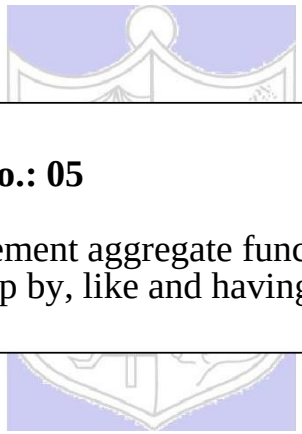


Experiment No.: 05

Title: To implement aggregate functions with order by, group by, like and having clause.

Batch:SY-IT(B3) **Roll No.:16010423076** **Experiment No: 05**

Aim: To implement aggregate functions with order by, group by, like and having clause.

Resources needed: PostgreSQL PgAdmin4

Theory:

The ORDER BY clause is used to sort the data in ascending or descending order, based on one or more columns.

```
SELECT column-list
FROM table_name
[WHERE condition]
[ORDER BY column1, column2, .. columnN] [ASC | DESC];
```

The GROUP BY clause is used in collaboration with the SELECT statement to group together those rows in a table that have identical data. This is done to eliminate redundancy in the output and/or compute aggregates that apply to these groups.

The GROUP BY clause follows the WHERE clause in a SELECT statement and precedes the ORDER BY clause.

```
SELECT column-list
FROM table_name
WHERE [ conditions ]
GROUP BY column1, column2....columnN
ORDER BY column1, column2....columnN
```

The LIKE operator is used to match text values against a pattern using wildcards. If the search expression can be matched to the pattern expression, the LIKE operator will return true, which is 1. There are two wildcards used in conjunction with the LIKE operator:

- The percent sign (%)
- The underscore (_)

The percent sign represents zero, one, or multiple numbers or characters. The underscore represents a single number or character. These symbols can be used in combinations.

If either of these two signs is not used in conjunction with the LIKE clause, then the LIKE acts like the equals operator.

```
SELECT FROM table_name
WHERE column LIKE 'XXXX%'
or
SELECT FROM table_name
```

```
WHERE column LIKE '%XXXX%'

or

SELECT FROM table_name

WHERE column LIKE 'XXXX_'

or

SELECT FROM table_name

WHERE column LIKE '_XXXX'

or

SELECT FROM table_name

WHERE column LIKE '_XXXX_'
```

Here are examples showing WHERE part having different LIKE clause with '%' and '_' operators:

Statement	Description
WHERE SALARY::text LIKE '200%'	Finds any values that start with 200
WHERE SALARY::text LIKE '%200%'	Finds any values that have 200 in any position
WHERE SALARY::text LIKE '_00%'	Finds any values that have 00 in the second and third positions
WHERE SALARY::text LIKE '2_%_%'	Finds any values that start with 2 and are at least 3 characters in length
WHERE SALARY::text LIKE '%2'	Finds any values that end with 2
WHERE SALARY::text LIKE '_2%3'	Finds any values that have a 2 in the second position and end with a 3
WHERE SALARY::text LIKE '2___3'	Finds any values in a five-digit number that start with 2 and end with 3

The HAVING clause allows us to pick out particular rows where the function's result meets

some condition.
The WHERE clause places conditions on the selected columns, whereas the HAVING clause places conditions on groups created by the GROUP BY clause.

```
SELECT column1, column2
FROM table1, table2
WHERE [ conditions ]
GROUP BY column1, column2
HAVING [ conditions ]
ORDER BY column1, column2
```

Results: (Queries printout with output)

1. Write 13 queries using ‘order by’, ‘group by’, ‘like’ and ‘having’ clause.
   5 with normal aggregate fun,3 with clauses and aggregate function and 5 with like operator

Example:


```
1.   SELECT * FROM COMPANY ORDER BY NAME, SALARY ASC;
2.   SELECT NAME, SUM(SALARY) FROM COMPANY GROUP BY NAME;
3.   SELECT * FROM COMPANY WHERE AGE::text LIKE '2%';
4.   SELECT * FROM COMPANY WHERE ADDRESS  LIKE '%-%';
5.   SELECT NAME FROM COMPANY GROUP BY name HAVING count(name) > 1;
```

Outcomes:

Altering the Table to make it have 10 Rows

```
-- Update existing rows with city names and distances
UPDATE routes SET
   origin = 'Delhi',
   destination = 'Mumbai',
   distance = 1400
WHERE route_id = 1;
```

```
UPDATE routes SET
   origin = 'Kolkata',
   destination = 'Chennai',
   distance = 1650
WHERE route_id = 2;

UPDATE routes SET
   origin = 'Bengaluru',
   destination = 'Hyderabad',
   distance = 600
WHERE route_id = 3;

UPDATE routes SET
   origin = 'Pune',
   destination = 'Goa',
   distance = 450
WHERE route_id = 4;

UPDATE routes SET
   origin = 'Jaipur',
   destination = 'Agra',
   distance = 240
WHERE route_id = 5;

-- Insert additional rows with city names and distances
INSERT INTO routes (route_id, origin, destination, distance) VALUES
```

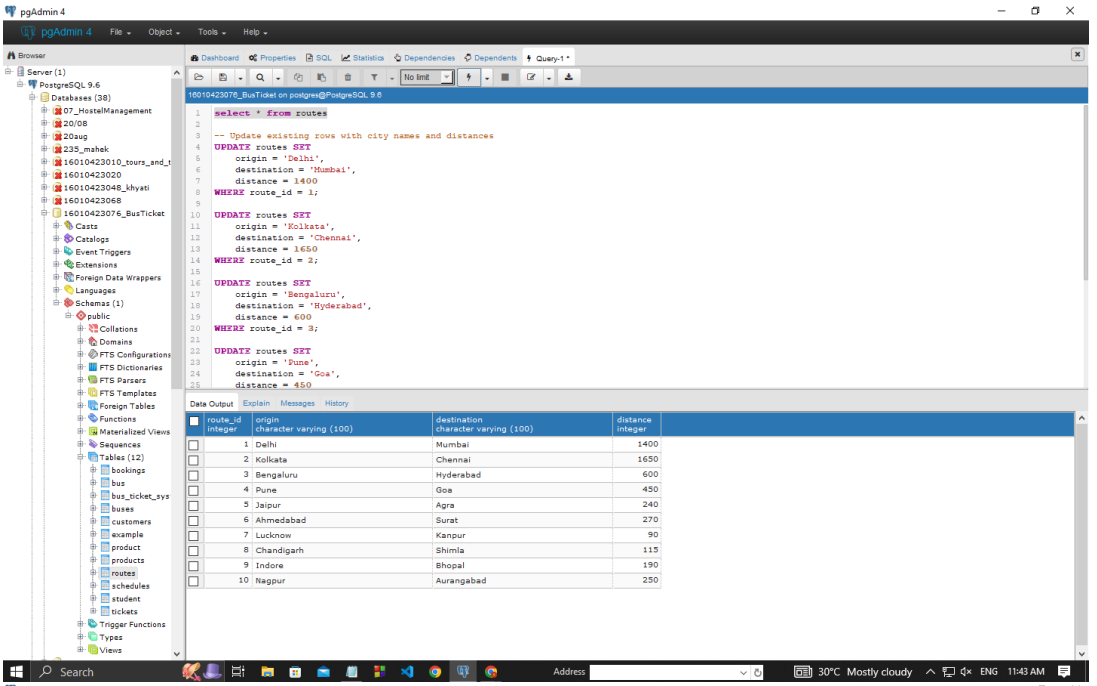
(6, 'Ahmedabad', 'Surat', 270),

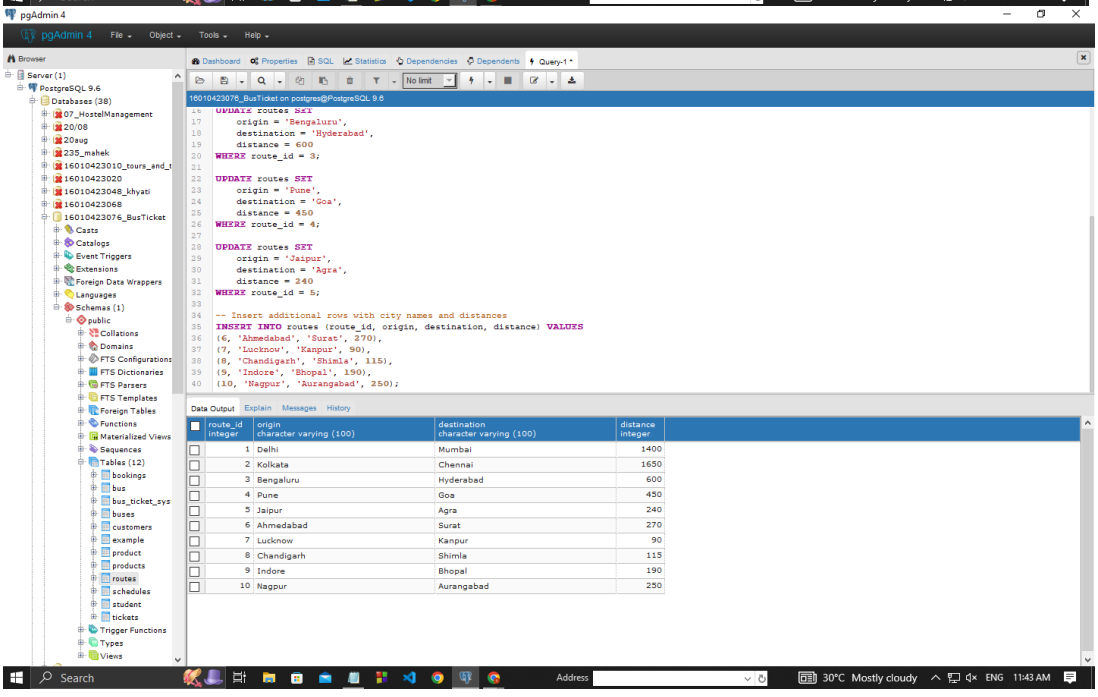
(7, 'Lucknow', 'Kanpur', 90),

(8, 'Chandigarh', 'Shimla', 115),

(9, 'Indore', 'Bhopal', 190),

(10, 'Nagpur', 'Aurangabad', 250);

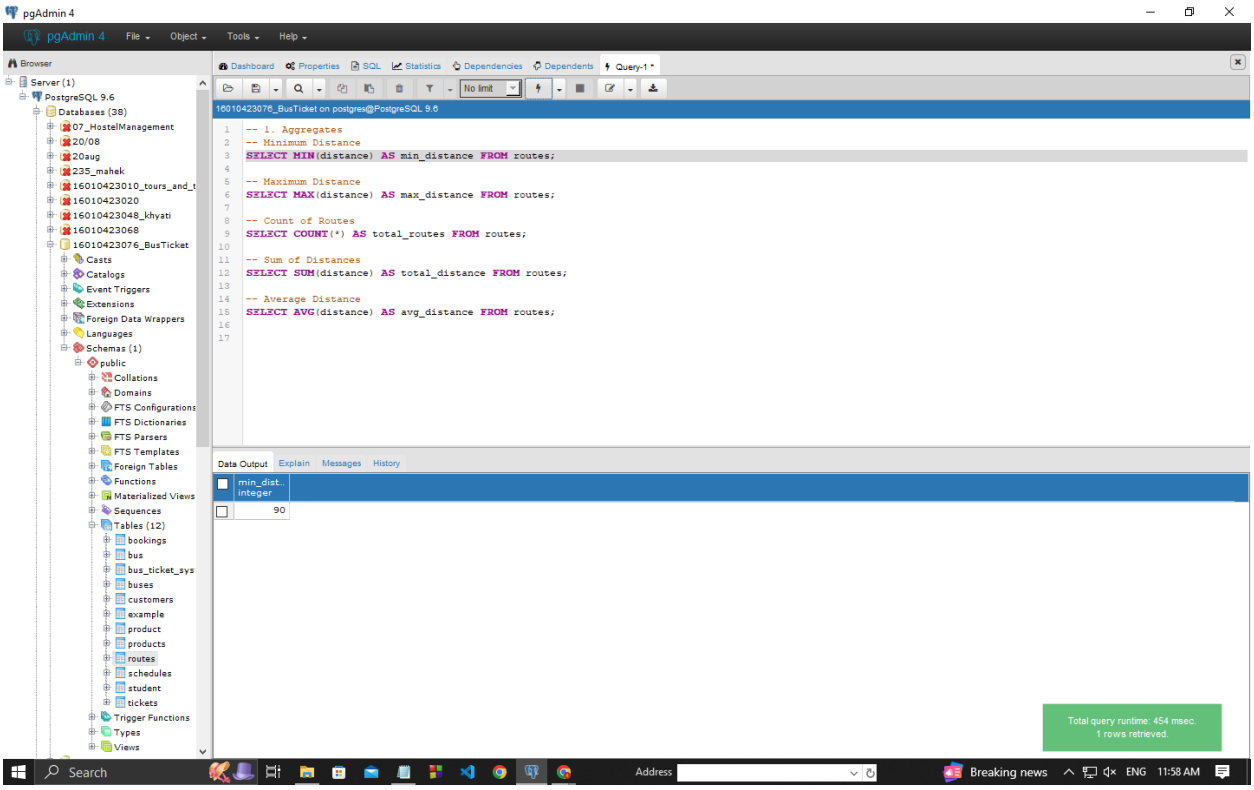




1. Simple Aggregates

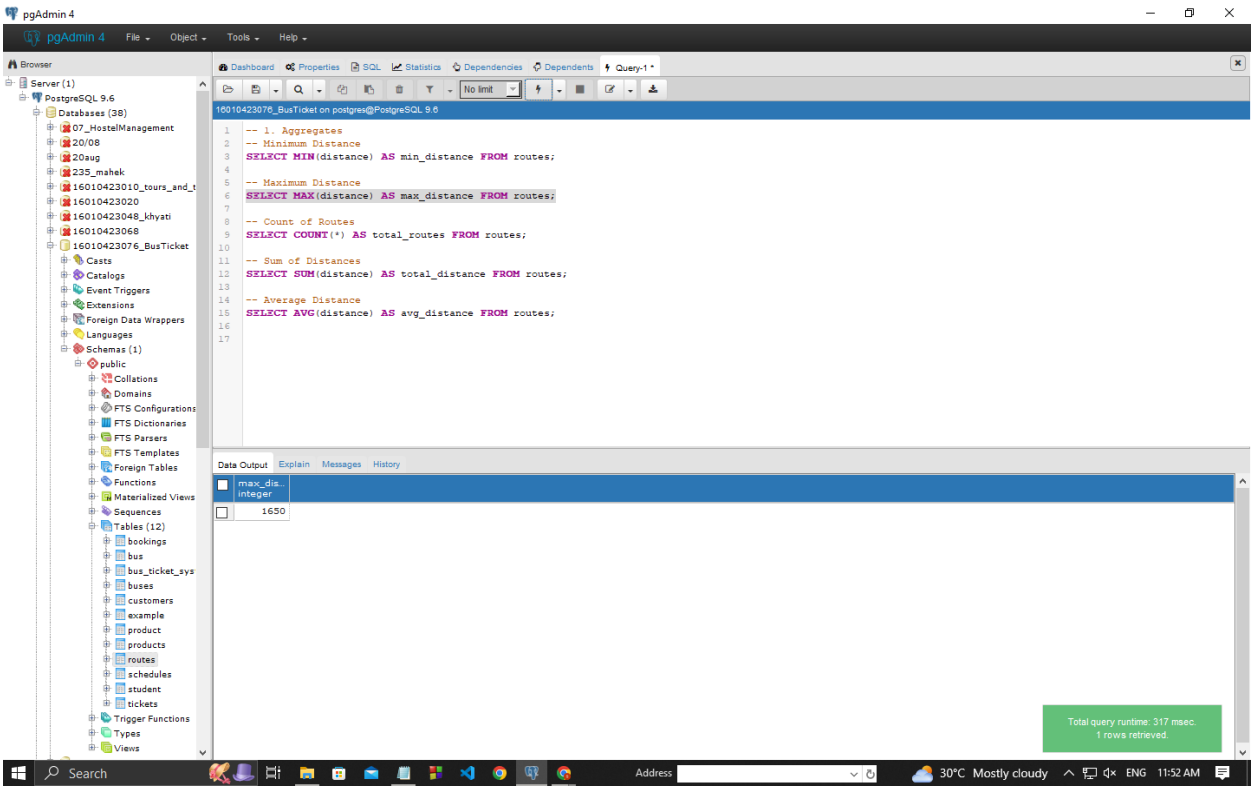
-- Minimum Distance

SELECT MIN(distance) AS min_distance FROM routes;



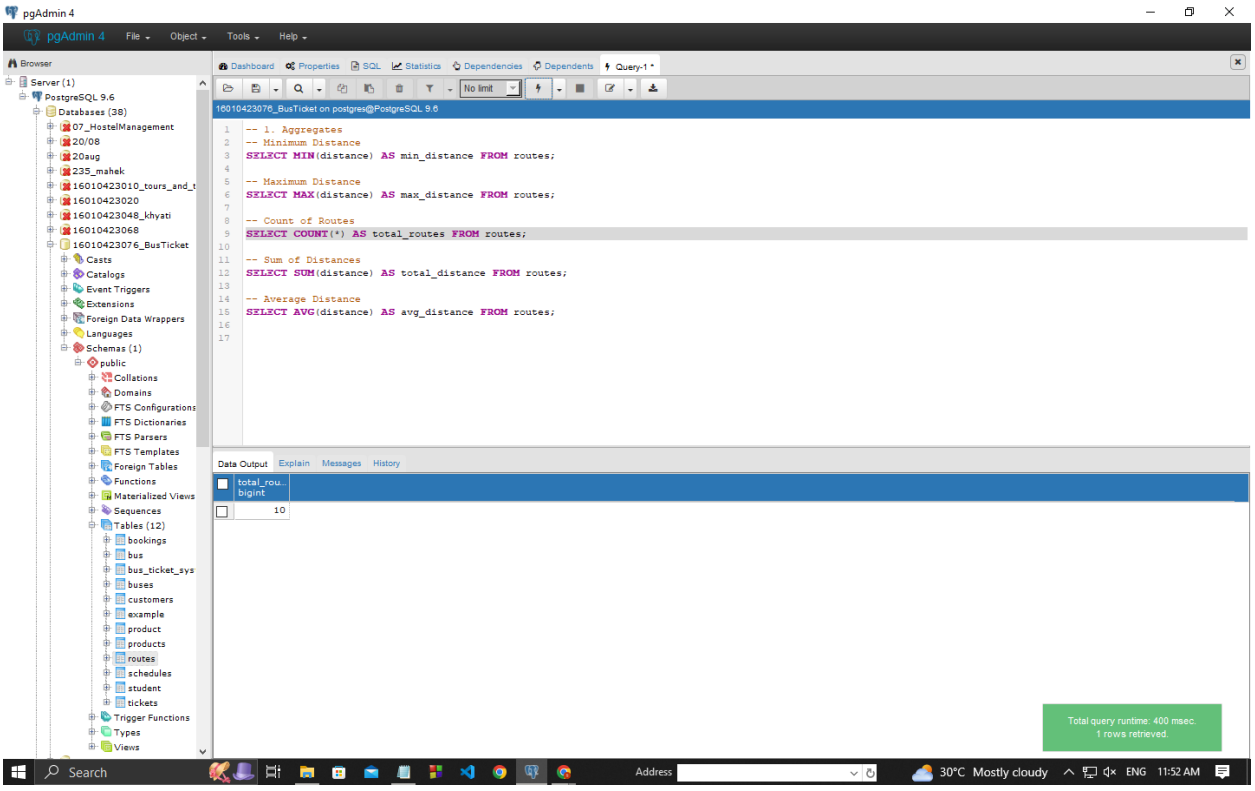
-- Maximum Distance

SELECT MAX(distance) AS max_distance FROM routes;



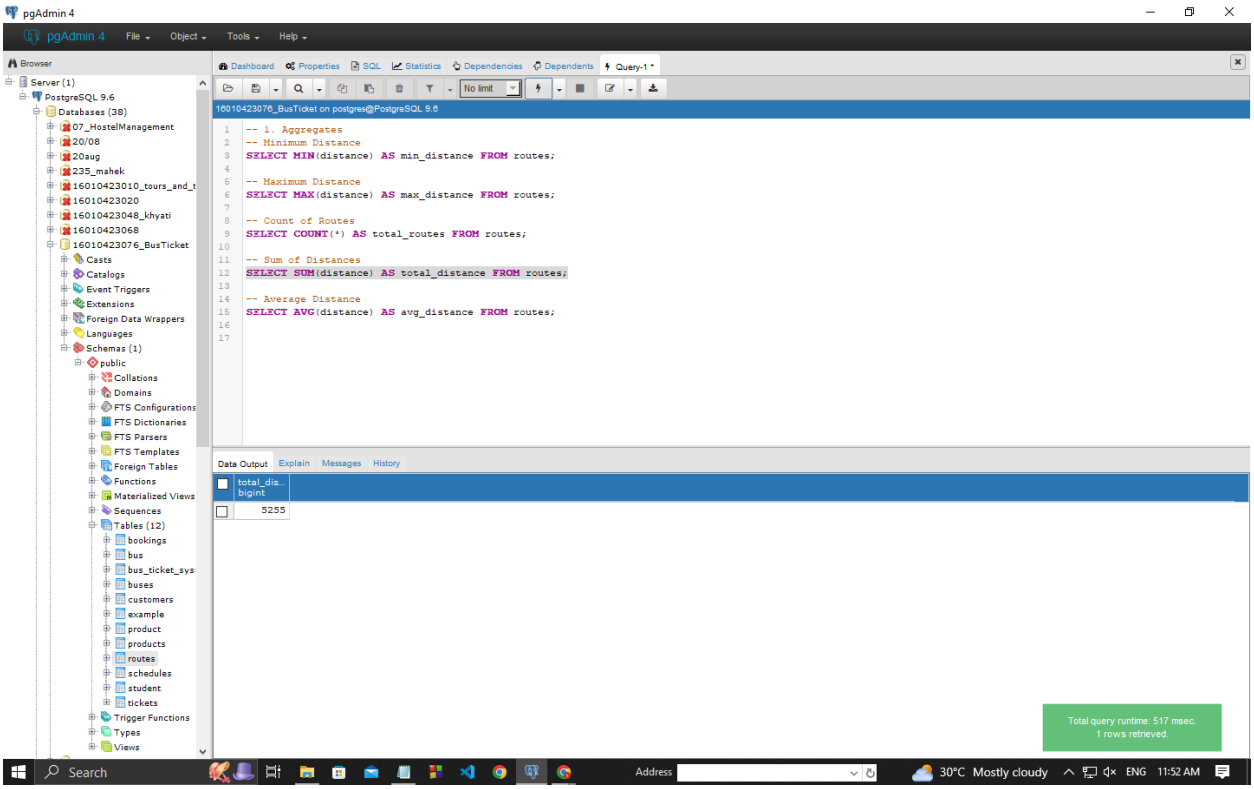
-- Count of Routes

SELECT COUNT(*) AS total_routes FROM routes;



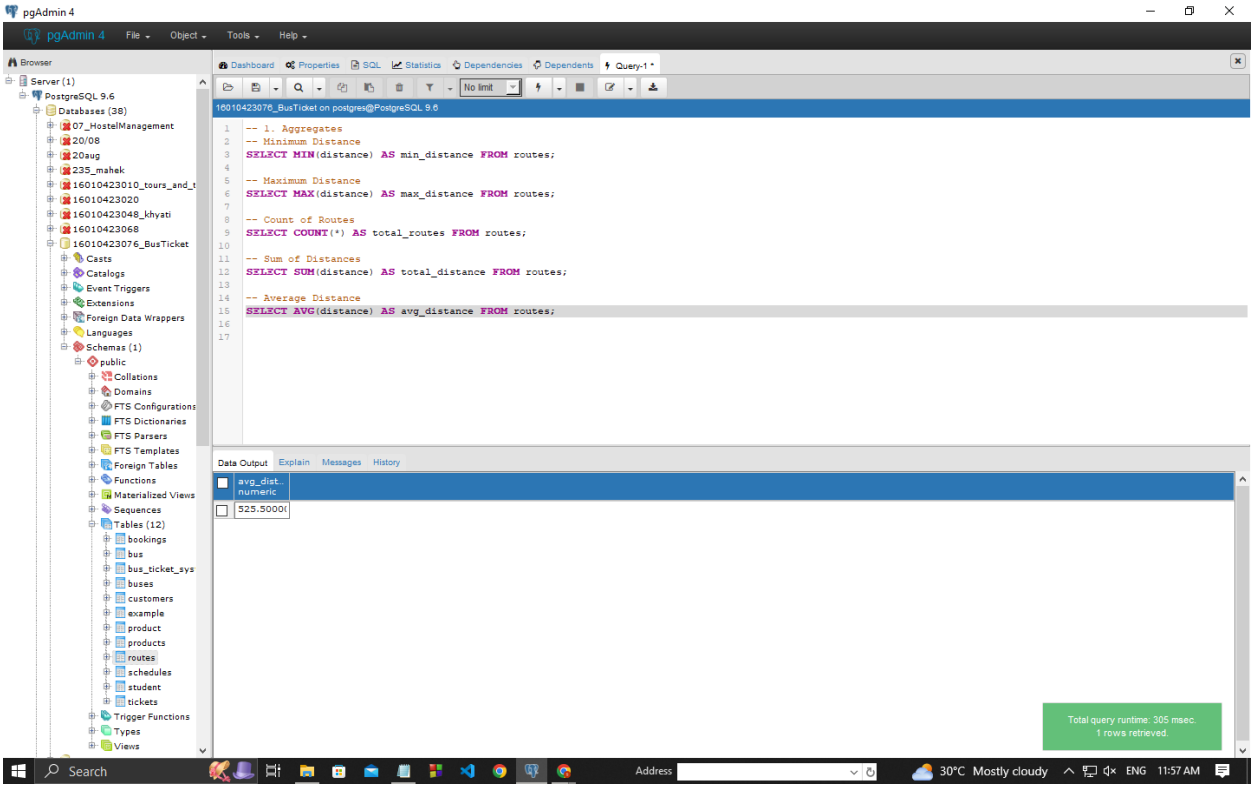
-- Sum of Distances

SELECT SUM(distance) AS total_distance FROM routes;



-- Average Distance

SELECT AVG(distance) AS avg_distance FROM routes;

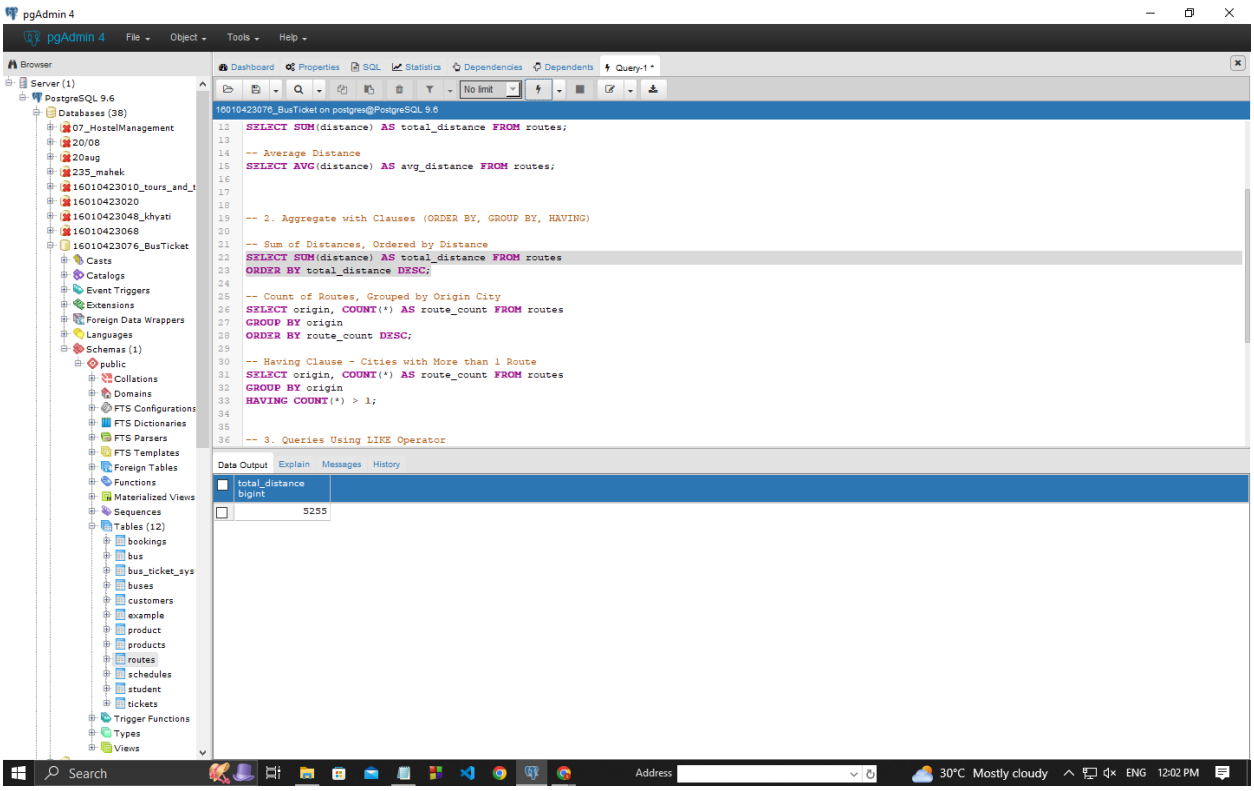


2. Aggregate with Clauses (ORDER BY, GROUP BY, HAVING)

-- Sum of Distances, Ordered by Distance

SELECT SUM(distance) AS total_distance FROM routes

ORDER BY total_distance DESC;

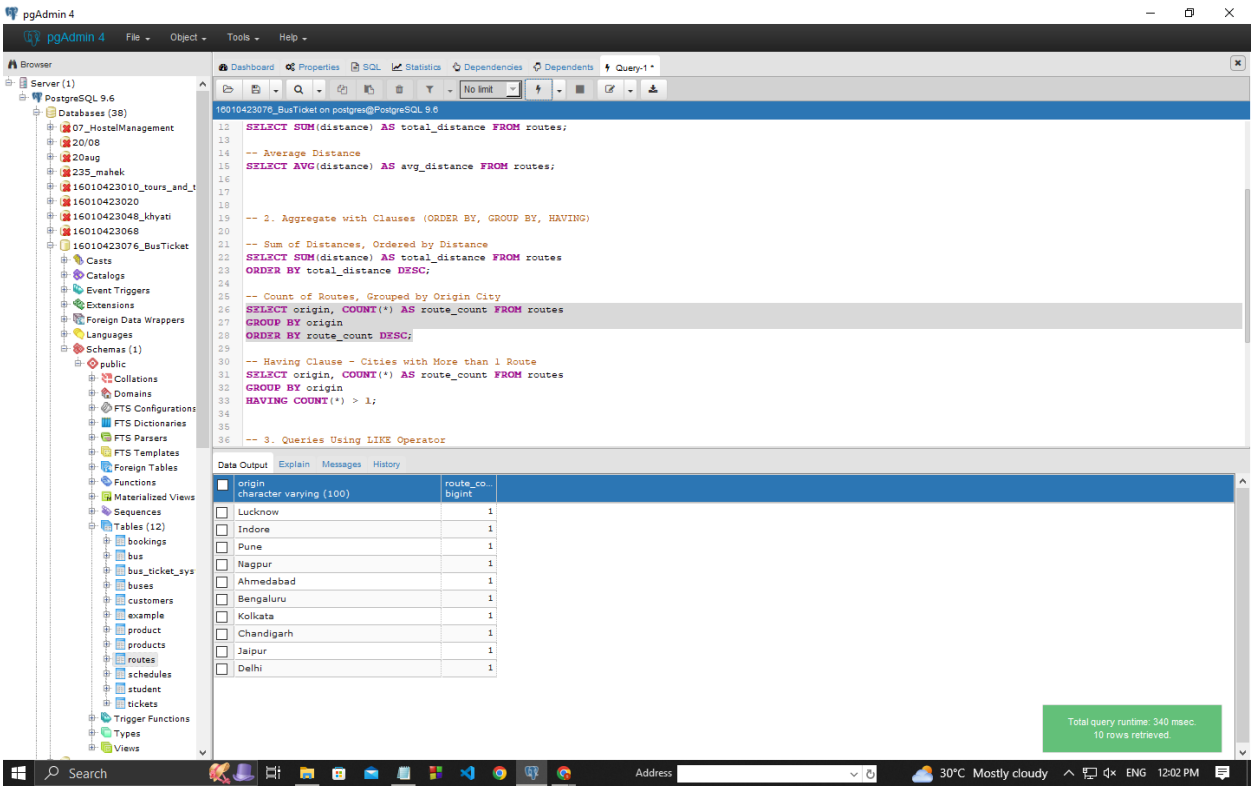


-- Count of Routes, Grouped by Origin City

SELECT origin, COUNT(*) AS route_count FROM routes

GROUP BY origin

ORDER BY route_count DESC;

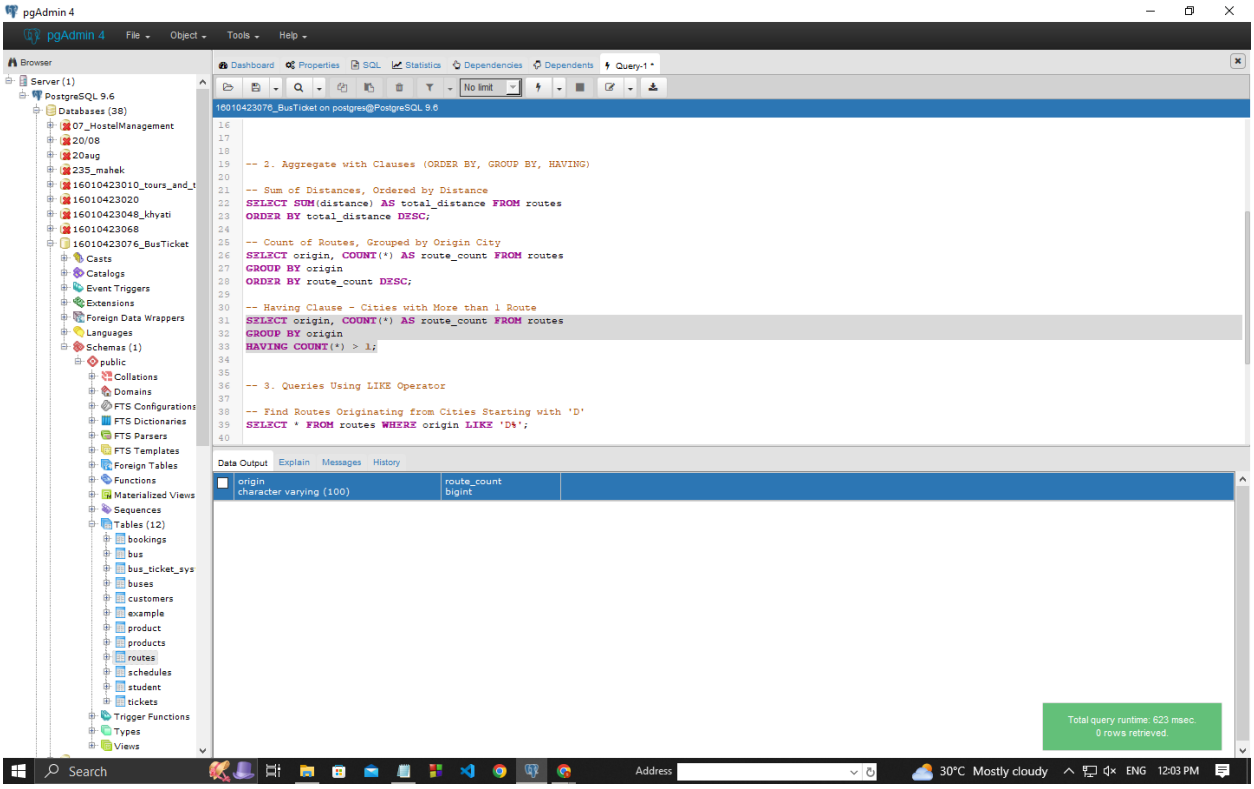


-- Having Clause - Cities with More than 1 Route

SELECT origin, COUNT(*) AS route_count FROM routes

GROUP BY origin

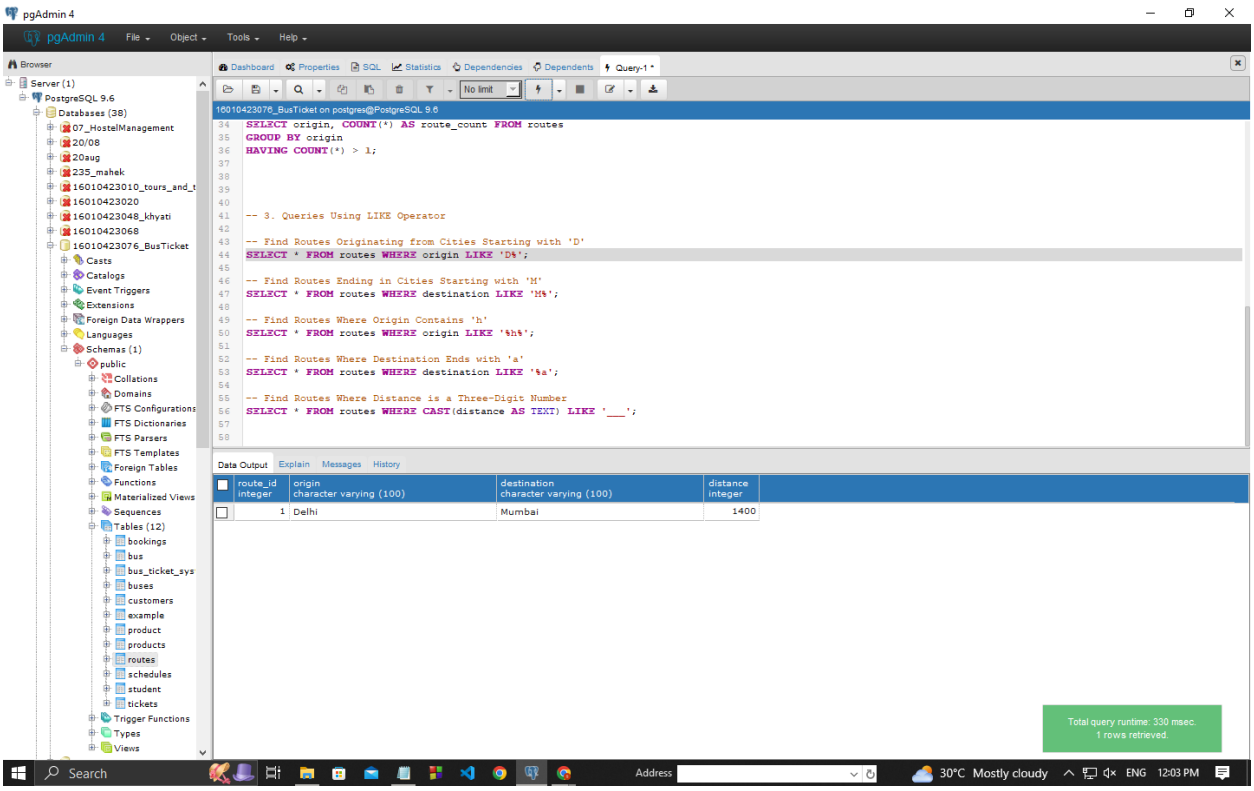
HAVING COUNT(*) > 1;



3. Queries Using LIKE Operator

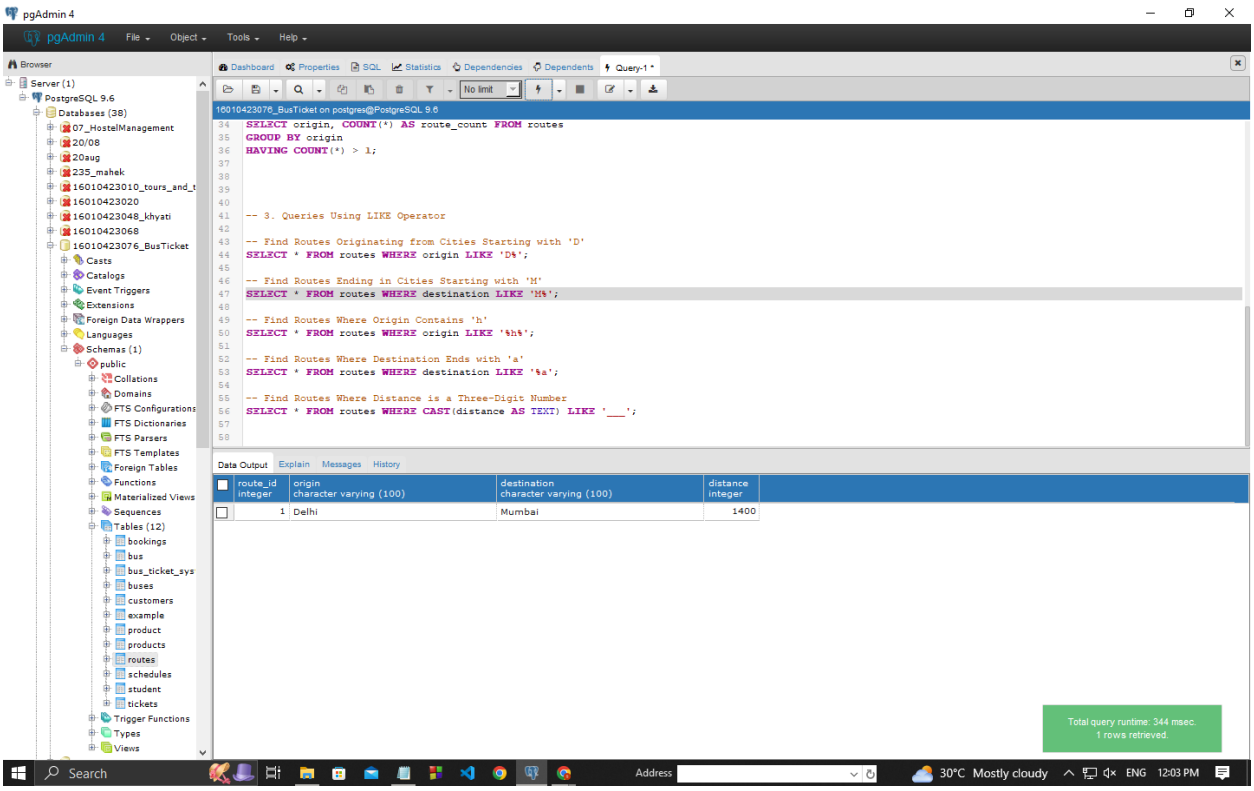
-- Find Routes Originating from Cities Starting with 'D'

SELECT * FROM routes WHERE origin LIKE 'D%';



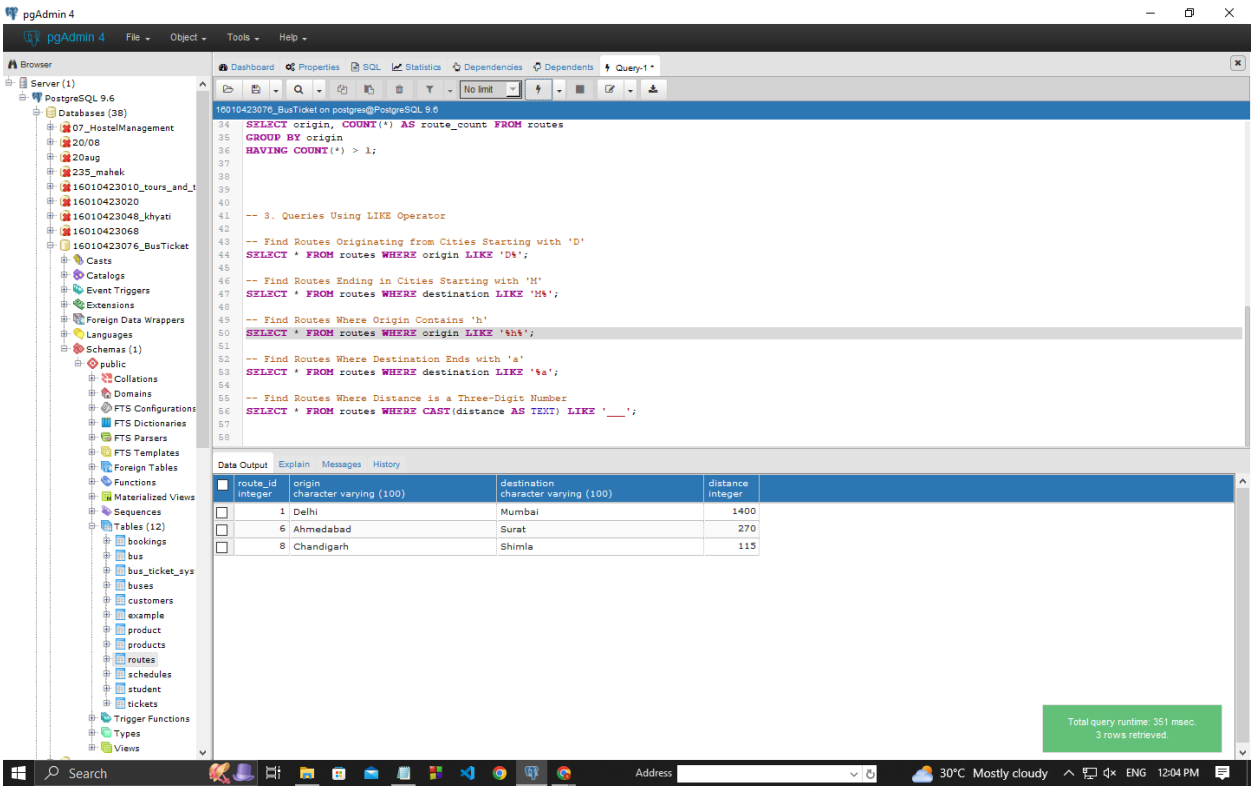
-- Find Routes Ending in Cities Starting with 'M'

SELECT * FROM routes WHERE destination LIKE 'M%';



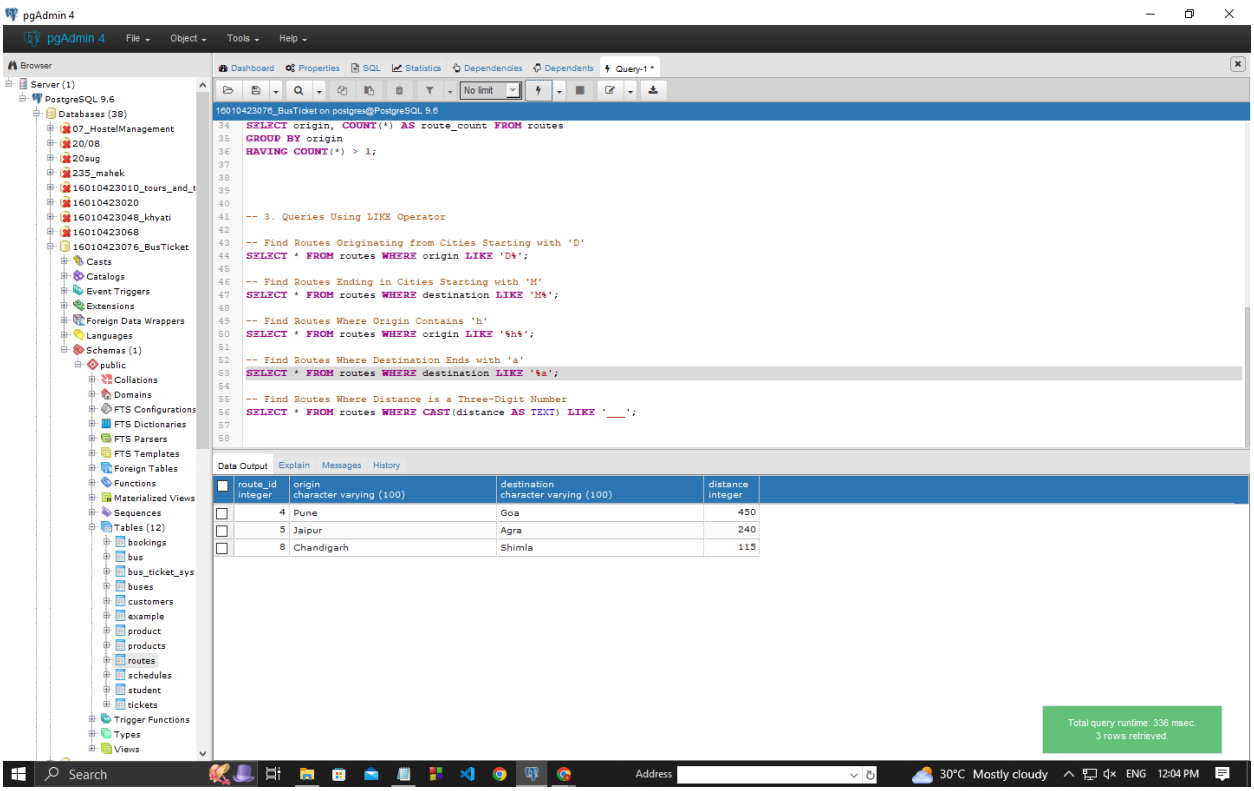
-- Find Routes Where Origin Contains 'h'

SELECT * FROM routes WHERE origin LIKE '%h%';



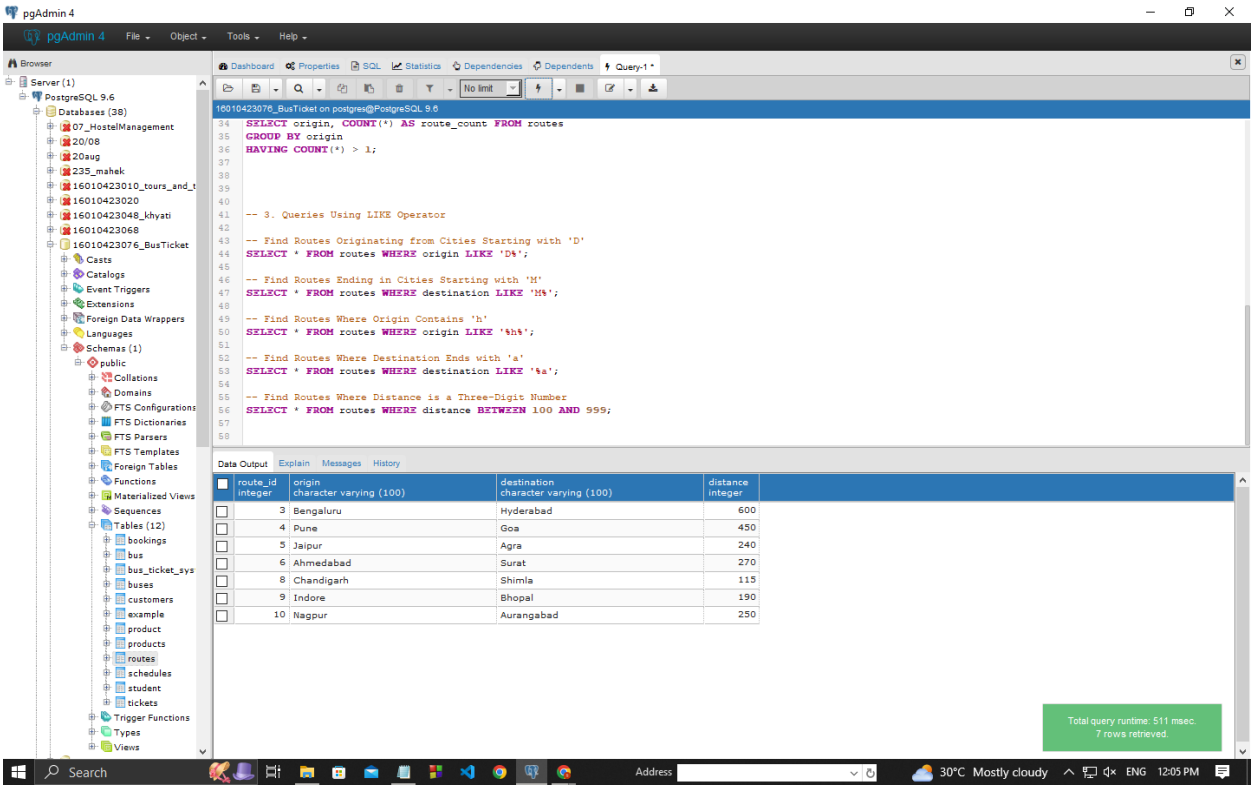
-- Find Routes Where Destination Ends with 'a'

SELECT * FROM routes WHERE destination LIKE '%a';



-- Find Routes Where Distance is a Three-Digit Number

```
SELECT * FROM routes WHERE CAST(distance AS TEXT) LIKE '___';
```



Questions:

Q1 Can you apply like operator on integer value? explain with example how?

In PostgreSQL (and SQL in general), the LIKE operator is primarily used for pattern matching on string values. It's not directly applicable to integer values because integers are not inherently text-based and do not have patterns in the same sense as strings do.

However, you can convert integers to strings and then apply the LIKE operator. This is done using the CAST or ::text operation to convert the integer to a text type. Here's an example:

```
-- Sample table
CREATE TABLE numbers (
    id SERIAL PRIMARY KEY,
    value INT
);
```

```
-- Insert some values
INSERT INTO numbers (value) VALUES (123), (456), (789);

-- Query using LIKE operator on integer values
SELECT * FROM numbers
WHERE value::text LIKE '12%';
```

In this example, the integer values are cast to text using value::text, and then the LIKE operator is used to find values where the text representation of the number starts with '12'.

Q2 Why aggregate functions are more used with order by, group by and having clauses? Can we change order of these clauses when used in single query

Aggregate functions (like SUM(), COUNT(), AVG(), etc.) are typically used in conjunction with GROUP BY, ORDER BY, and HAVING clauses in SQL queries for several reasons:

- GROUP BY: This clause groups rows that have the same values into summary rows, like "total sales per region". Aggregate functions are then applied to these groups.
- HAVING: This clause filters groups based on a condition, similar to WHERE, but it operates on aggregated data. For example, you might want to find regions where the total sales exceed a certain amount.
- ORDER BY: This clause sorts the result set. After aggregation, you might want to order the results based on the aggregated values, like sorting by total sales in descending order.

```
-- Sample table
CREATE TABLE sales (
    region TEXT,
    amount NUMERIC
);

-- Insert some values
INSERT INTO sales (region, amount) VALUES
('North', 100),
('North', 200),
('South', 150),
('South', 50),
('East', 300),
('West', 250);

-- Query using GROUP BY, HAVING, and ORDER BY
SELECT region, SUM(amount) AS total_sales
FROM sales
GROUP BY region
HAVING SUM(amount) > 100
ORDER BY total_sales DESC;
```

In this example:

- GROUP BY is used to aggregate sales by region.
- HAVING filters out regions where total sales are less than or equal to 100.
- ORDER BY sorts the results by total_sales in descending order.

Clause Order:
In a single SQL query, the order of these clauses is fixed and cannot be changed:
SELECT
FROM
WHERE
GROUP BY
HAVING
ORDER BY

The order is important because SQL processes the clauses in this sequence. For example, GROUP BY must come before HAVING because HAVING is meant to filter groups created by GROUP BY.

Conclusion:
I learned how to update and insert rows in SQL tables, use simple aggregate functions like SUM and COUNT, and apply conditions using GROUP BY, HAVING, and ORDER BY. I also understood how to use the LIKE operator on strings and convert integers to text for pattern matching. Overall, it enhanced my understanding of querying and manipulating data in SQL.

Grade: AA / AB / BB / BC / CC / CD /DD

Signature of faculty in-charge with date

References:

Books:

1. Elmasri and Navathe, “Fundamentals of Database Systems”, 6th Edition, Pearson Education
2. Korth, Slberchatz,Sudarshan, :”Database System Concepts”, 6th Edition, McGraw – Hill.