

Name: Nandesh T

UID: 24BAI70121

Course: Bachelor of CSE (AI) – 2nd Year

Subject: Database Management Systems

Experiment 2: Advanced Data Aggregation and Filtering

1. Aim of the Session

The aim of this practical is to implement and analyze **Group Functions** and **Conditional Filtering** in SQL. The session focuses on using `GROUP BY`, `HAVING`, and `ORDER BY` clauses to extract meaningful insights from an employee dataset.

2. Objective of the Session

By completing this practical, I have achieved the following:

- Developed a schema for employee management using appropriate data types like `NUMERIC` and `DATE`.
- Mastered the use of **Aggregate Functions** (specifically `AVG`) to perform calculations on data groups.
- Learned to differentiate between the `WHERE` clause (row-level filtering) and the `HAVING` clause (group-level filtering).
- Gained proficiency in sorting aggregated results using the `ORDER BY` clause.

3. Practical / Experiment Steps

The following implementation tasks were completed:

1. **Schema Definition:** Created the `employee` table with constraints and precise numeric scaling for salaries.
2. **Data Population:** Inserted diverse records representing various departments (IT, HR, Sales, Finance) and salary ranges.
3. **Basic Aggregation:** Calculated the average salary per department using the `GROUP BY` clause.
4. **Advanced Filtering:** Applied the `HAVING` clause to filter out departments where the average salary did not meet a specific threshold.

5. **Complex Querying:** Combined WHERE, GROUP BY, HAVING, and ORDER BY into a single query to refine results based on individual salaries and group averages simultaneously.

4. Procedure of the Practical

The experiment was conducted following these sequential steps:

1. **System Initialization:** Logged into the PostgreSQL environment via pgAdmin 4 using localhost as the host server.
2. **Table Construction:** Executed the CREATE TABLE command to define the structure for the employee dataset.
3. **Data Insertion:** Ran multiple INSERT statements to populate the table with the provided employee data.
4. **Initial Verification:** Used SELECT * to confirm that all employee records were correctly stored and formatted.
5. **Group Analysis:** Executed a GROUP BY query to observe the distribution of average salaries across different departments.
6. **Applying Group Filters:** Integrated the HAVING clause to restrict the output to high-paying departments (Average > 30,000).
7. **Final Refinement:** Executed a comprehensive query that filtered individual employees (Salary > 20,000), grouped them by department, and sorted the results in descending order.
8. **Output Recording:** Captured screenshots of the query results and saved the final SQL script for documentation.

5. I/O Analysis (Input / Output Analysis)

Input Queries

SQL

```
CREATE TABLE Employee (
    Id VARCHAR(5),
    Name VARCHAR(50),
    Department VARCHAR(30),
    Salary INT,
    DOJ DATE
);

SELECT department, AVG(salary) AS avg_salary
FROM employee
WHERE salary > 20000
GROUP BY department
HAVING AVG(salary) > 30000
ORDER BY avg_salary DESC;
```

Output Details

- **Aggregate Results:** The system successfully grouped employees by department.

The screenshot shows a SQL query editor interface. At the top, there are tabs for "Query" and "Query History". Below the tabs, three SQL statements are listed:

```
16  SELECT department, AVG(salary), AS avg_salary,  
17  FROM employee  
18  WHERE salary > 20000  
19  GROUP BY department  
20  HAVING AVG(salary) > 30000  
21  ORDER BY avg_salary DESC;  
22  
23  SELECT department, AVG(salary) AS avg_salary  
24  FROM employee GROUP BY department;  
25  
26  SELECT department, AVG(salary) AS avg_salary  
27  FROM employee GROUP BY department  
    HAVING AVG (salary) > 30000 ;
```

Below the queries, there are tabs for "Data Output", "Messages", and "Notifications". Under "Data Output", there is a toolbar with icons for file operations and a "SQL" button. A message bar indicates "Showing rows: 1 to 4" and "Page No: 1 of 1".

	department character varying (30)	avg_salary numeric
1	Finance	50000.0000000000000000
2	Sales	40000.0000000000000000
3	IT	80000.0000000000000000
4	HR	100000.0000000000000000

- **Filtering Logic:** The WHERE clause correctly excluded employees with salaries under 20,000 before calculating averages.

- **Group Filtering:** The HAVING clause ensured only departments with an average salary exceeding 30,000 were displayed in the final output.

Query Query History

```

16   SELECT department, AVG(salary) AS avg_salary
17   FROM employee
18   WHERE salary > 20000
19   GROUP BY department
20   HAVING AVG(salary) > 30000
21   ORDER BY avg_salary DESC;
22
23   SELECT department, AVG(salary) AS avg_salary
24   FROM employee GROUP BY department;
25
26   SELECT department, AVG(salary) AS avg_salary
27   FROM employee GROUP BY department
28   HAVING AVG (salary) > 30000 ;

```

Data Output Messages Notifications

Showing rows: 1 to 4 Page No: 1 of 1

	department character varying (30)	avg_salary numeric
1	Finance	50000.00000000000000
2	Sales	40000.00000000000000
3	IT	80000.00000000000000
4	HR	100000.00000000000000

- **Sorting:** The ORDER BY clause successfully sorted the final results from highest to lowest average salary.

The screenshot shows a SQL query editor interface. The top section contains the following SQL code:

```

10 INSERT INTO Employee VALUES ('e002', 'Abhi', 'Sales', 40000, '02-AUG-2022');
11 INSERT INTO Employee VALUES ('e003', 'Nethra', 'HR', 100000, '03-JUN-2019');
12 INSERT INTO Employee VALUES ('e004', 'Gowtham', 'IT', 80000, '18-NOV-2020');
13 INSERT INTO Employee VALUES ('e005', 'Prabas', 'Finance', 50000, '19-SEP-2022');
14
15 SELECT department, AVG(salary) AS avg_salary
16 FROM employee
17 WHERE salary > 20000
18 GROUP BY department
19 HAVING AVG(salary) > 30000
20 ORDER BY avg_salary DESC;
21

```

The bottom section displays the results of the query:

	department character varying (30)	avg_salary numeric
1	HR	100000.000000000000
2	IT	80000.000000000000
3	Finance	50000.000000000000
4	Sales	40000.000000000000

6. Learning Outcome

Through this session, I have developed the following competencies:

- **Analytical Skills:** Gained the ability to transform raw row-level data into high-level summary reports using aggregation.
- **Query Logic:** Understood the logical execution order of SQL clauses: FROM → WHERE → GROUP BY → HAVING → SELECT → ORDER BY.
- **Practical Exposure:** Experienced handling real-world data scenarios, such as department-wise salary analysis and performance-based filtering in a professional database environment.