Experiment No : 06

Title: To create nested queries and view for the given Database.

Batch: **Roll No.:** **Experiment No: 06**

Aim: To create nested queries and view for the given database.

Resources needed: PostgreSQL PgAdmin3

Theory:

Nested subqueries:

in clause:

The in connective tests for the set membership, where the set is a collection of values produced by a select clause.

For example to select details of the books written by r.p.jain and d.perry use

select book_id, book_name,price from book where author in(„r.p.jain‟, „d. perry‟,‟godse‟);

not in:

This connective tests for absence of the set membership.

For example to select details of the books written by authors other than r.p.jain and d.perry use

select book_id, book_name,price from book where author not in(„r.p.jain‟, „d. perry‟,‟godse‟);

all:

this keyword is basically used in set comparison query.

It is used in association with relational operators.

“> all” corresponds to the phrase „greater than all‟.

For example to display details of the book that have price greater than all the books published in year 2000 use.

Select book_id, book_name, price from book where price >all (select price from book where pub_year=‟2000‟);

any or some:

These keywords are used with relational operators in where clause of set comparison query.

“=some” is identical to in and “<>some” is identical to not in.

“>any “ is nothing but „greater than at least one‟.

exists and not exists:

exists is the test for non empty set. It is represented by an expression of the form ‘exists (select ……. From …….) ‘. Such expression evaluates to true only if the result evaluvating the subquery represented by the (select ……. From ……) is non empty.

for example to select names of the books for which order is placed use

select book_name from book where exists(select * from order where book_id=order.book_id);

Views:

Views are virtual tables created from already existing tables by selecting certain columns or certain rows. A view can be created from one or many tables. View allows to,

- Restrict access to the data such that a user can only see limited data instead of complete table.
- Summarize data from various tables which can be used to generate reports.

In PostgrSQL, Views are created using the CREATE VIEW statement given bellow.

CREATE [TEMP | TEMPORARY] VIEW view_name AS SELECT column1, column2.....
FROM table_name WHERE [condition];
For example,
Consider COMPANY table having following records:

```
id | name | age | address | salary
 ----+-------+-----+------------+--------
1 | Paul | 32 | California | 20000
2 | Allen | 25 | Texas | 15000
3 | Teddy | 23 | Norway | 20000
4 | Mark | 25 | Rich-Mond | 65000
5 | David | 27 | Texas | 85000
6 | Kim | 22 | South-Hall | 45000
7 | James | 24 | Houston | 10000
```

Following statement creates a view from COMPANY table.

CREATE VIEW COMPANY_VIEW AS SELECT ID, NAME, AGE FROM COMPANY;
Now, query can be written on COMPANY_VIEW in similar way as that of an actual table, as shown below,
SELECT * FROM COMPANY_VIEW;
This would produce the following result:
View can be dropped using “DROP VIEW” statement.

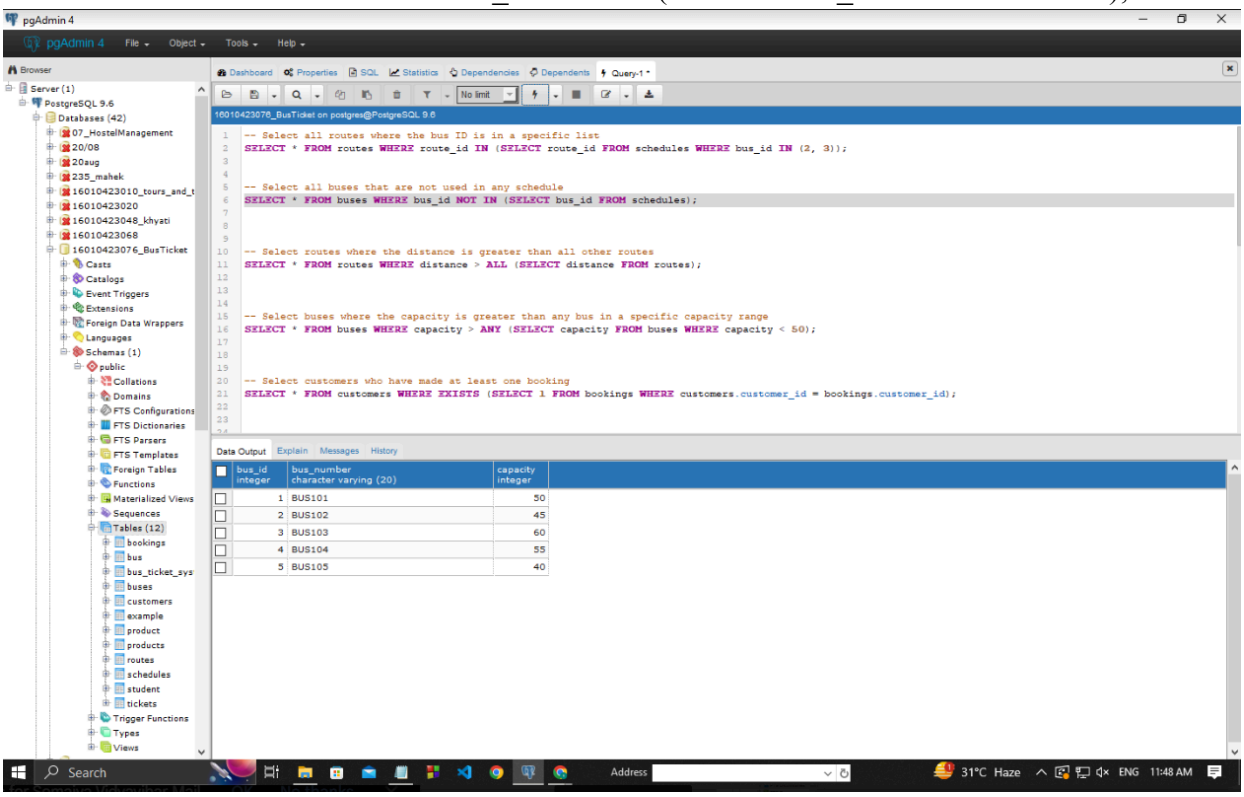
Procedure / Approach /Algorithm / Activity Diagram:

1. Refer different syntax given in theory section and formulate queries consisting of nested sub queries, in , not in, as, group by, having etc clauses and different set operations for your database.
2. Create views from existing tables
   Execute SELECT,UPDATE,INSERT statements on views and original table.

Results: (Program printout with output / Document printout as per the format)

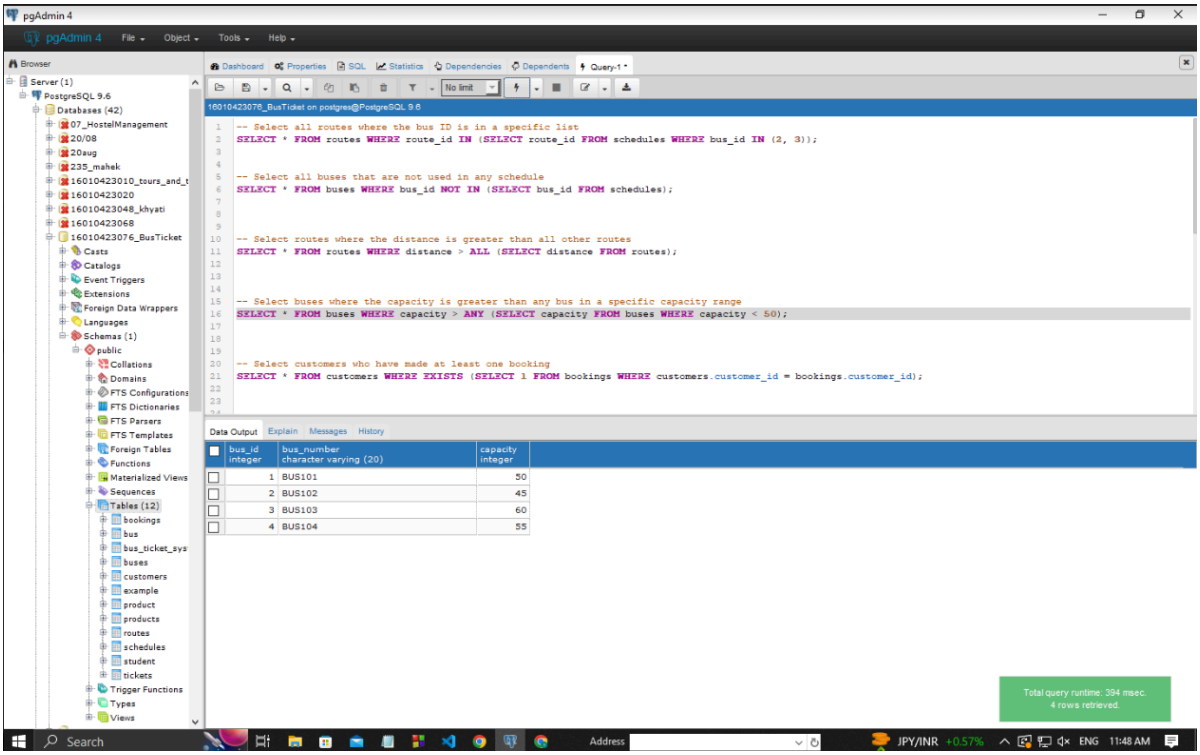
-- in
-- Select routes where the origin city is in a specific set of cities
SELECT * FROM routes WHERE origin IN ('Delhi', 'Indore');

-- not in
-- Select all buses that are not used in any schedule
SELECT * FROM buses WHERE bus_id NOT IN (SELECT bus_id FROM schedules);



-- all
-- Routes where the distance is greater than all other routes
SELECT * FROM routes WHERE distance > ALL (SELECT distance FROM routes);

-- any
-- Buses where the capacity is greater than any bus in a specific capacity range
SELECT * FROM buses WHERE capacity > ANY (SELECT capacity FROM buses WHERE capacity < 50);

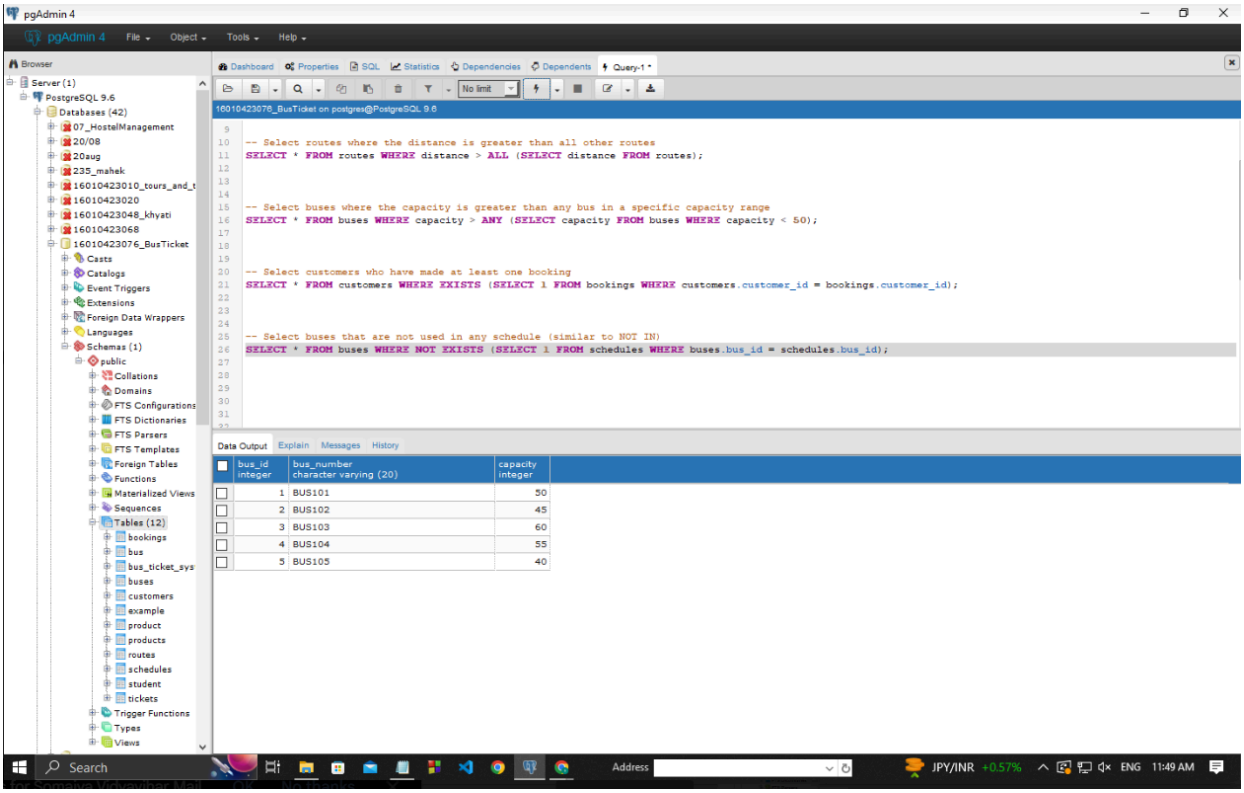


-- exists

-- Customers who have made at least one booking
SELECT * FROM customers WHERE EXISTS (SELECT 1 FROM bookings WHERE customers.customer_id = bookings.customer_id);

-- not exist

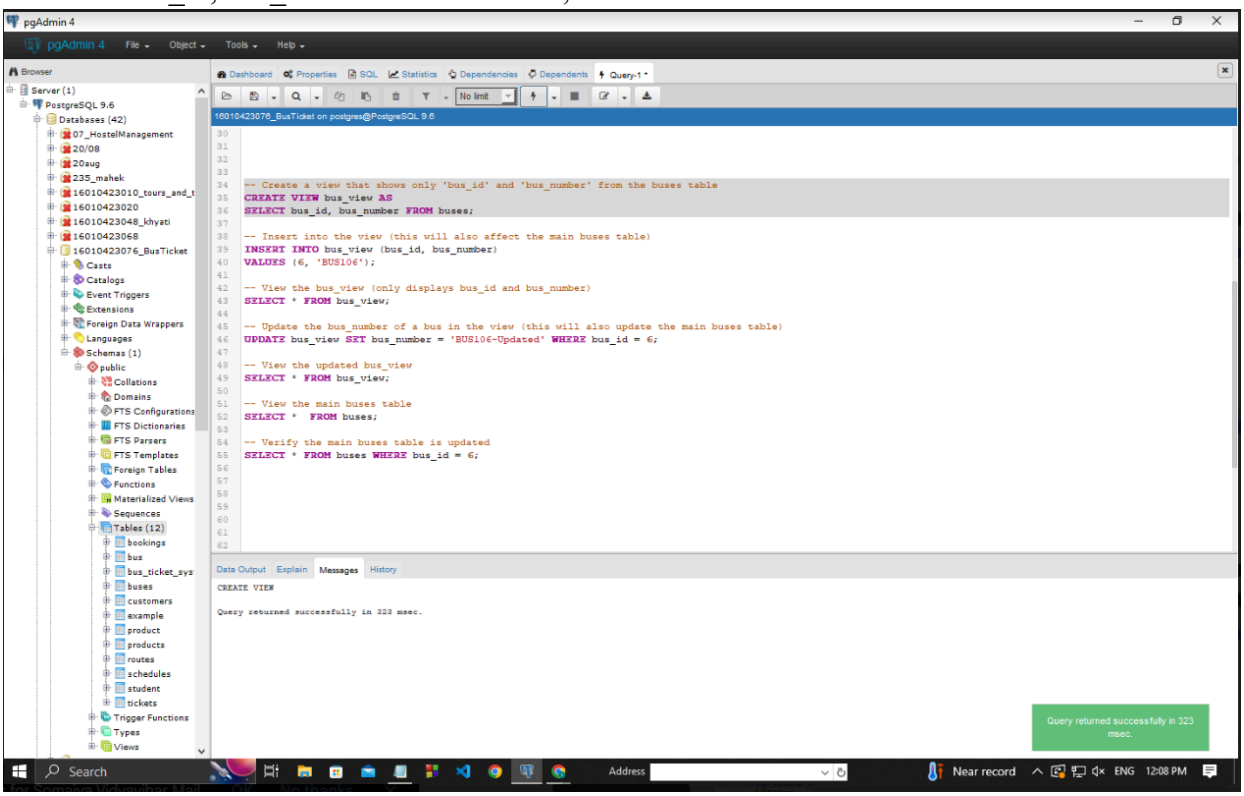
-- Buses that are not used in any schedule (similar to NOT IN)
SELECT * FROM buses WHERE NOT EXISTS (SELECT 1 FROM schedules WHERE buses.bus_id = schedules.bus_id);



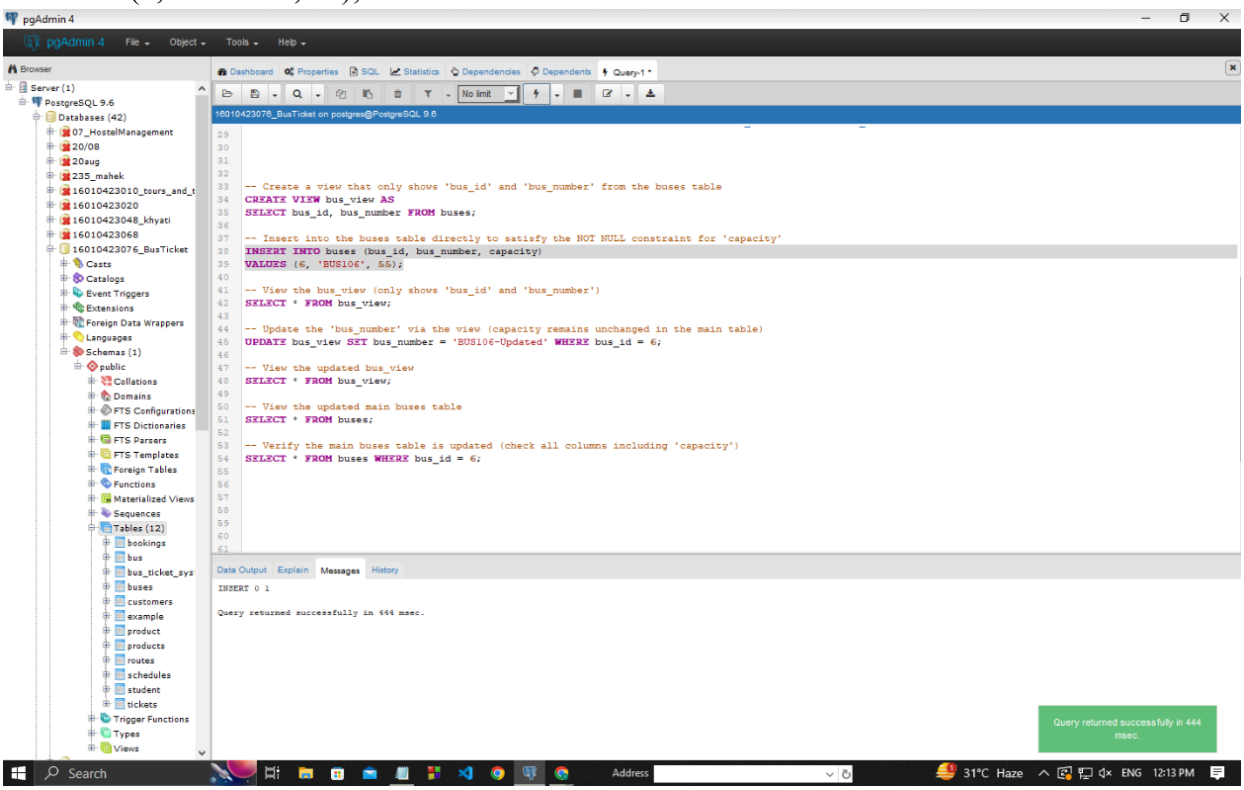
--View

-- Create a view that only shows 'bus_id' and 'bus_number' from the buses table

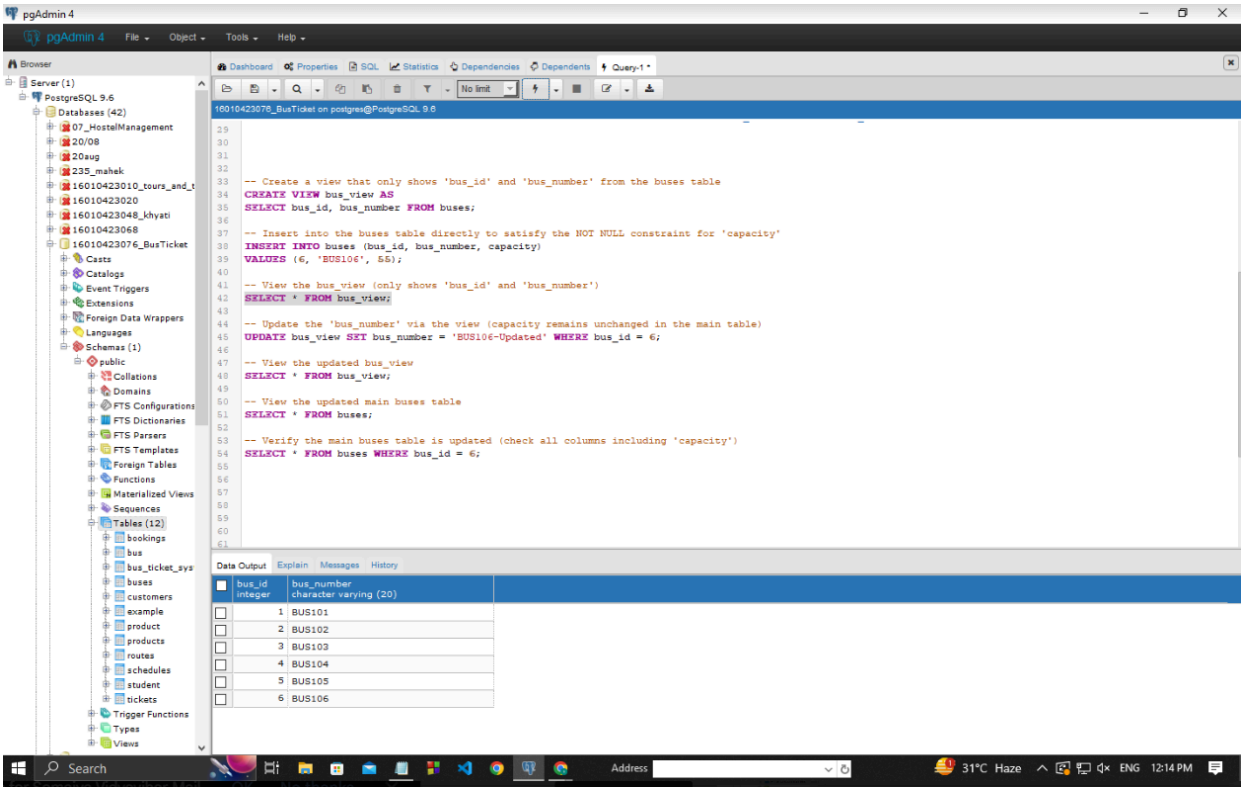
```
CREATE VIEW bus_view AS
SELECT bus_id, bus_number FROM buses;
```



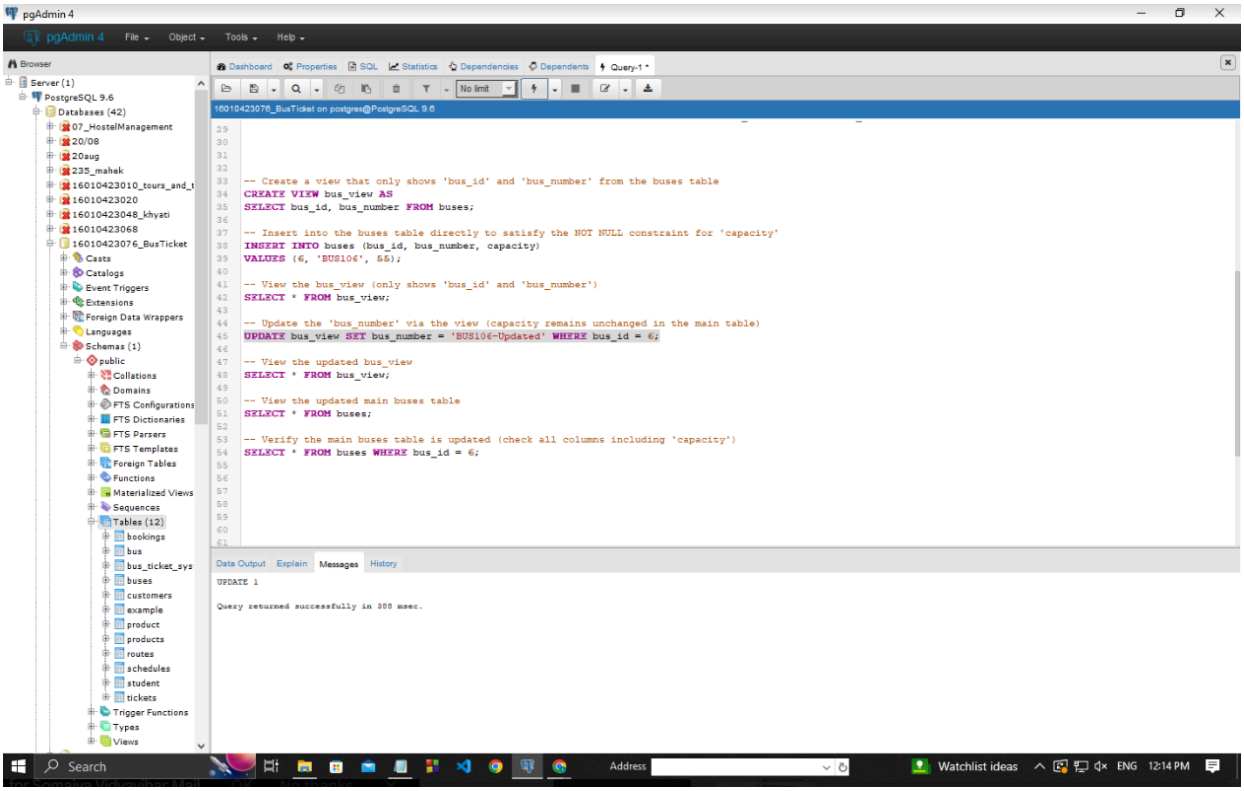
-- Insert into the buses table directly to satisfy the NOT NULL constraint for 'capacity'
INSERT INTO buses (bus_id, bus_number, capacity)
VALUES (6, 'BUS106', 55);



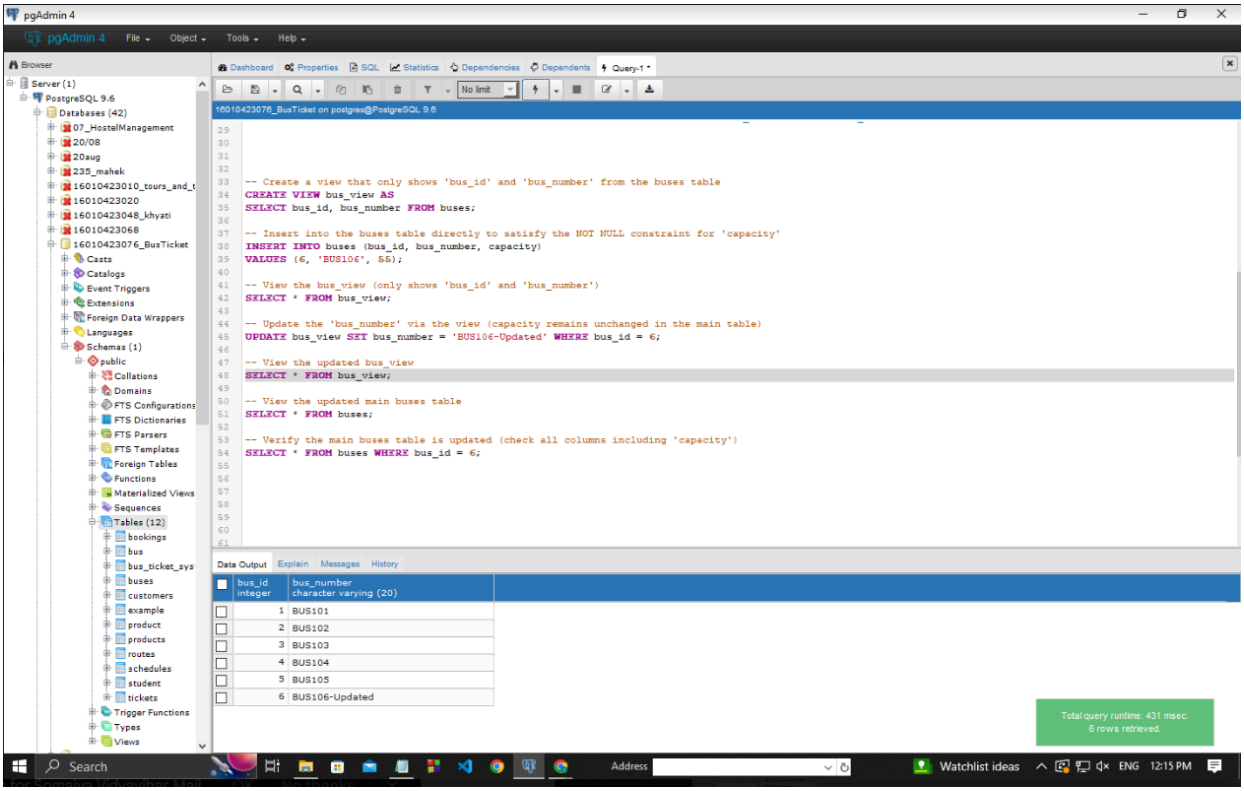
-- View the bus_view (only shows 'bus_id' and 'bus_number')
SELECT * FROM bus_view;



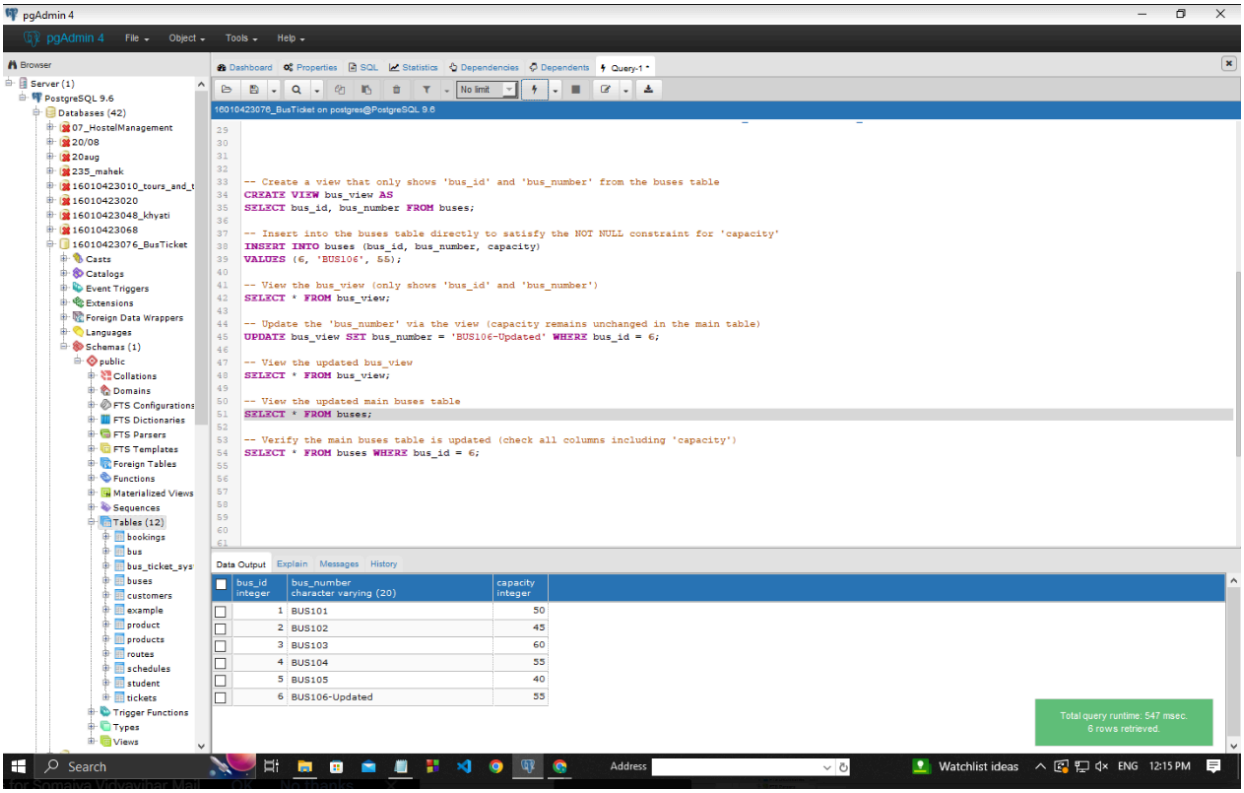
-- Update the 'bus_number' via the view (capacity remains unchanged in the main table)
UPDATE bus_view SET bus_number = 'BUS106-Updated' WHERE bus_id = 6;



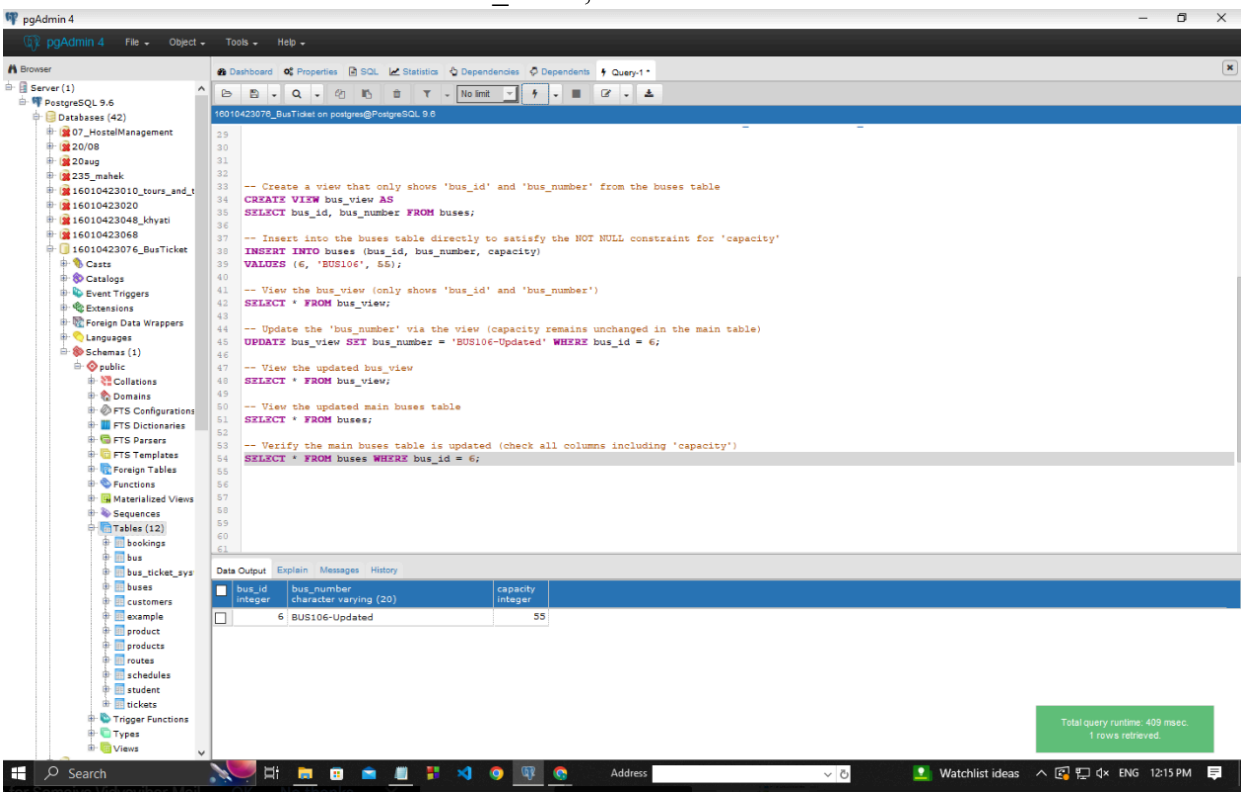
-- View the updated bus_view
SELECT * FROM bus_view;



-- View the updated main buses table
SELECT * FROM buses;



-- Verify the main buses table is updated (check all columns including 'capacity')
SELECT * FROM buses WHERE bus_id = 6;



Questions:

1. Explain what are the disadvantages using view on update function.
   Performance Issues: Updating data through views can lead to poor performance, especially when the view is built from multiple base tables or includes complex joins and calculations. It might slow down the query execution.
   Limited Updates: Some views, especially those based on joins or containing aggregate functions, might not allow updates to the underlying data or may restrict which columns can be updated. In some cases, views may be read-only, preventing any updates.
   Data Integrity Concerns: If a view is complex (e.g., involves calculations or filters), updating through the view may not always maintain data integrity or reflect the expected changes in the underlying base tables.
   Complexity: Maintaining views for update operations can complicate database design, especially when changes to the underlying tables require corresponding updates in the view logic.

2 Can we use where clause with group by clause? Justify your answer
   Yes, the `WHERE` clause can be used with the `GROUP BY` clause. The `WHERE` clause is used to filter rows before they are grouped. In this way, it helps limit the data that is passed to the `GROUP BY` clause for grouping.
   Example :
   SELECT department, COUNT(*)
   FROM employees
   WHERE salary > 50000
   GROUP BY department;

3 Can we use having and group by clause without Aggregate functions? Justify your answer
   Yes, the `HAVING` and `GROUP BY` clause can be used without aggregate functions. However, the `HAVING` clause is generally meant to filter the results of aggregate functions after grouping. If no aggregate function is used, `HAVING` simply acts as a second `WHERE` clause but for grouped data.
   Example :
   SELECT department
   FROM employees
   GROUP BY department
   HAVING department LIKE 'Sales%';

Outcomes:

CO2 Apply data models to real world scenarios.

Conclusion: (Conclusion to be based on the objectives and outcomes achieved)

I learned how to use nested queries like IN, NOT IN, ALL, ANY, EXISTS and NOT EXISTS to filter data effectively. I also explored creating and updating views, realizing that while views simplify queries, updating through them can impact performance and limit what can be changed. Additionally, I understood how WHERE works with GROUP BY to filter data before grouping and how HAVING can be used without aggregate functions. Overall, I gained a stronger grasp of advanced SQL querying and view management.

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

Signature of faculty in-charge with date

References:

Books/ Journals/ Websites:

1. Korth, Slberchatz,Sudarshan, :"Database System Concepts", 6th Edition, McGraw – Hill
2. Elmasri and Navathe, " Fundamentals of Database Systems", 5thEdition, PEARSON Education.