Name: Yashraj Jangra

UID: 24BAI70476

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

```
-- Use This Command to Drop the Table if it Already Exists
-- DROP TABLE USERS CASCADE CONSTRAINTS;

CREATE TABLE users (
    user_id INT PRIMARY KEY,
```

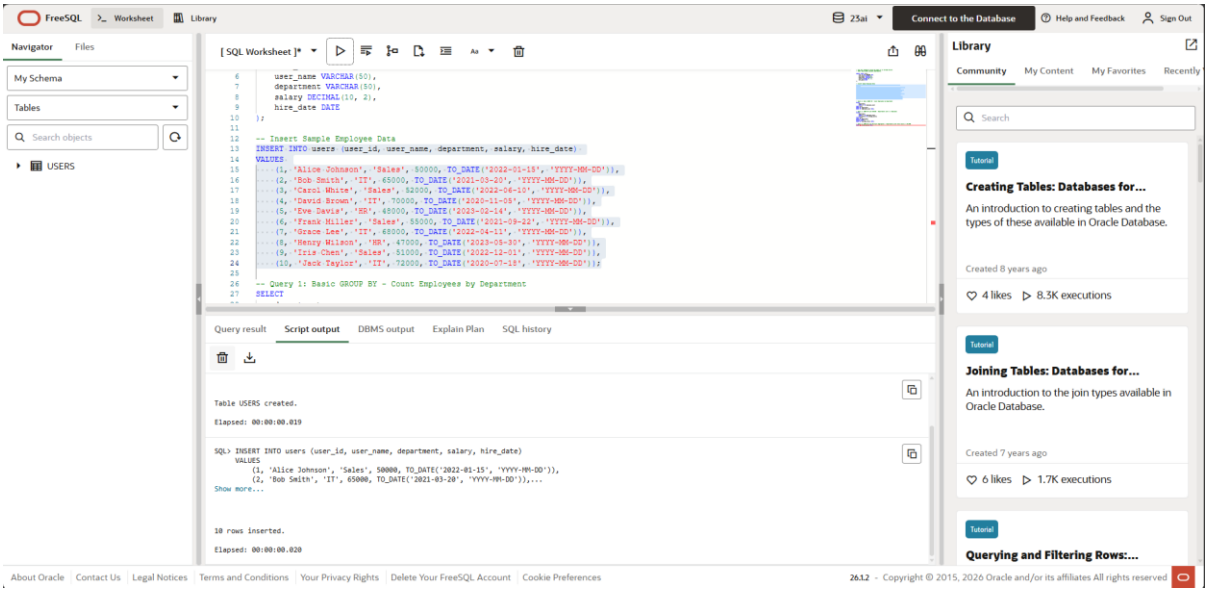
```
    user_name VARCHAR(50),
    department VARCHAR(50),
    salary DECIMAL(10, 2),
    hire_date DATE
);

-- Insert Sample Employee Data
INSERT INTO users (user_id, user_name, department, salary, hire_date)
VALUES
    (1, 'Alice Johnson', 'Sales', 50000, TO_DATE('2022-01-15', 'YYYY-MM-DD')),
    (2, 'Bob Smith', 'IT', 65000, TO_DATE('2021-03-20', 'YYYY-MM-DD')),
    (3, 'Carol White', 'Sales', 52000, TO_DATE('2022-06-10', 'YYYY-MM-DD')),
    (4, 'David Brown', 'IT', 70000, TO_DATE('2020-11-05', 'YYYY-MM-DD')),
    (5, 'Eve Davis', 'HR', 48000, TO_DATE('2023-02-14', 'YYYY-MM-DD')),
    (6, 'Frank Miller', 'Sales', 55000, TO_DATE('2021-09-22', 'YYYY-MM-DD')),
    (7, 'Grace Lee', 'IT', 68000, TO_DATE('2022-04-11', 'YYYY-MM-DD')),
    (8, 'Henry Wilson', 'HR', 47000, TO_DATE('2023-05-30', 'YYYY-MM-DD')),
    (9, 'Iris Chen', 'Sales', 51000, TO_DATE('2022-12-01', 'YYYY-MM-DD')),
    (10, 'Jack Taylor', 'IT', 72000, TO_DATE('2020-07-18', 'YYYY-MM-DD'));

-- Query 1: Basic GROUP BY - Count Employees by Department
SELECT
    department,
    COUNT(*) AS employee_count
FROM users
GROUP BY department
```

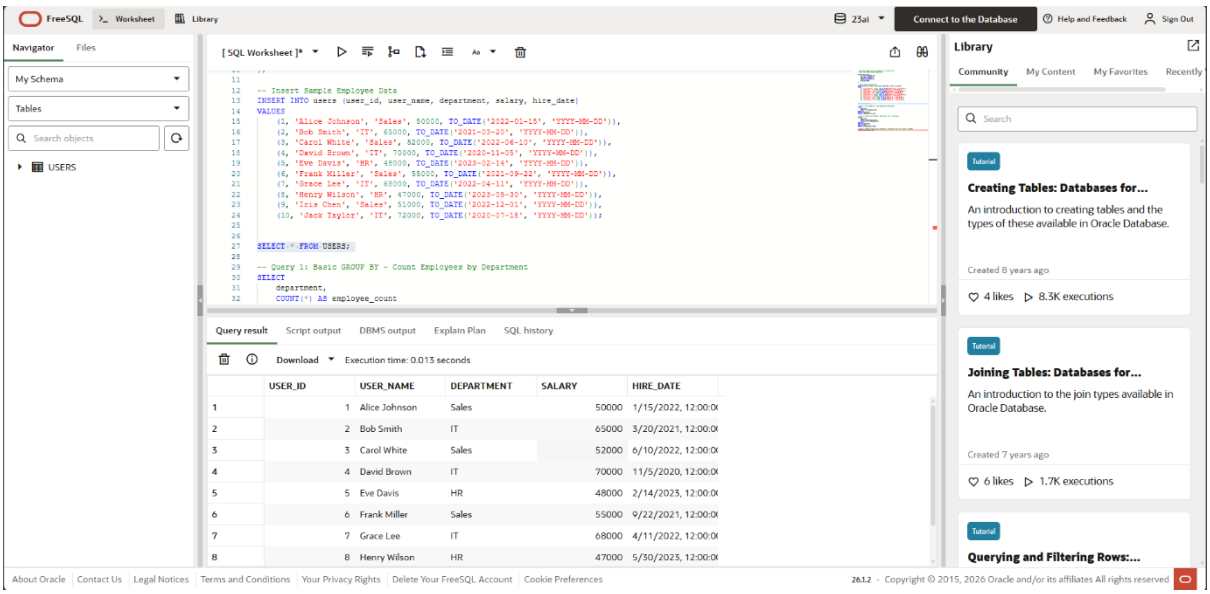
```
ORDER BY employee_count DESC;


-- Query 2: GROUP BY with HAVING - Departments with > 2 Employees
SELECT
    department,
    COUNT(*) AS employee_count,
    AVG(salary) AS average_salary
FROM users
GROUP BY department
HAVING COUNT(*) > 2
ORDER BY average_salary DESC;


-- Query 3: GROUP BY with Multiple Aggregates - Departments with Total Salary > 150,000
SELECT
    department,
    COUNT(*) AS employee_count,
    SUM(salary) AS total_salary,
    AVG(salary) AS average_salary
FROM users
GROUP BY department
HAVING SUM(salary) > 150000
ORDER BY total_salary DESC;
```

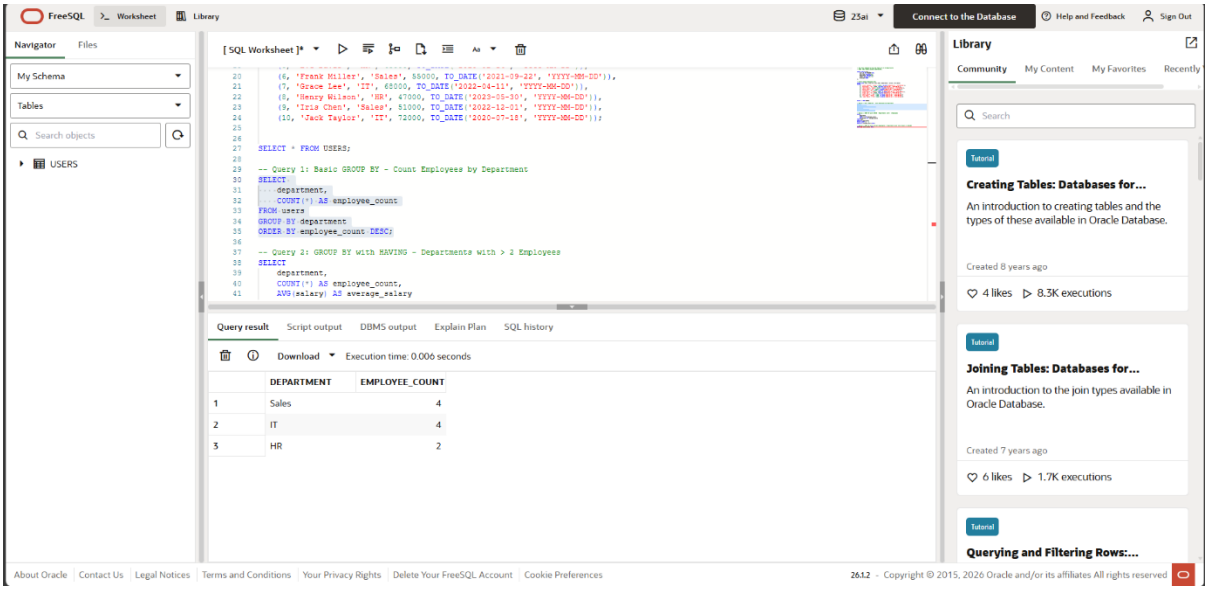
Output Details

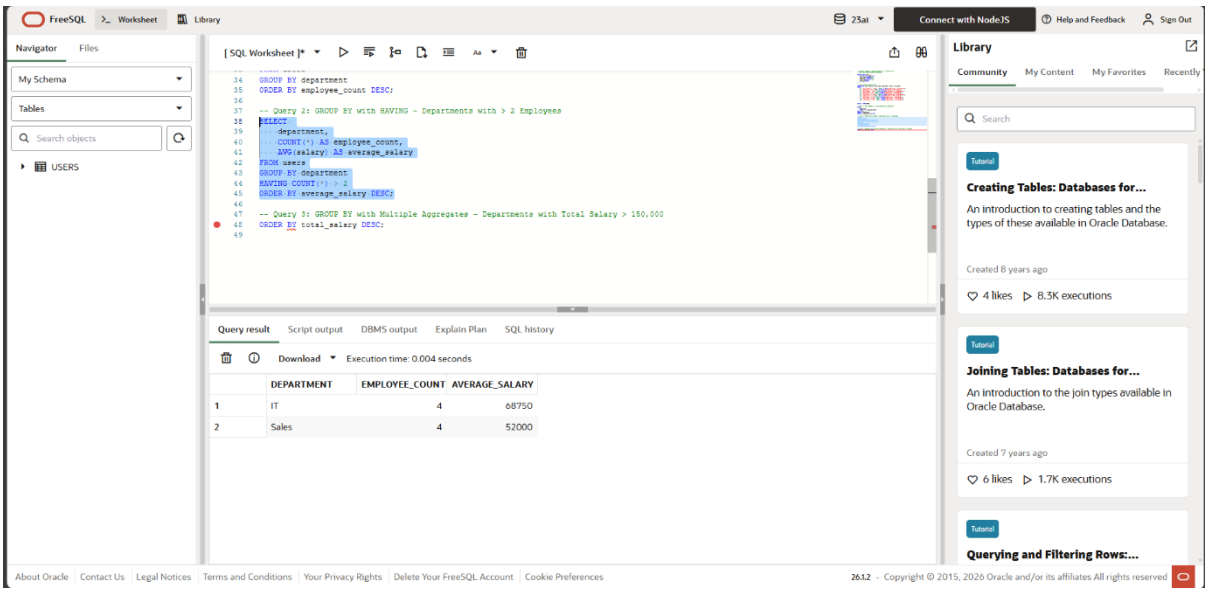
- Aggregate Results: The system successfully grouped employees by department.

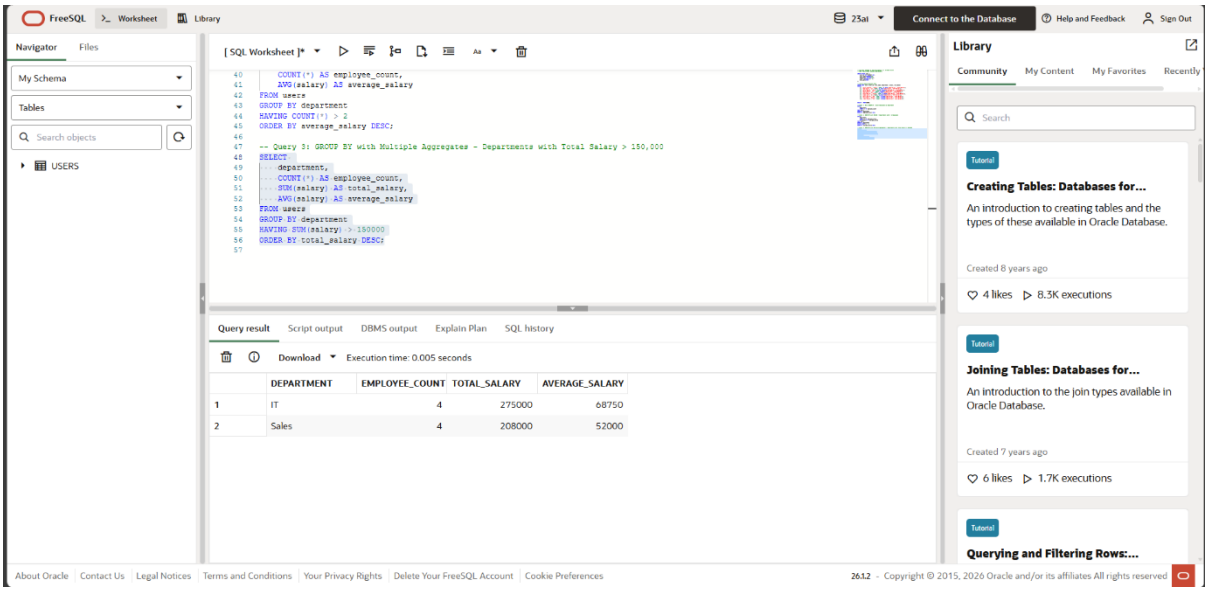
- Filtering Logic: The WHERE clause correctly excluded employees with salaries under 20,000 (like Sara and Vikram) before calculating averages.
- Group Filtering: The HAVING clause ensured only departments with an average salary exceeding 30,000 were displayed in the final output.
- Sorting: The ORDER BY clause successfully sorted the final results from highest to lowest average salary.











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.