
3	Rohan	Finance	6
4	Amit	IT	5
5	Sneha	HR	5
6	Kunal	Marketing	4
7	Neha	HR	3

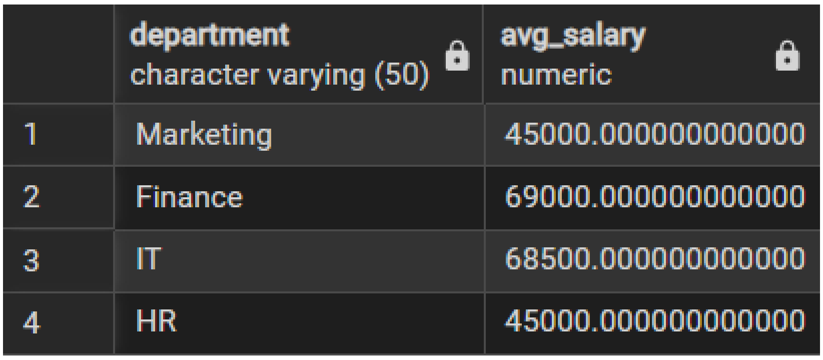
iv. Grouping data for aggregation.

-- Average salary per department

SELECT department, AVG(salary) AS avg_salary

FROM employee_details

GROUP BY department;



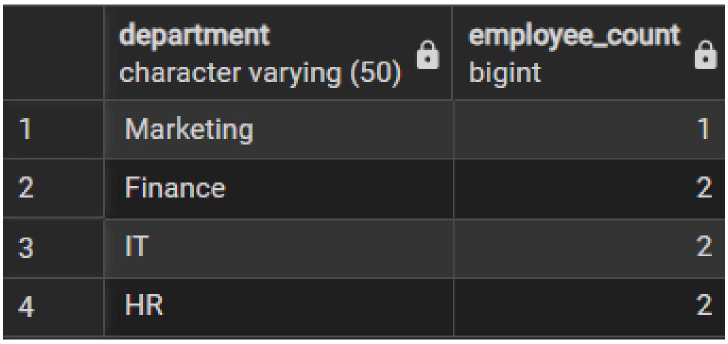
	department character varying (50)	avg_salary numeric
1	Marketing	45000.000000000000
2	Finance	69000.000000000000
3	IT	68500.000000000000
4	HR	45000.000000000000

-- Number of employees in each department

SELECT department, COUNT(emp_id) AS employee_count

FROM employee_details

GROUP BY department;



	department character varying (50)	employee_count bigint
1	Marketing	1
2	Finance	2
3	IT	2
4	HR	2

v. Applying conditions on aggregated data using HAVING clause.

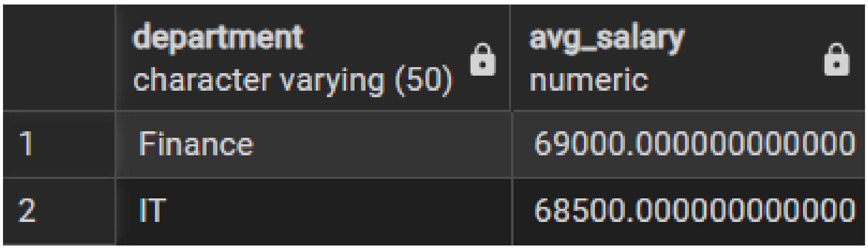
-- Departments with average salary greater than 50000

SELECT department, AVG(salary) AS avg_salary

FROM employee_details

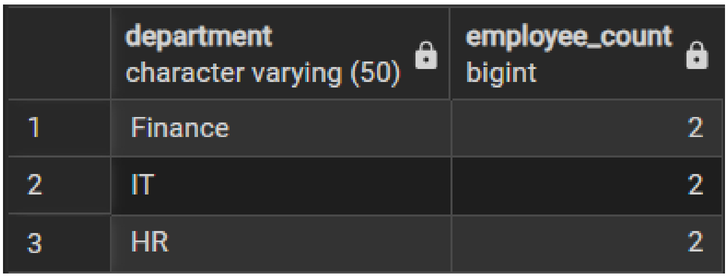
GROUP BY department

HAVING AVG(salary) > 50000;



	department character varying (50)	avg_salary numeric
1	Finance	69000.000000000000
2	IT	68500.000000000000

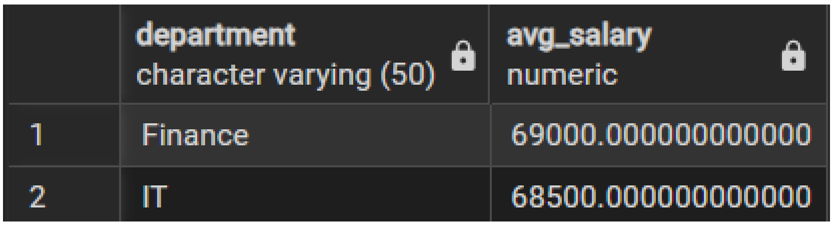
```
-- Departments having more than or equal to 2 employees

SELECT department, COUNT(emp_id) AS employee_count

FROM employee_details

GROUP BY department

HAVING COUNT(emp_id) >= 2;
```



	department character varying (50)	employee_count bigint
1	Finance	2
2	IT	2
3	HR	2

vi. Using WHERE and HAVING together

```
-- Display departments where:

-- 1. Only employees with more than 4 years of experience are considered (row-level filtering)

-- 2. The average salary of those employees is greater than 60000 (group-level filtering)

SELECT department, AVG(salary) AS avg_salary

FROM employee_details

WHERE experience > 4          -- Filters rows before grouping

GROUP BY department            -- Groups records department-wise

HAVING AVG(salary) > 60000;     -- Filters groups after aggregation
```



	department character varying (50)	avg_salary numeric
1	Finance	69000.000000000000
2	IT	68500.000000000000

7. Learning Outcomes (What I have learned):

- 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.