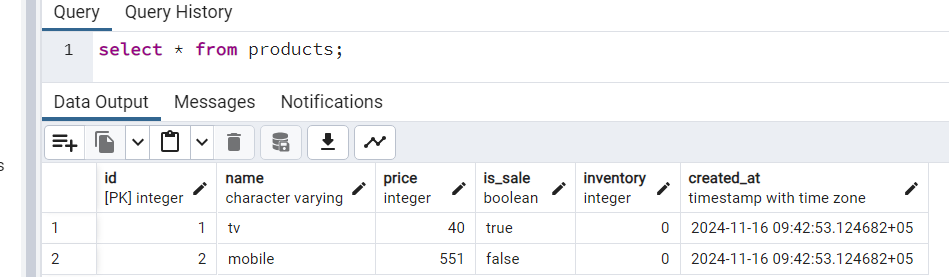
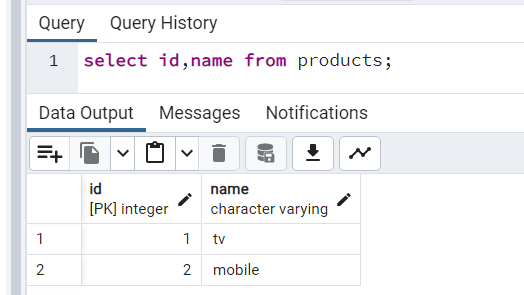
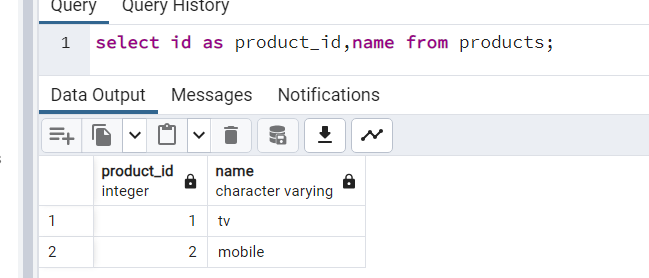
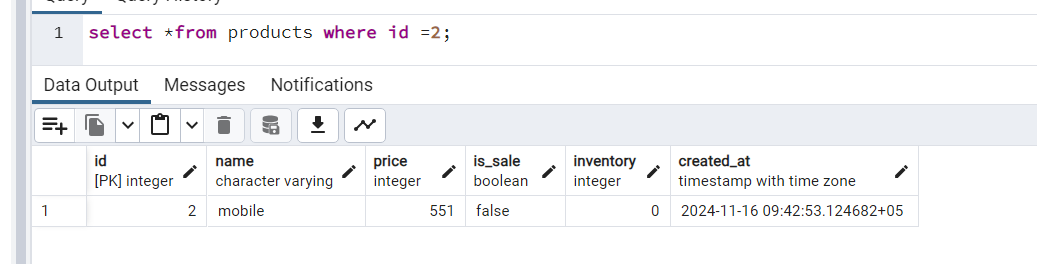
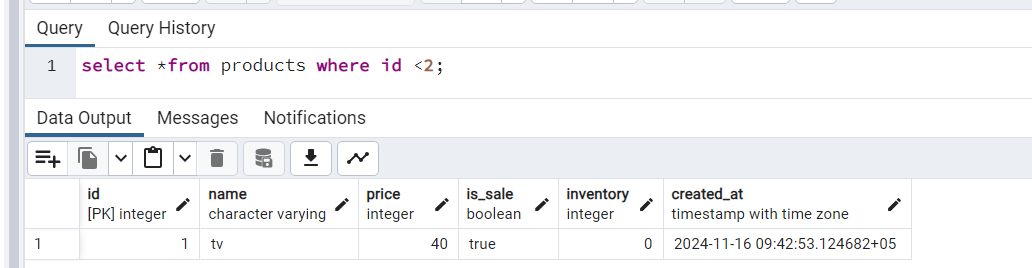
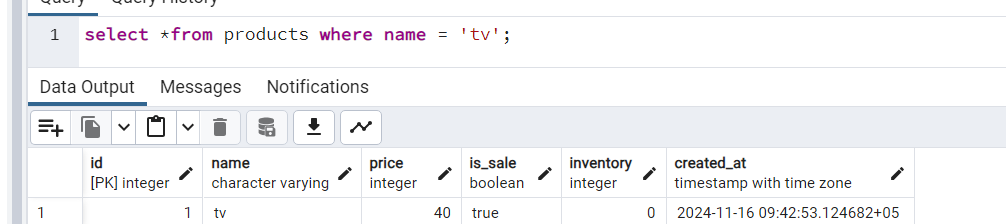
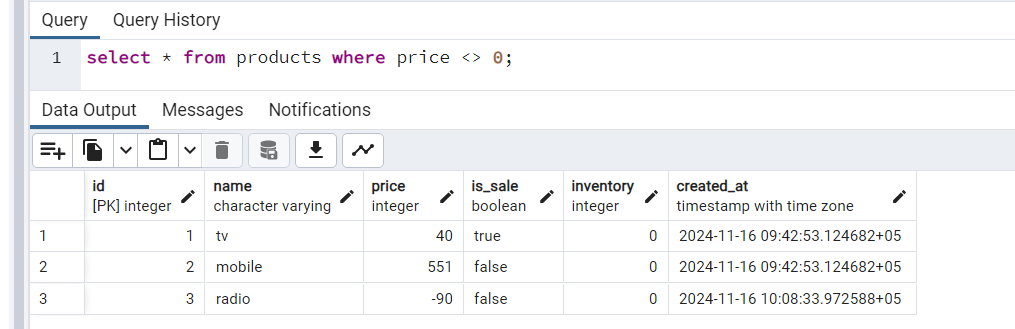
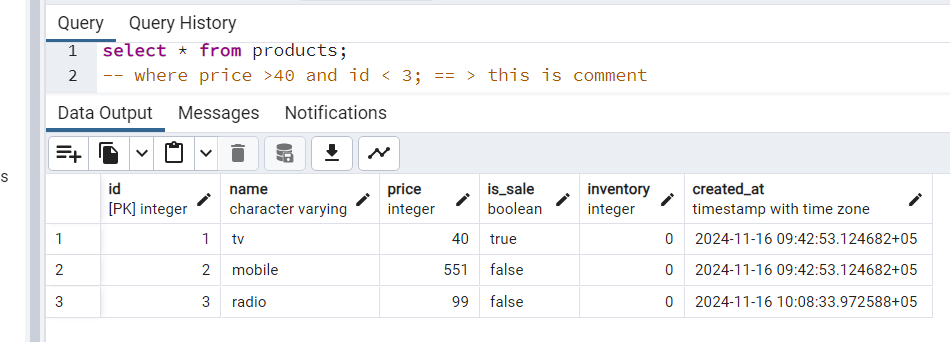
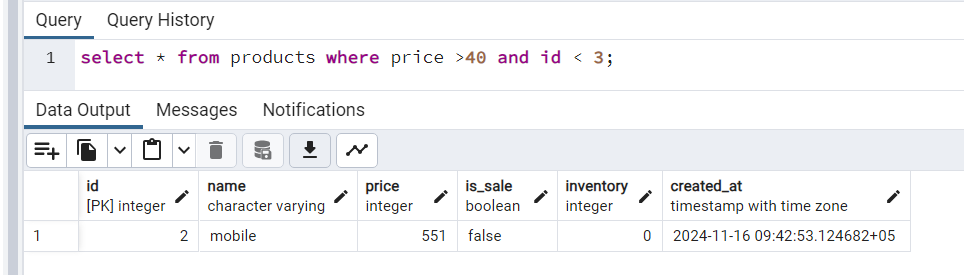
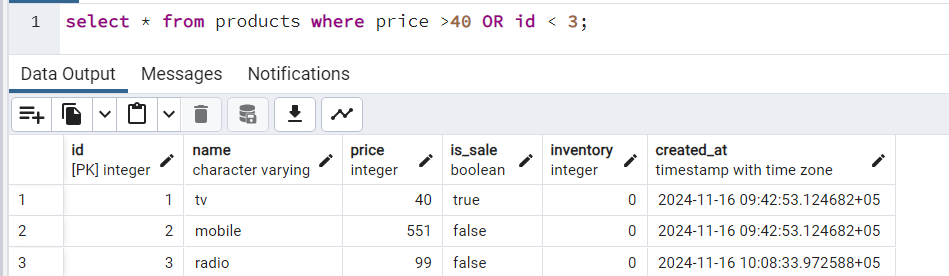
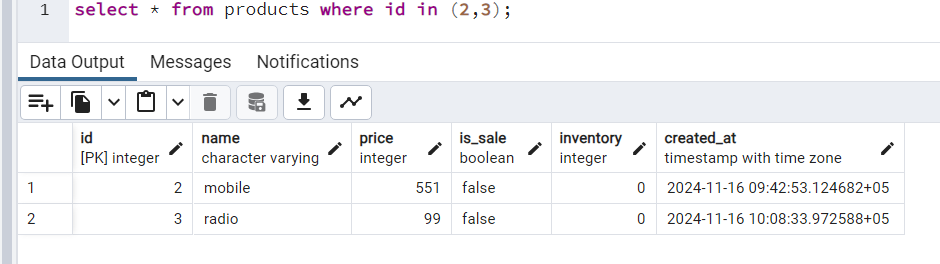
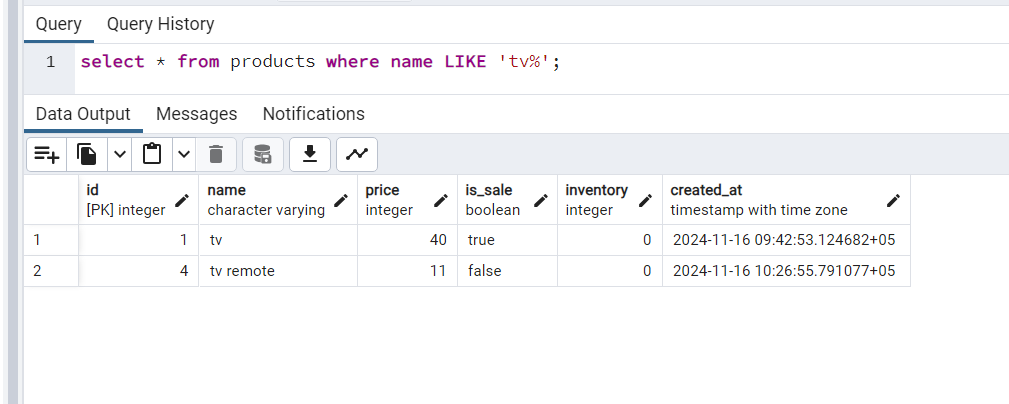
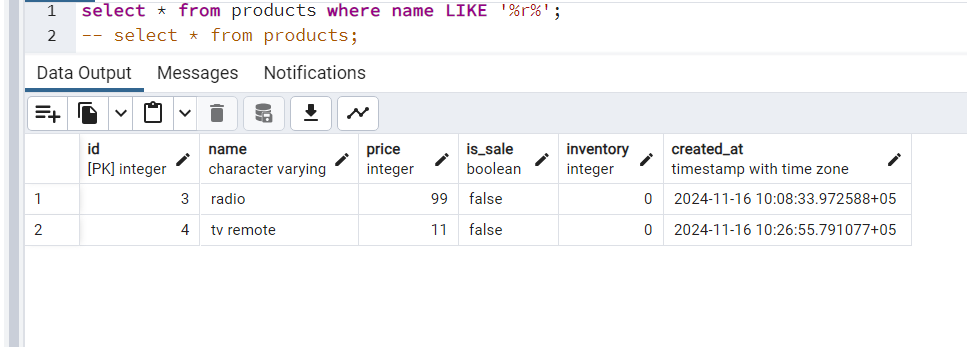
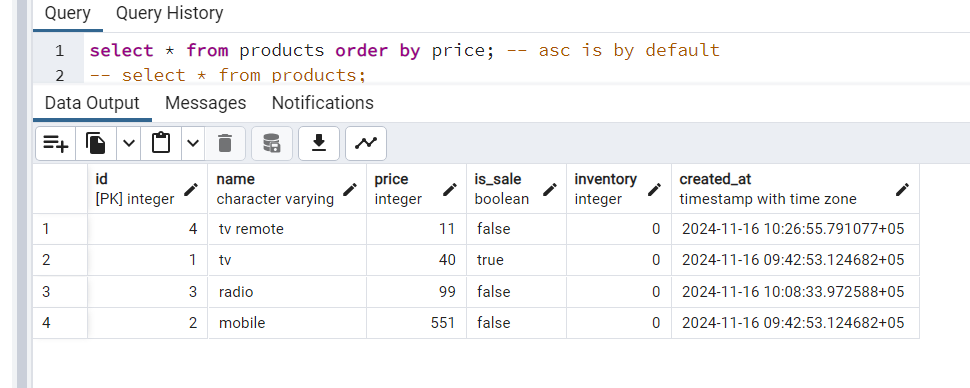
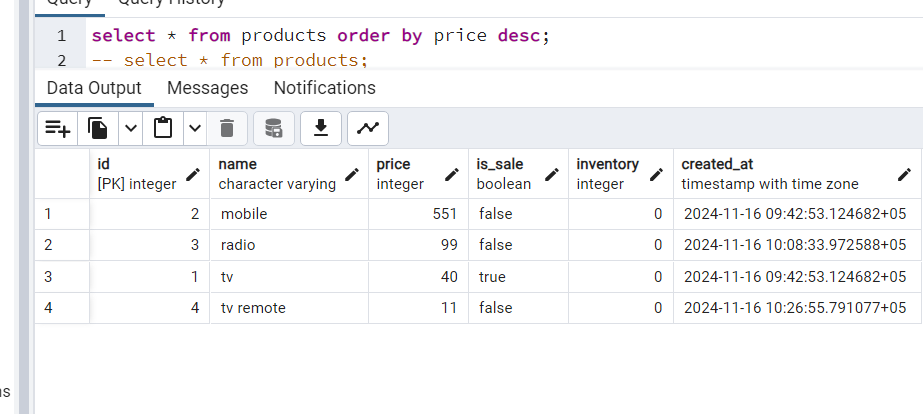
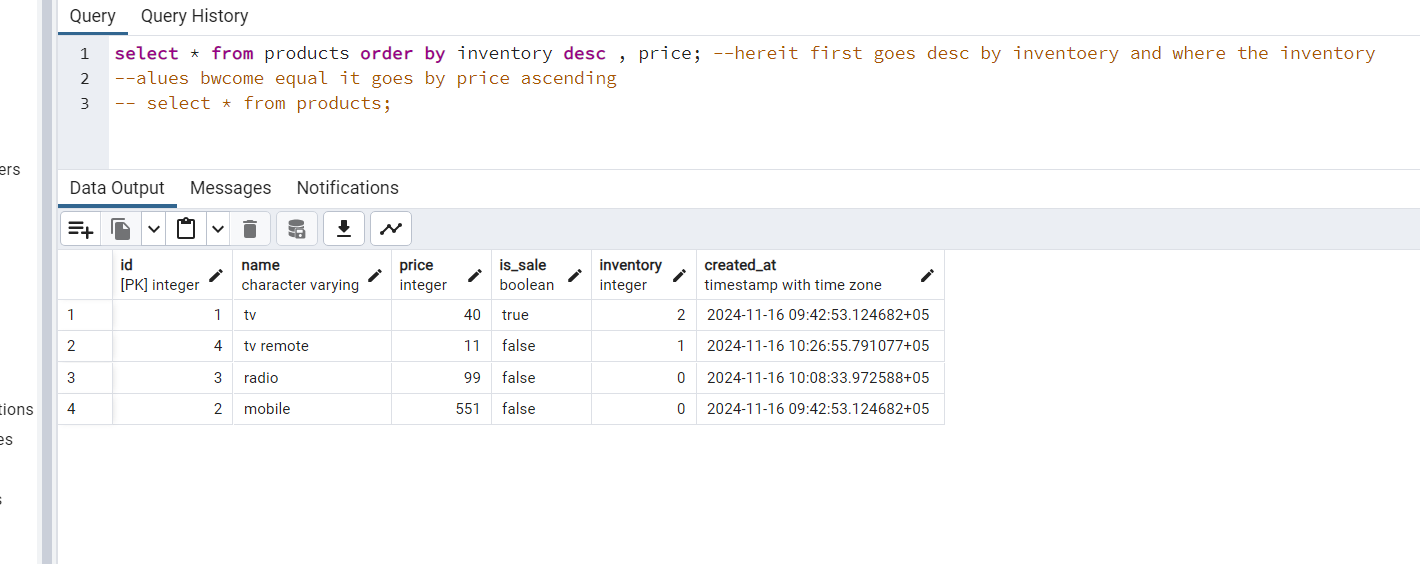
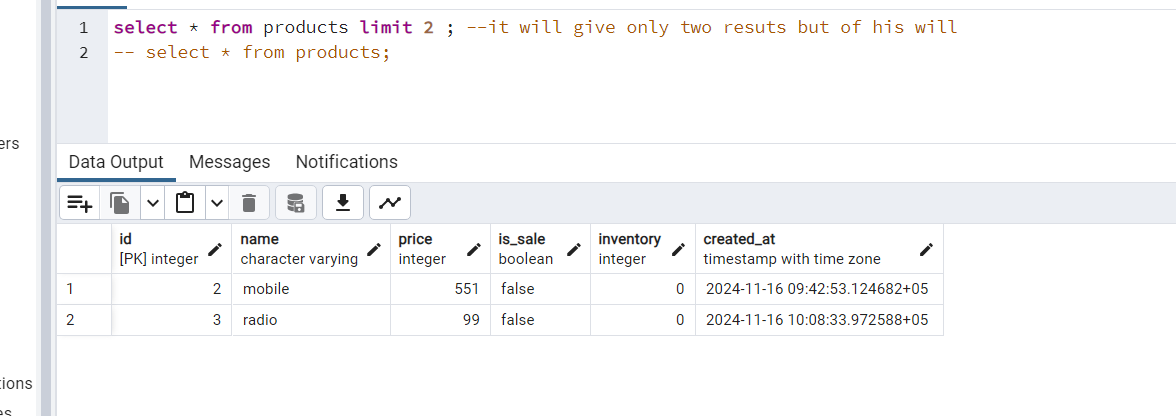
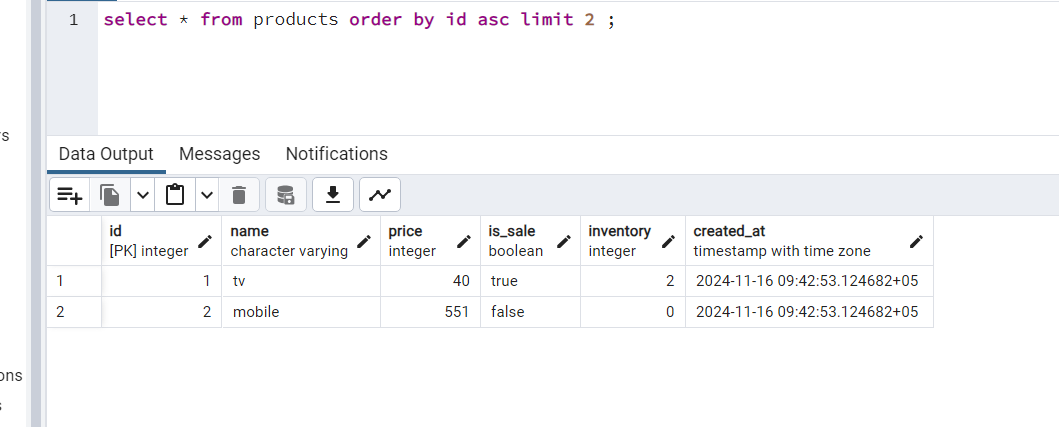
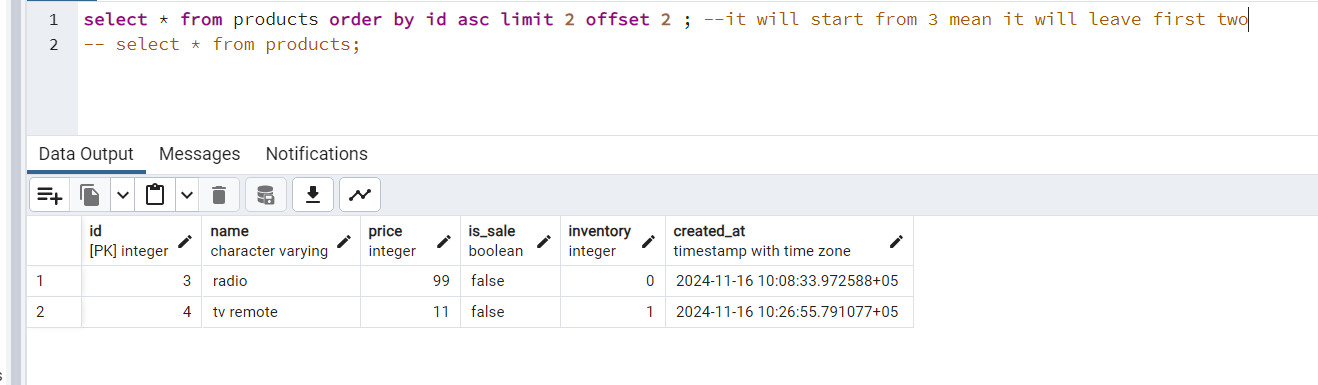
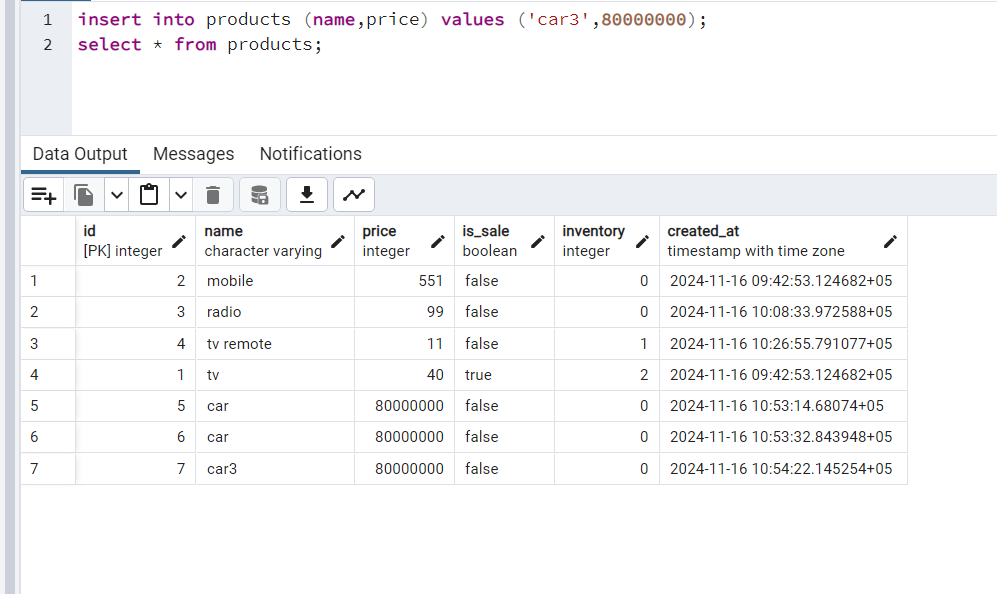
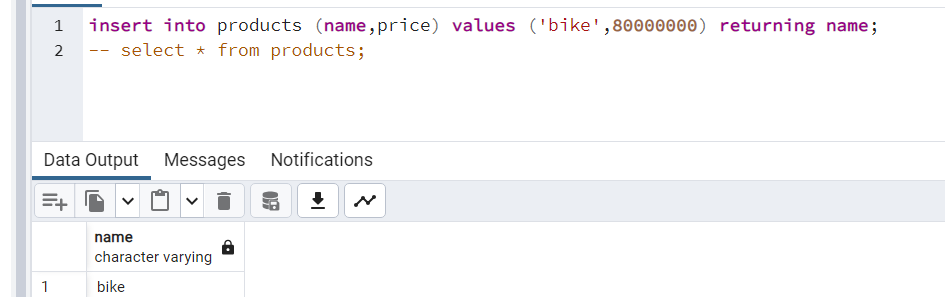
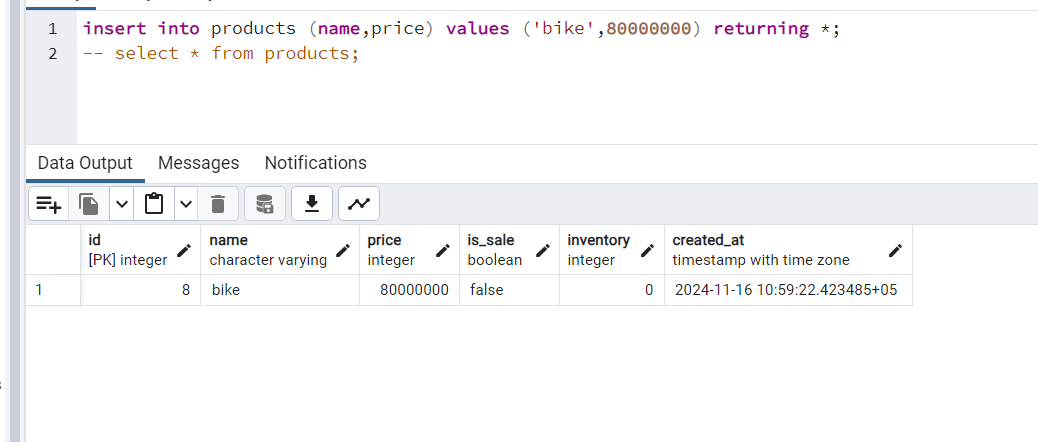
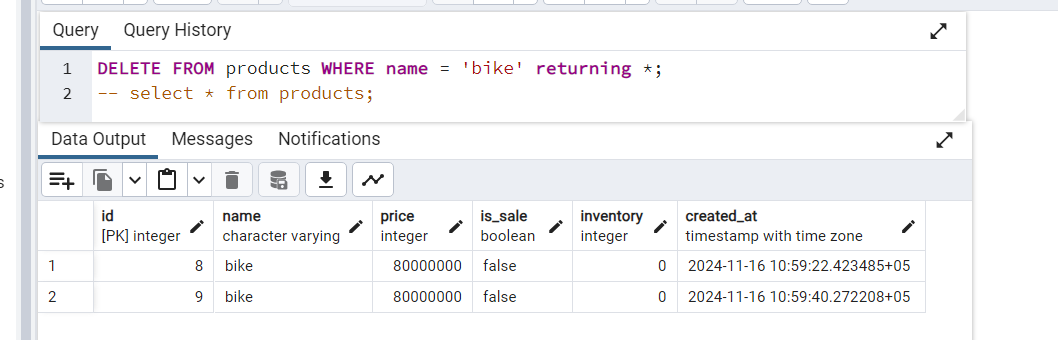
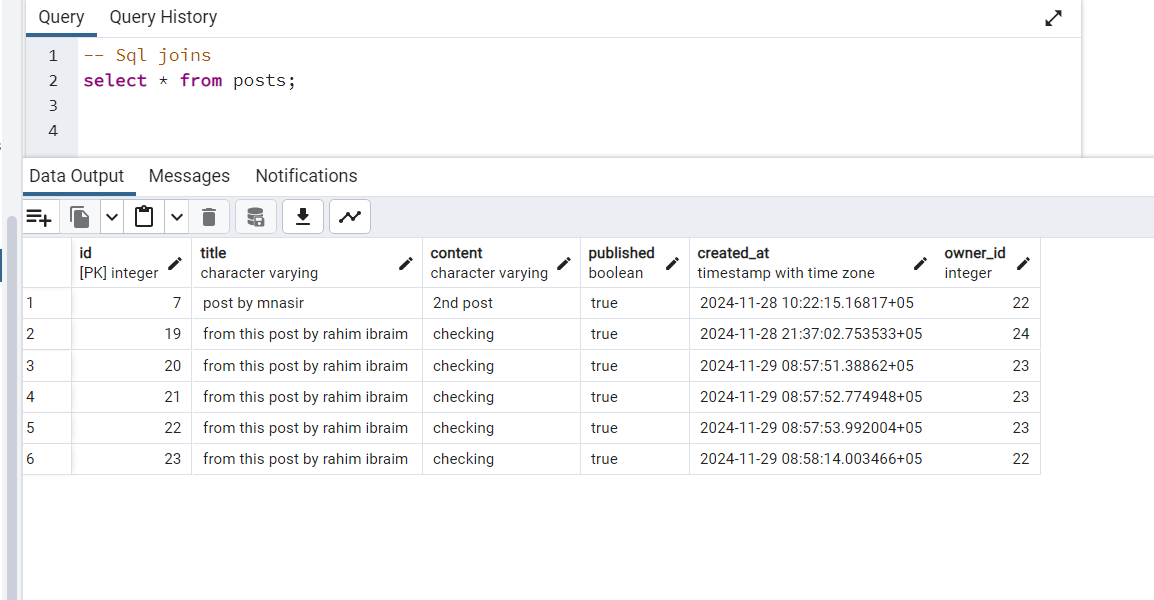
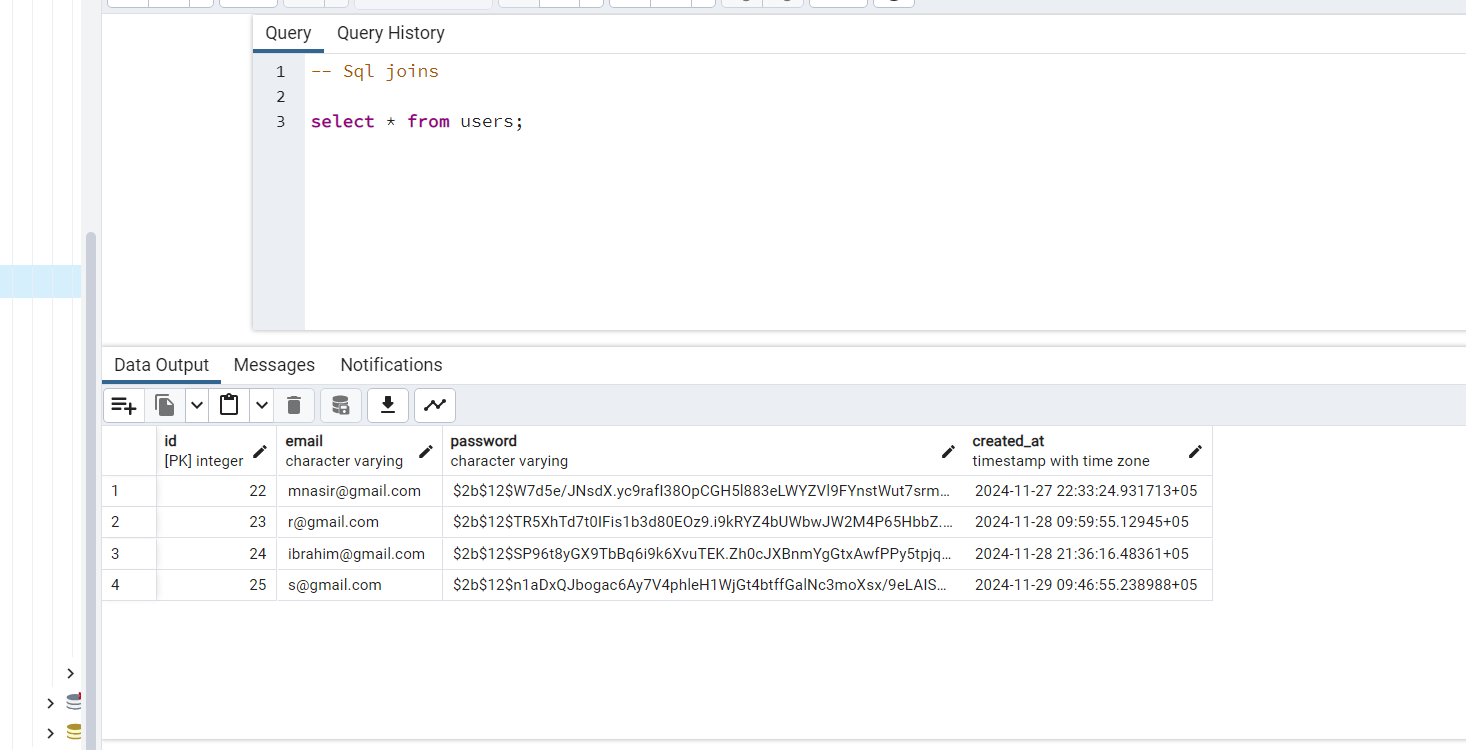
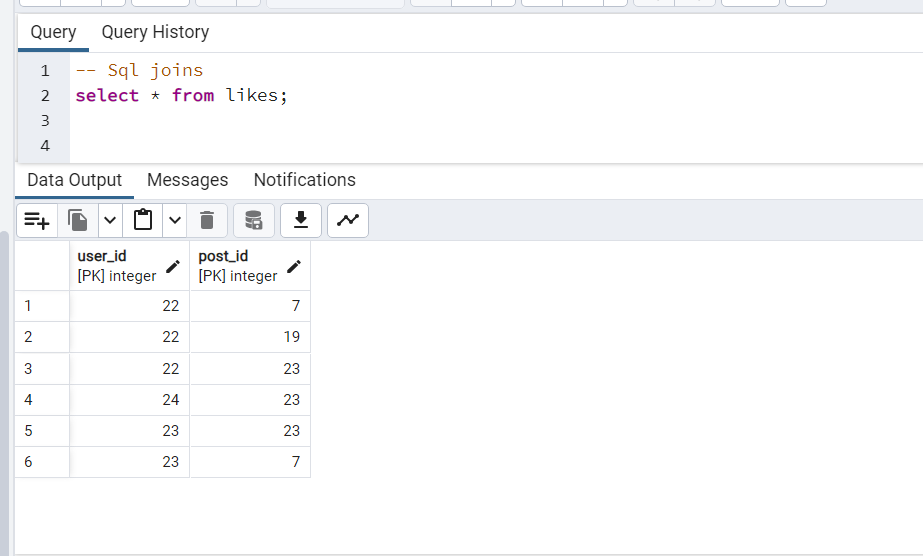
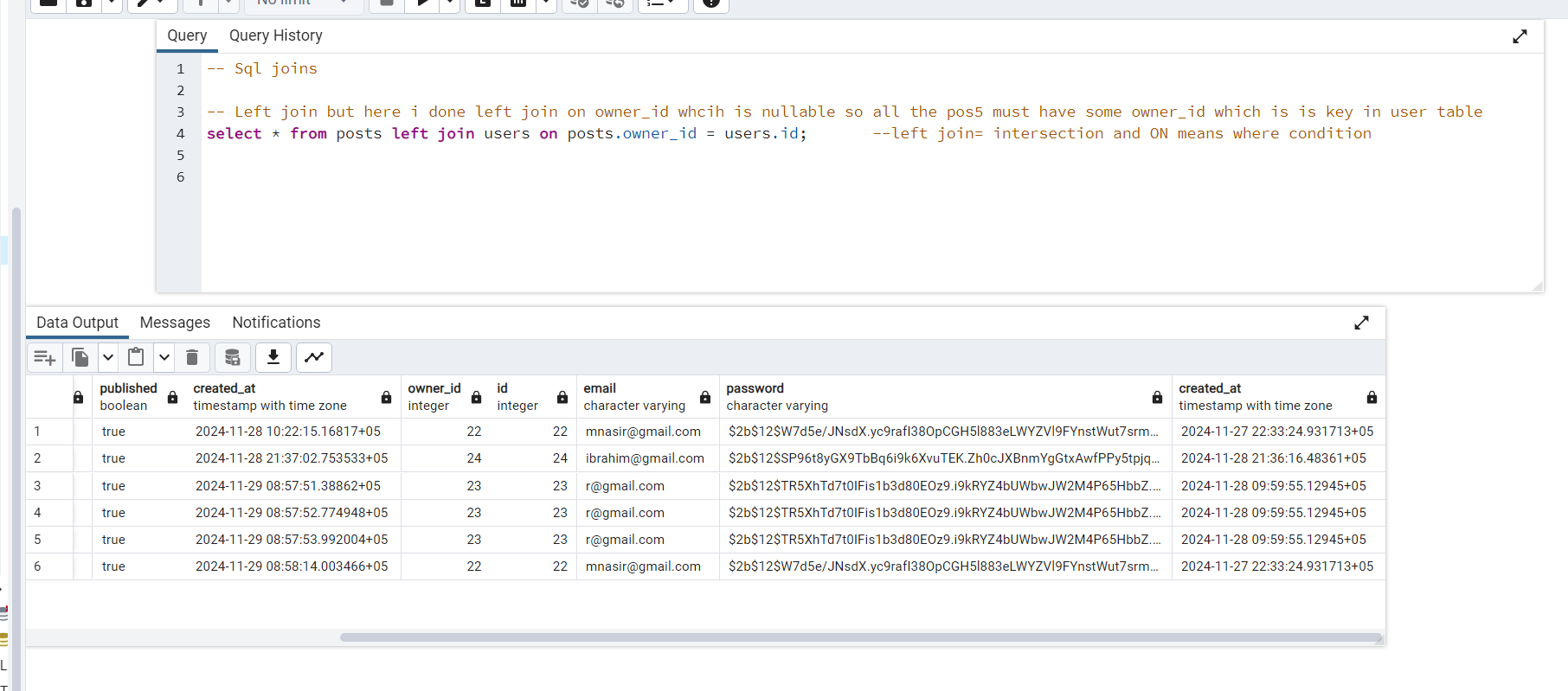
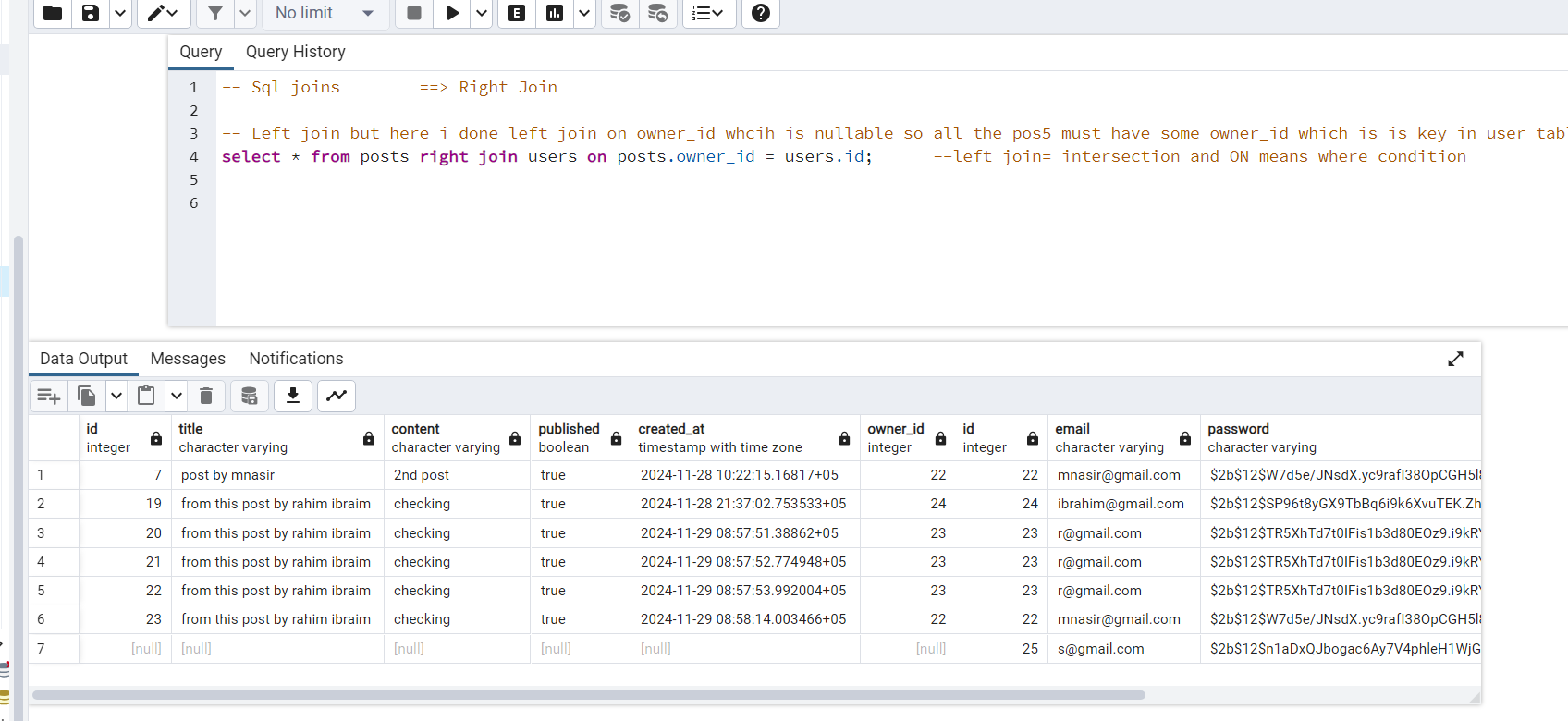
**Database queries**

1. 
2. 
3. 
4. 
5. 
6. 
7. 
8. 
9. 
10. 
11. 
12. 
13. 
14. 
15. 
16. 
17. 
18. 
19. 
20. 
21. 
22. 
23. 
24. 
25. 
26. 
27. 



1. 

Explanation of above:

Understanding how SQL queries are executed step-by-step involves understanding **query execution order**, also known as **logical query processing order**. SQL doesn't process queries from left to right as written. Instead, it follows a defined order:

**Logical Query Execution Order for:**

sql

Copy code

SELECT users.id, COUNT(posts.id)

FROM posts

RIGHT JOIN users ON posts.owner\_id = users.id

GROUP BY users.id;

**Step-by-Step Execution Flow:**

1. **FROM Clause:**
   * The first step is to identify the source tables: posts and users.
   * At this point, the database hasn't joined the tables yet—it just locates them.

**Intermediate Result:**

* + The raw data of the posts and users tables.

1. **RIGHT JOIN Clause:**
   * Combines rows from posts and users based on the condition posts.owner\_id = users.id.
   * Ensures all rows from users are included in the result, even if there's no matching row in posts.
   * Non-matching rows from posts are filled with NULL.

**Intermediate Result After Join:**

| **users.id** | **posts.id** | **owner\_id** |
| --- | --- | --- |
| 1 | 1 | 1 |
| 1 | 2 | 1 |
| 3 | 3 | 3 |
| 2 | NULL | NULL |

1. **GROUP BY Clause:**
   * Groups the joined rows by users.id.
   * For each unique users.id, it creates a group of rows.

**Grouped Data Example:**

* + Group for users.id = 1: Rows with posts.id = 1 and 2.
  + Group for users.id = 2: No rows, only user details.
  + Group for users.id = 3: Row with posts.id = 3.

1. **COUNT(posts.id) (Aggregation):**
   * For each group created in the previous step, the COUNT(posts.id) function is applied:
     + Counts the non-NULL posts.id values in each group.
     + If no posts.id exists (all NULL), the count is 0.

**Aggregated Results:**

| **users.id** | **COUNT(posts.id)** |
| --- | --- |
| 1 | 2 |
| 2 | 0 |
| 3 | 1 |

1. **SELECT Clause:**
   * Finally, the SELECT clause retrieves the columns specified: users.id and the aggregated COUNT(posts.id).

**Final Output:**

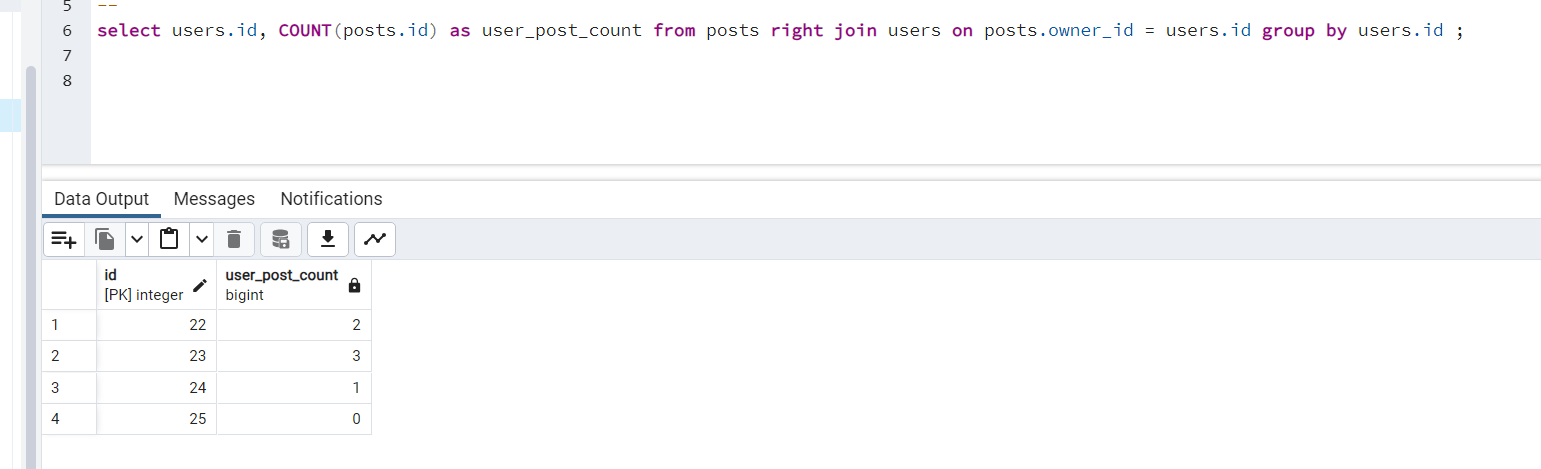
| **users.id** | **COUNT(posts.id)** |
| --- | --- |
| 1 | 2 |
| 2 | 0 |
| 3 | 1 |

**What Happens First, and What Happens Last?**

1. **First:**  
   The FROM clause loads data from the posts and users tables, and the RIGHT JOIN combines them.
2. **Intermediate:**  
   The GROUP BY clause organizes the rows into groups, one for each unique users.id.
3. **Next:**  
   The COUNT(posts.id) function calculates the number of posts for each group.
4. **Last:**  
   The SELECT clause retrieves the results to display.

This flow explains the logical processing order of your query from start to finish. **Let me know if you'd like further clarifications!**

4o

Top of Form

Bottom of Form

Aggregate functions are specialized SQL functions that perform calculations on a set of values and return a single summarized result. They are commonly used in conjunction with the GROUP BY clause to group rows and apply calculations to each group. Here are the most commonly used aggregate functions:

**Common Aggregate Functions in SQL:**

1. **COUNT()**
   * **Purpose:** Counts the number of rows or non-NULL values in a column.
   * **Example:**

sql

Copy code

SELECT COUNT(\*) FROM users;

Returns the total number of rows in the users table.

1. **SUM()**
   * **Purpose:** Calculates the total sum of a numeric column.
   * **Example:**

sql

Copy code

SELECT SUM(price) FROM orders;

Returns the total price of all orders.

1. **AVG()**
   * **Purpose:** Computes the average value of a numeric column.
   * **Example:**

sql

Copy code

SELECT AVG(salary) FROM employees;

Returns the average salary of employees.

1. **MAX()**
   * **Purpose:** Finds the maximum (highest) value in a column.
   * **Example:**

sql

Copy code

SELECT MAX(score) FROM exams;

Returns the highest score from the exams table.

1. **MIN()**
   * **Purpose:** Finds the minimum (lowest) value in a column.
   * **Example:**

sql

Copy code

SELECT MIN(age) FROM students;

Returns the youngest age in the students table.

**Use with GROUP BY:**

Aggregate functions become especially powerful when used with the GROUP BY clause to summarize data for each group. For example:

sql

Copy code

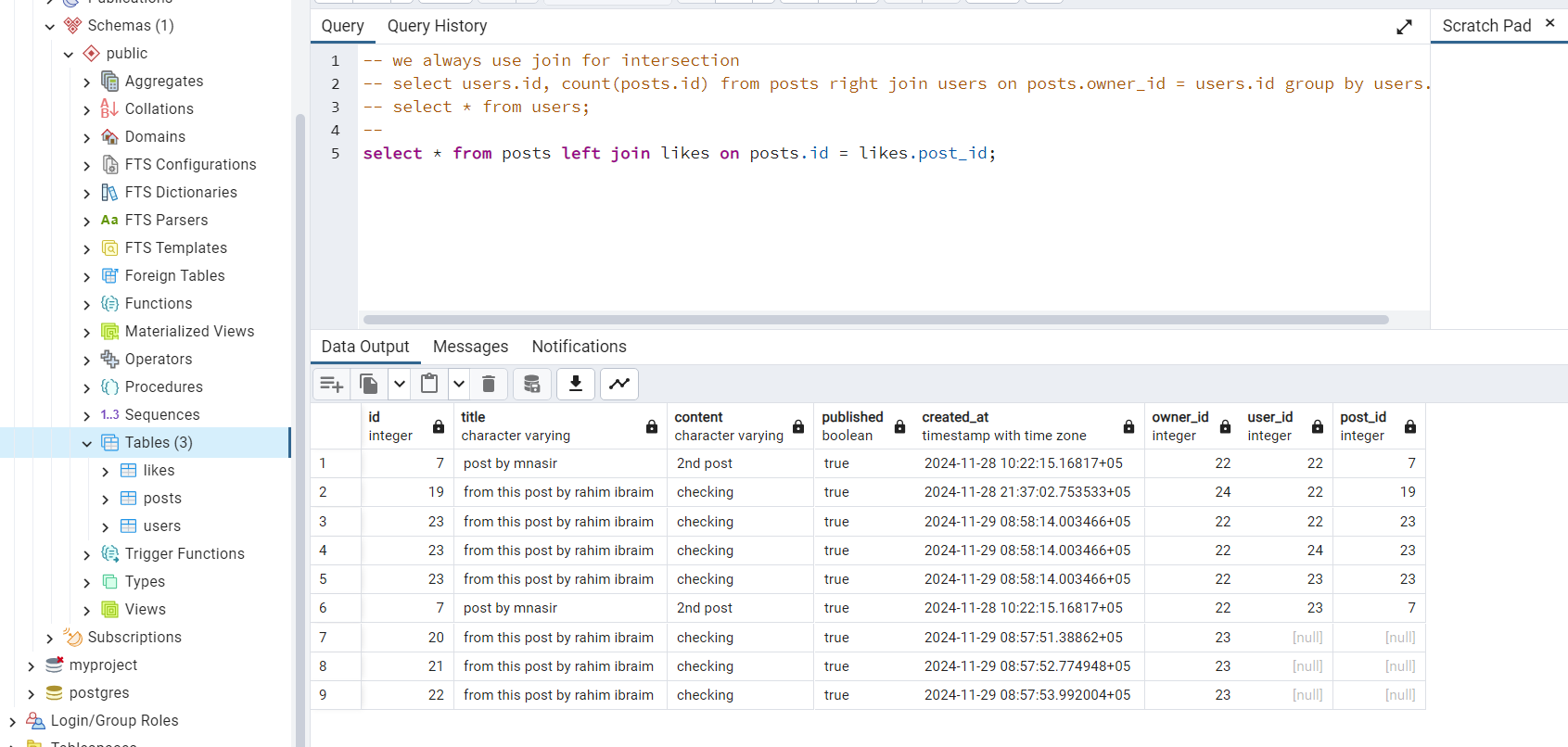
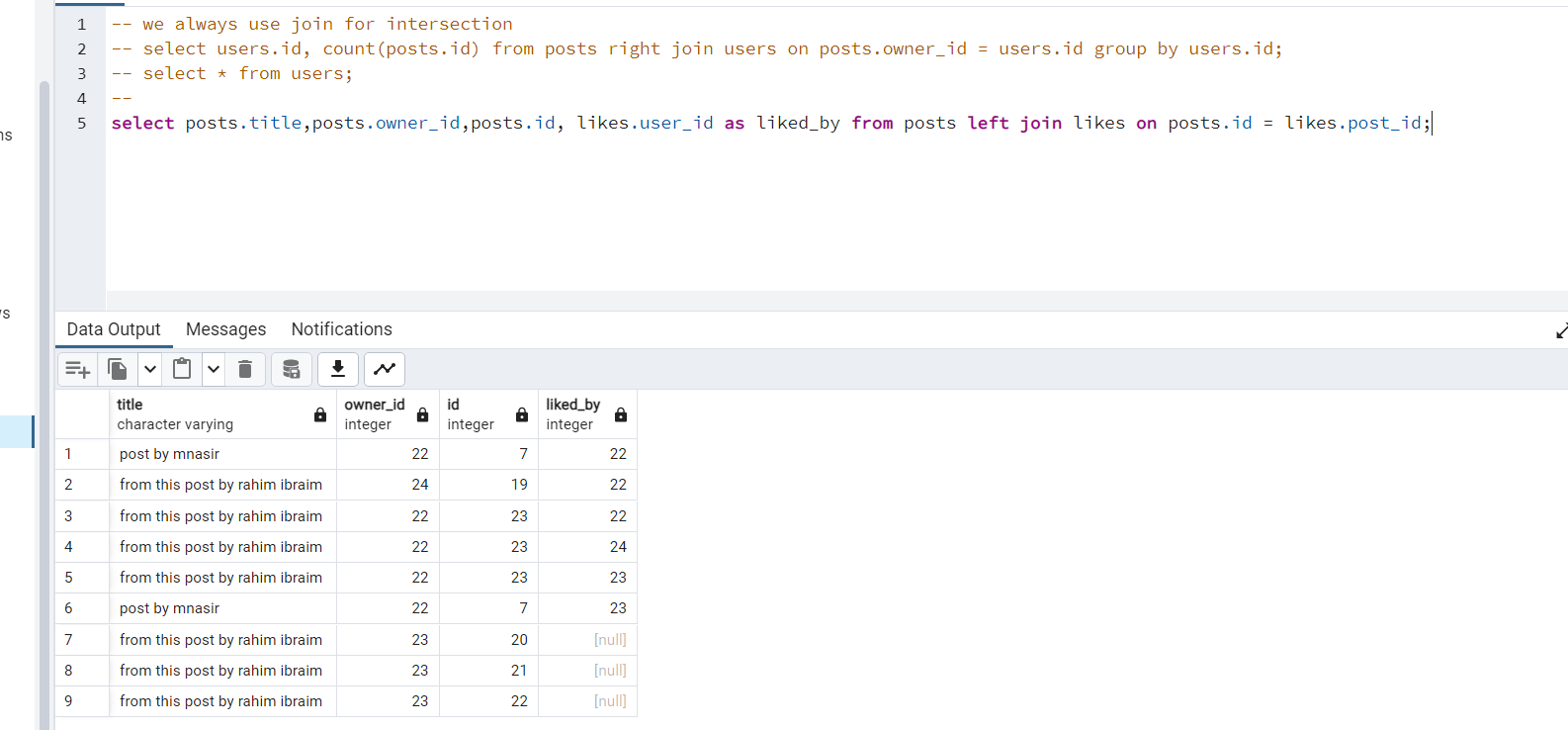
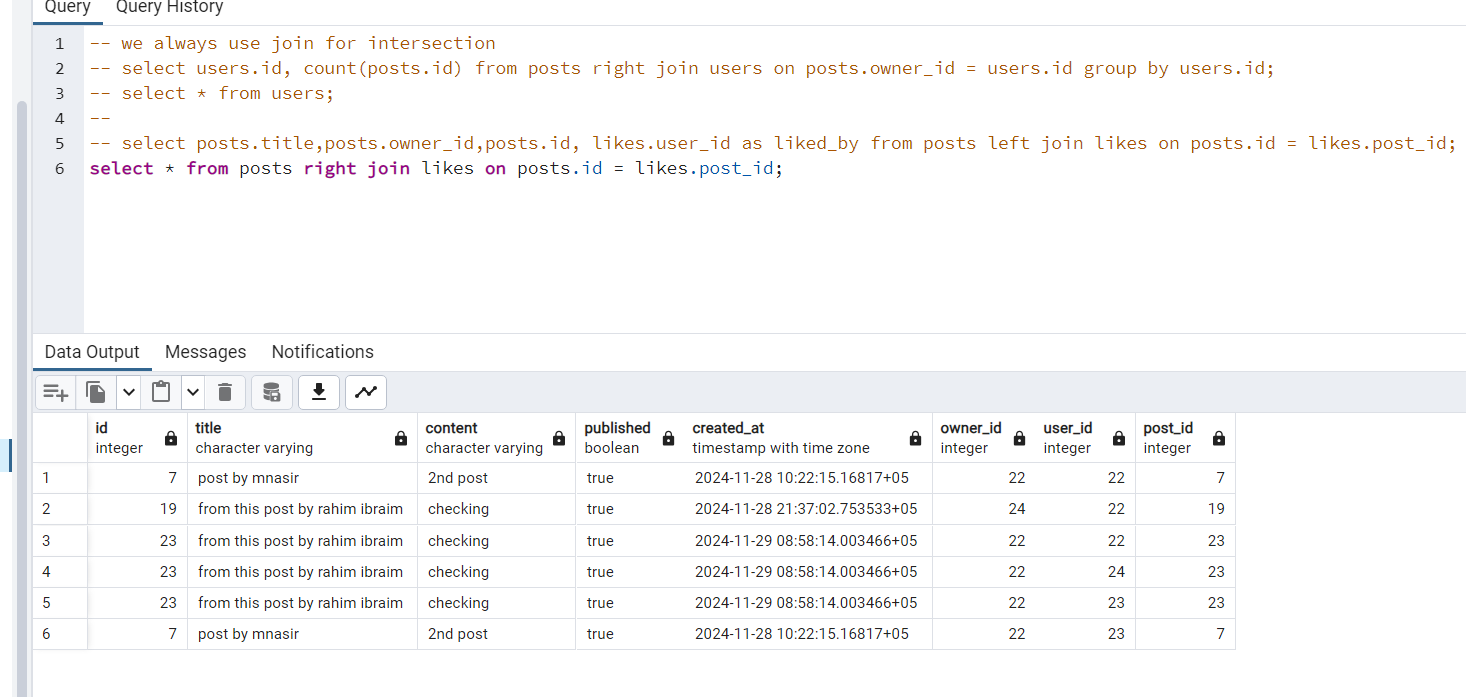
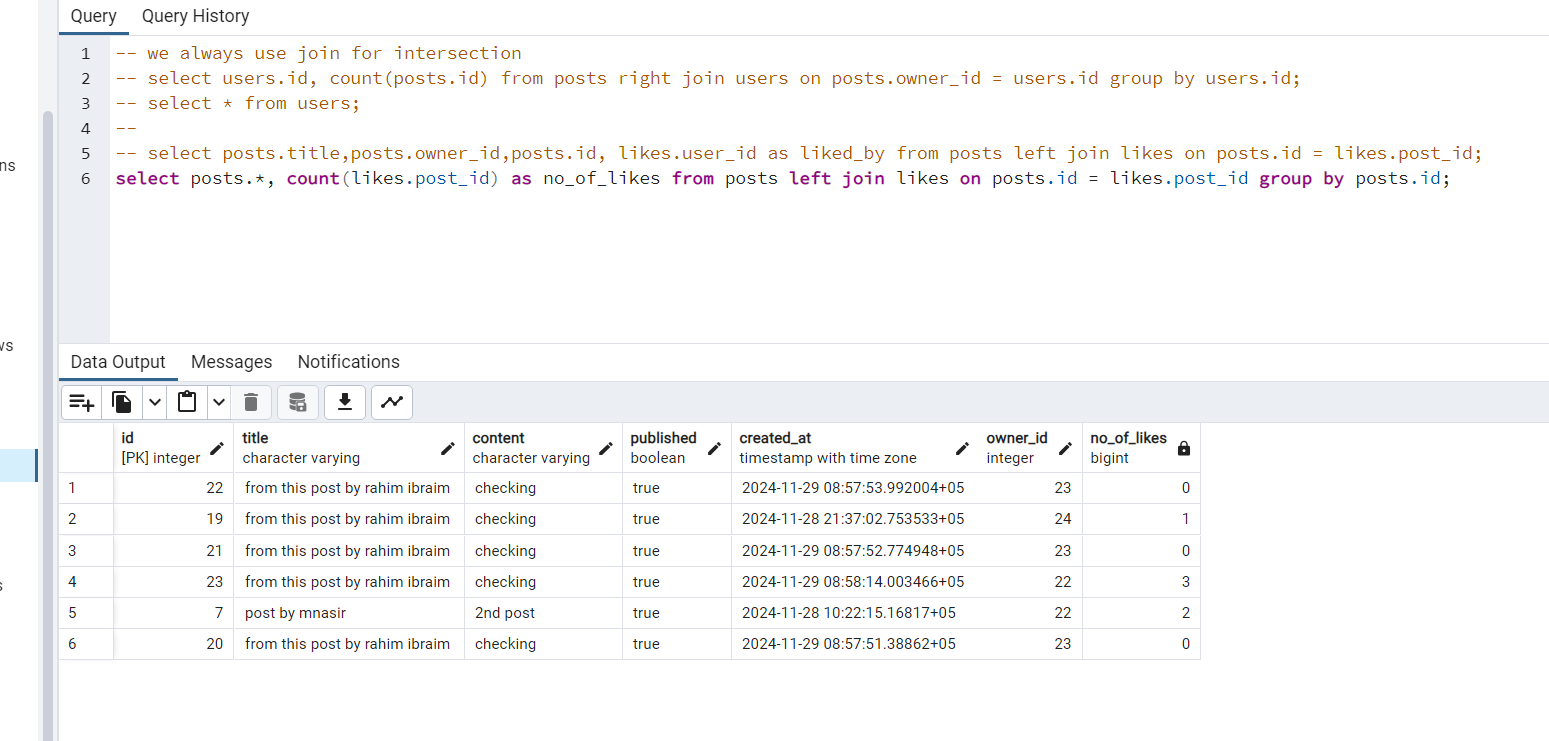
SELECT department, AVG(salary)

FROM employees

GROUP BY department;

* **Purpose:** Calculates the average salary for each department.

**Key Points About Aggregate Functions:**

1. **Ignore NULL Values:**  
   Most aggregate functions (except COUNT(\*)) ignore NULL values in the column they operate on.
2. **Combine with Other Clauses:**  
   Aggregate functions can be combined with WHERE, HAVING, and ORDER BY for more advanced filtering and sorting.
3. **Always Returns a Single Value Per Group:**  
   Each group defined in GROUP BY will have one result from the aggregate function.
4. How to see that who have liked the specific post 
5. Now I want to only give the important attributes not all
6. Using Right join and it shows only the liked posts 
7. here in above it provide som wroing outputs a it count only roes
8. In this it count right because here in count I wan tot count post\_id of like
9. in the above is the original thinfg mean all nfo regarding post an d their no. of likes
10. 