Word Content for Experiment 1.2 – SQL SELECT Queries

Experiment 1.2 – SQL SELECT Queries with WHERE, GROUP BY, HAVING, ORDER BY

Experiment:
Experiment 1.2: Practicing SQL SELECT queries with WHERE, GROUP BY, HAVING, and ORDER BY clauses to retrieve and analyze data from the EMPLOYEE table.

Aim:
The aim of this experiment is to practice writing SQL SELECT statements with filtering, grouping, sorting, and aggregate functions to analyze employee data.

Objective:

- Practice writing SQL SELECT statements.
- Apply filtering conditions using the WHERE clause.
- Sort query results using the ORDER BY clause.
- Group records using the GROUP BY clause.
- Filter grouped data using the HAVING clause.
- Use aggregate functions like COUNT(), SUM(), AVG(), MIN(), and MAX() to analyze data.

Software Requirements:

- Database: Oracle XE or PostgreSQL (PgAdmin)

Practical / Experiment Step:

1. Display the department name and the average salary of employees for each department.
2. Consider only those employees whose salary is greater than 20,000.
3. Display only those departments where the average salary is greater than 30,000.
4. Arrange the final output in descending order of average salary.

Procedure of the Experiment:

1. Start the system and log in to the computer.
2. Open the required database tool (Oracle XE or PgAdmin).
3. Connect to the database containing the EMPLOYEE table.

4. Examine the EMPLOYEE table structure and data.
5. Write SQL SELECT queries for practicing filtering, grouping, sorting, and aggregates.
6. Execute queries and verify outputs like MAX, AVG, SUM, COUNT.
7. Execute the final query requested in the experiment (big query).
8. Note down results and take screenshots for record.

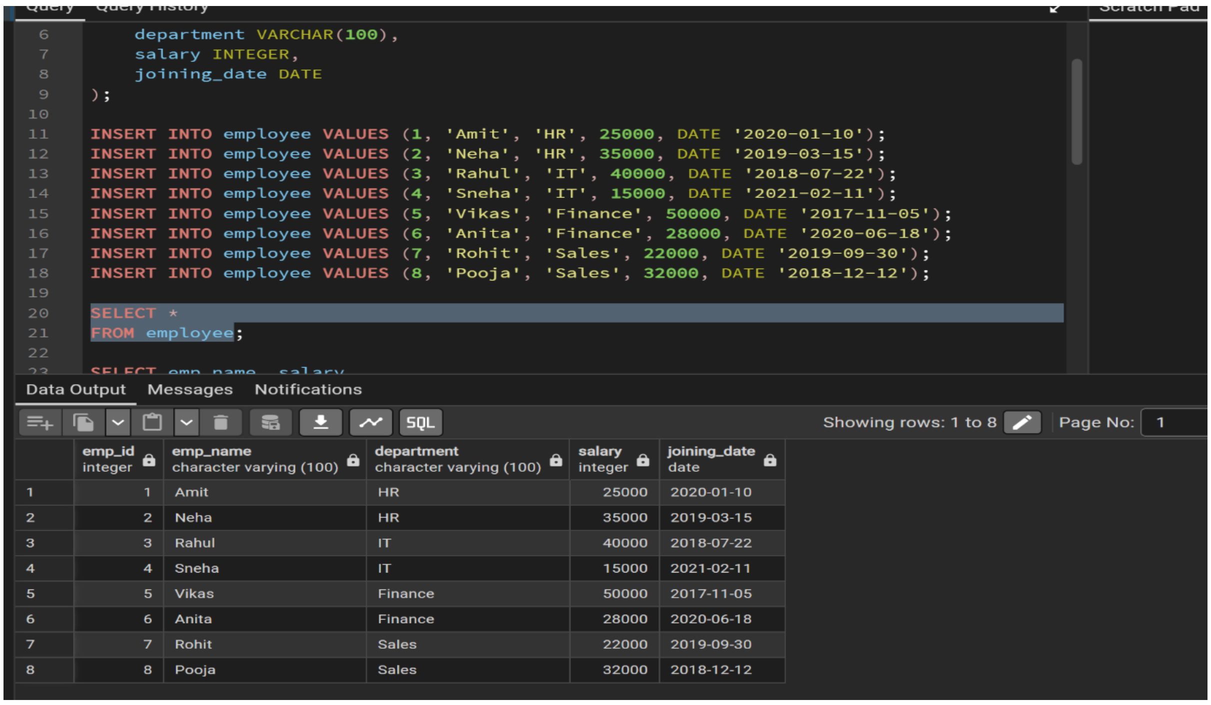
Input / Output Details:

Input:

- EMPLOYEE table with columns: emp_id, emp_name, Department, Salary, joining_date.
- SQL SELECT queries using WHERE, GROUP BY, HAVING, ORDER BY, and aggregate functions.

Output:

- Screenshots showing results of individual queries (MAX, AVG, SUM, etc.).
- Final screenshot showing the big query output as required.
- Output arranged according to the experiment steps.



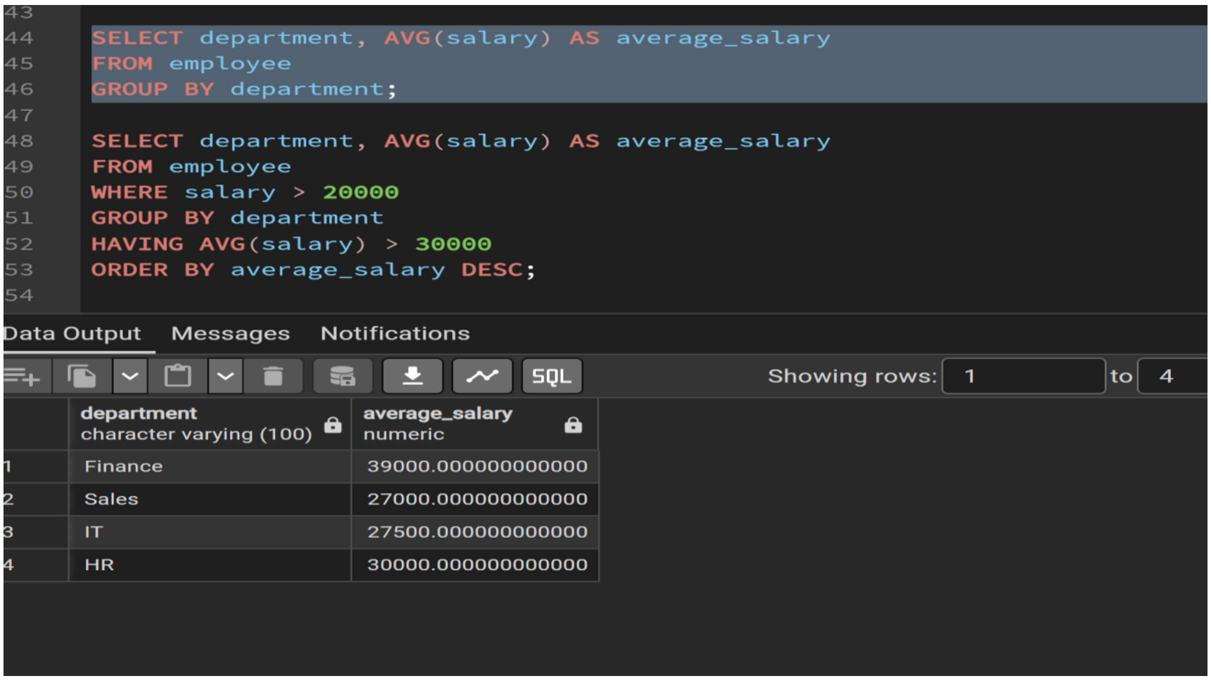
```
    department VARCHAR(100),
    salary INTEGER,
    joining_date DATE
);

INSERT INTO employee VALUES (1, 'Amit', 'HR', 25000, DATE '2020-01-10');
INSERT INTO employee VALUES (2, 'Neha', 'HR', 35000, DATE '2019-03-15');
INSERT INTO employee VALUES (3, 'Rahul', 'IT', 40000, DATE '2018-07-22');
INSERT INTO employee VALUES (4, 'Sneha', 'IT', 15000, DATE '2021-02-11');
INSERT INTO employee VALUES (5, 'Vikas', 'Finance', 50000, DATE '2017-11-05');
INSERT INTO employee VALUES (6, 'Anita', 'Finance', 28000, DATE '2020-06-18');
INSERT INTO employee VALUES (7, 'Rohit', 'Sales', 22000, DATE '2019-09-30');
INSERT INTO employee VALUES (8, 'Pooja', 'Sales', 32000, DATE '2018-12-12');

SELECT *
FROM employee;

SELECT emp_name, salary
```

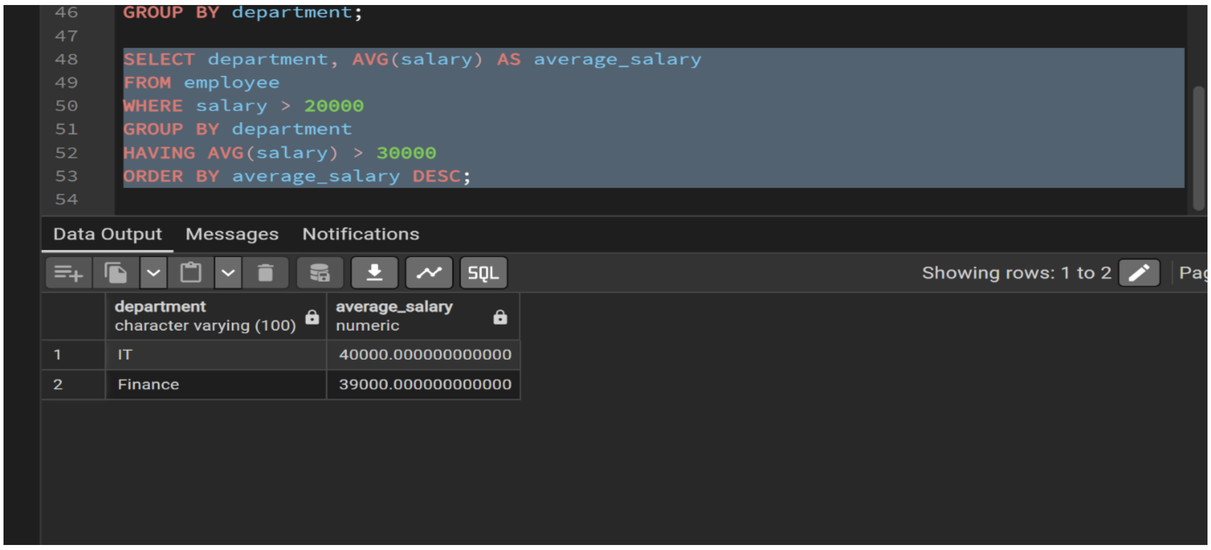
	emp_id integer	emp_name character varying (100)	department character varying (100)	salary integer	joining_date date
1	1	Amit	HR	25000	2020-01-10
2	2	Neha	HR	35000	2019-03-15
3	3	Rahul	IT	40000	2018-07-22
4	4	Sneha	IT	15000	2021-02-11
5	5	Vikas	Finance	50000	2017-11-05
6	6	Anita	Finance	28000	2020-06-18
7	7	Rohit	Sales	22000	2019-09-30
8	8	Pooja	Sales	32000	2018-12-12



```
SELECT department, AVG(salary) AS average_salary
FROM employee
GROUP BY department;

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

	department character varying (100)	average_salary numeric
1	Finance	39000.000000000000
2	Sales	27000.000000000000
3	IT	27500.000000000000
4	HR	30000.000000000000



```
GROUP BY department;

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

	department character varying (100)	average_salary numeric
1	IT	40000.000000000000
2	Finance	39000.000000000000

Learning Outcome:
After completing this experiment, the student will be able to:

- Filter records using the WHERE clause.
- Group records using GROUP BY.
- Apply conditions on grouped data using HAVING.
- Sort query results using ORDER BY.
- Analyze data using aggregate functions for insights.
- Understand how to write complex queries combining multiple SQL clauses.