1. Retrieve Locations of specific features (10 marks)

CREATE TABLE cafes (

id SERIAL PRIMARY KEY,

name VARCHAR(100),

location GEOMETRY(Point, 4326) -- Example assuming WGS 84 coordinate system

);

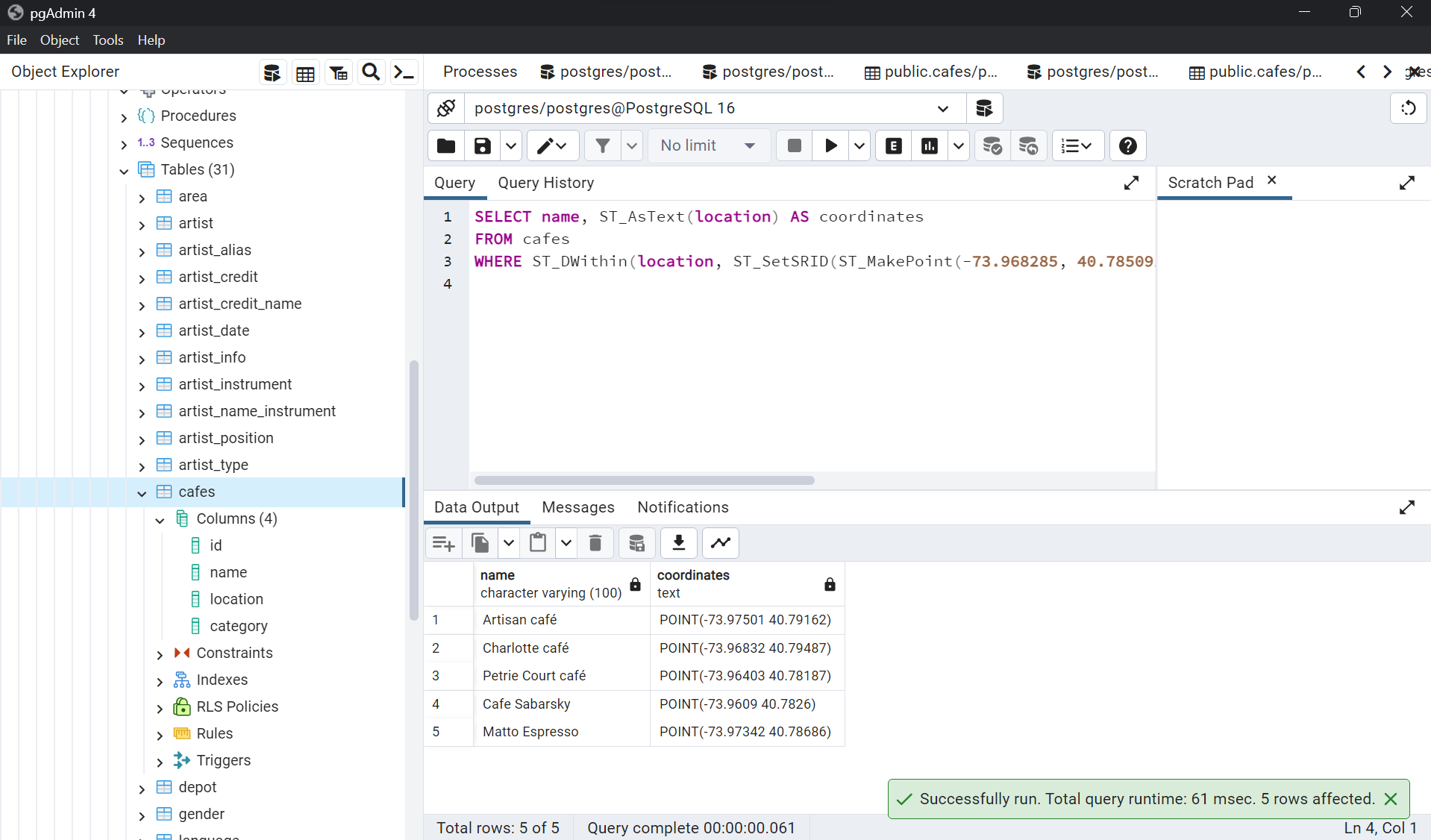
INSERT INTO cafes (name, location)

VALUES ('Black Press Cofee’, ST\_SetSRID(ST\_MakePoint(-73.97896, 40.77953), 4326));

VALUES ( , ST\_SetSRID(ST\_MakePoint(-73.98394, 40.78447), 4326));

SELECT name, ST\_AsText(location) AS coordinates

FROM cafes

WHERE ST\_DWithin(location, ST\_SetSRID(ST\_MakePoint(-73.968285, 40.785091), 4326), 0.01); -- Example radius in degrees (approximately 111 km) 

2. Calculate Distance between points (10 marks)

SELECT

name,

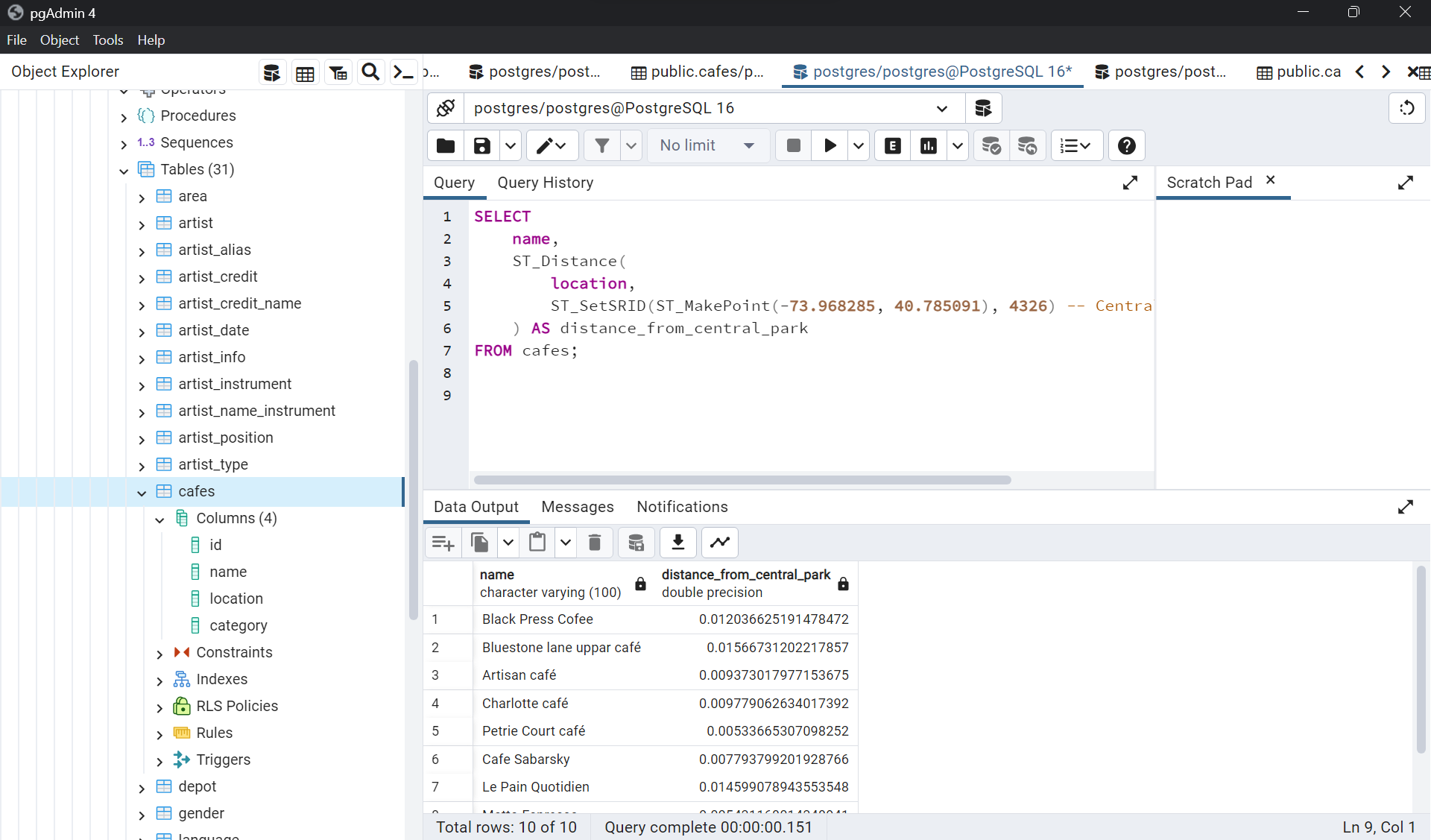
ST\_Distance(

location,

ST\_SetSRID(ST\_MakePoint(-73.968285, 40.785091), 4326) -- Central Park coordinates

) AS distance\_from\_central\_park

FROM cafes;



3.Calculate Areas of Interest (specific to each group) (10 marks)

SELECT

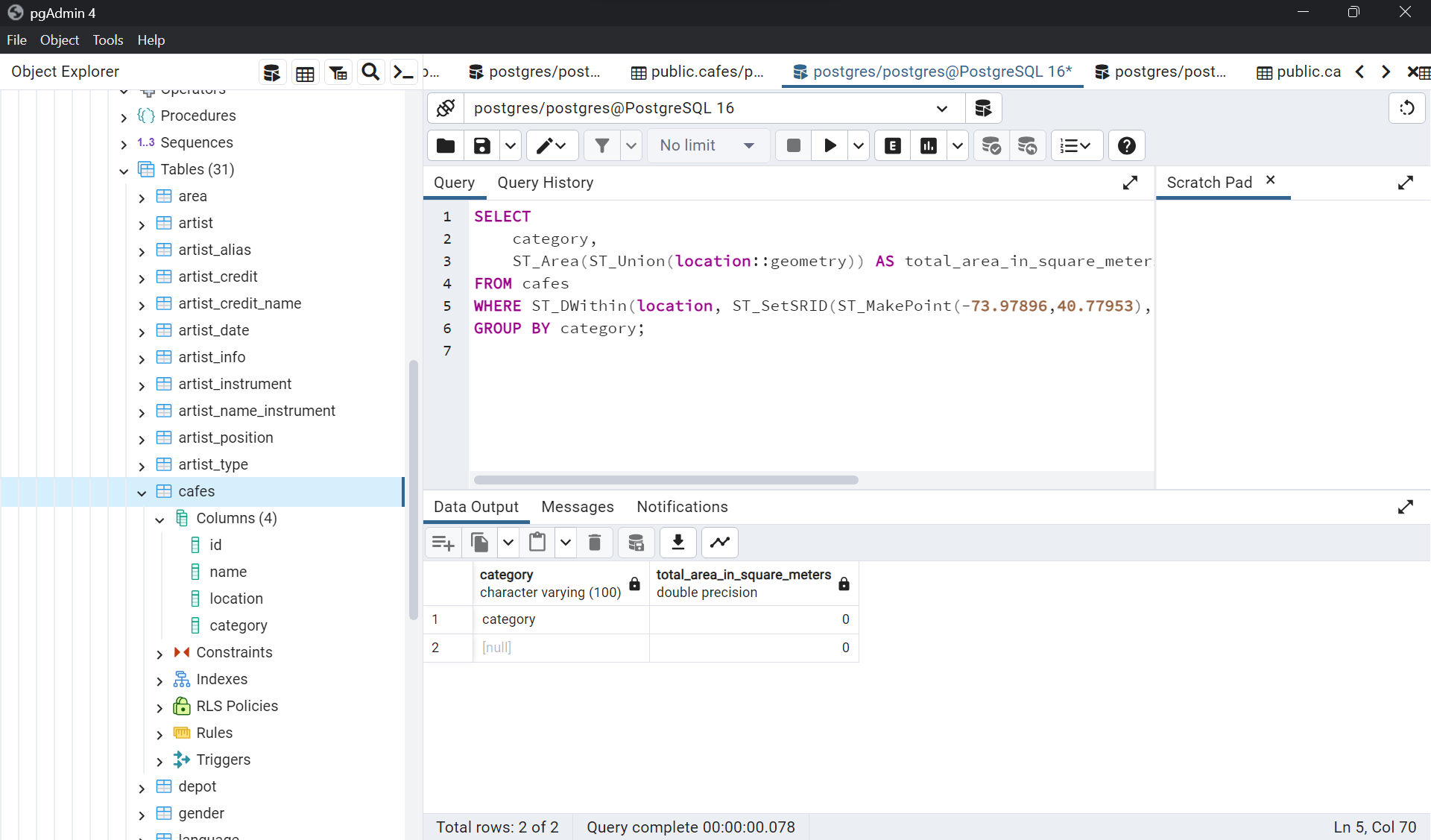
category,

ST\_Area(ST\_Union(location::geometry)) AS total\_area\_in\_square\_meters

FROM cafes

WHERE ST\_DWithin(location, ST\_SetSRID(ST\_MakePoint(-73.97896,40.77953), 4326), 0.01) -- Filter cafes within a certain radius

GROUP BY category;

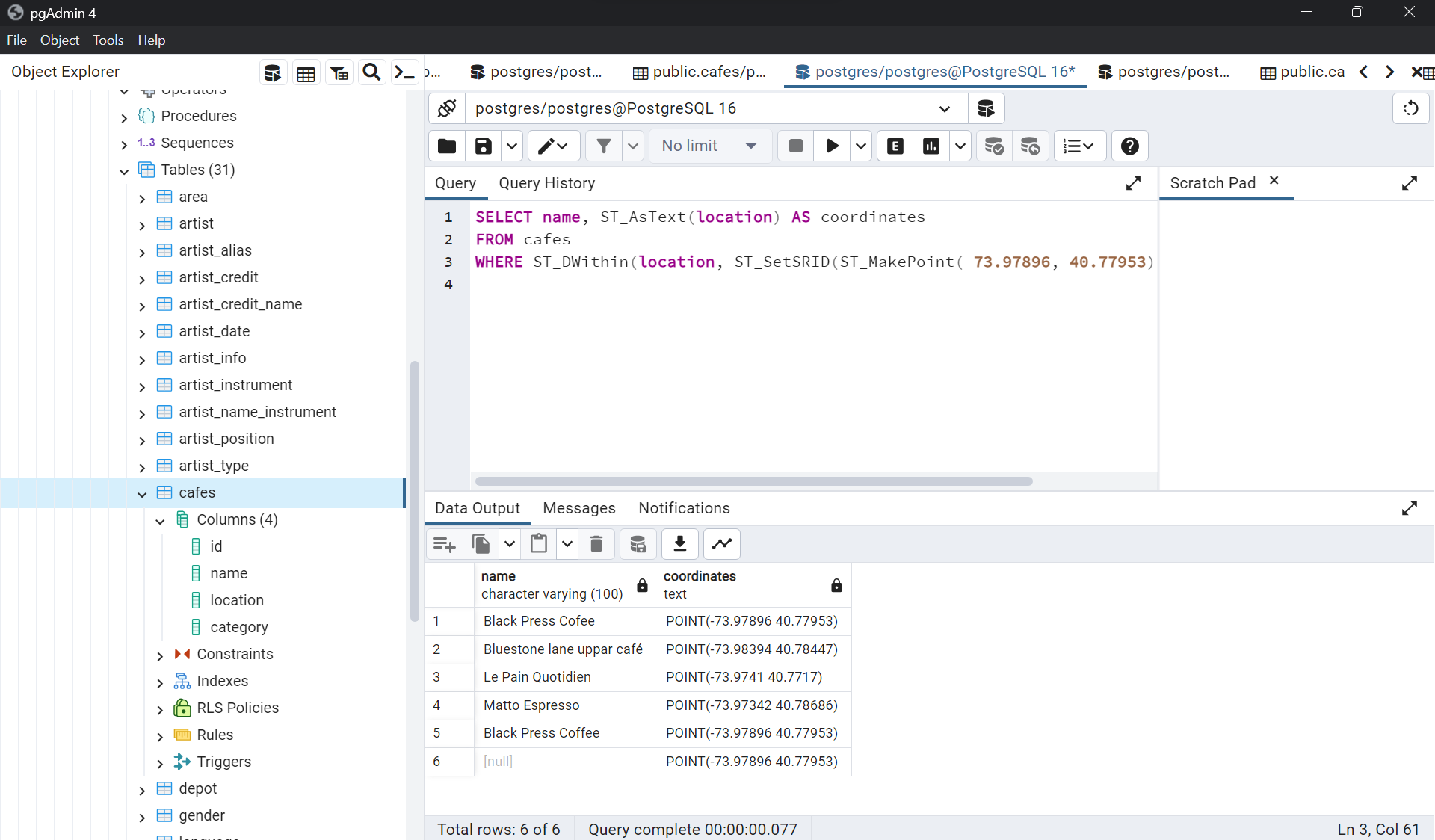


4. Analyze the queries (10 marks)

SELECT name, ST\_AsText(location) AS coordinates

FROM cafes

WHERE ST\_DWithin(location, ST\_SetSRID(ST\_MakePoint(-73.97896, 40.77953), 4326), 0.01);



5. Sorting and Limit Executions (10 marks)

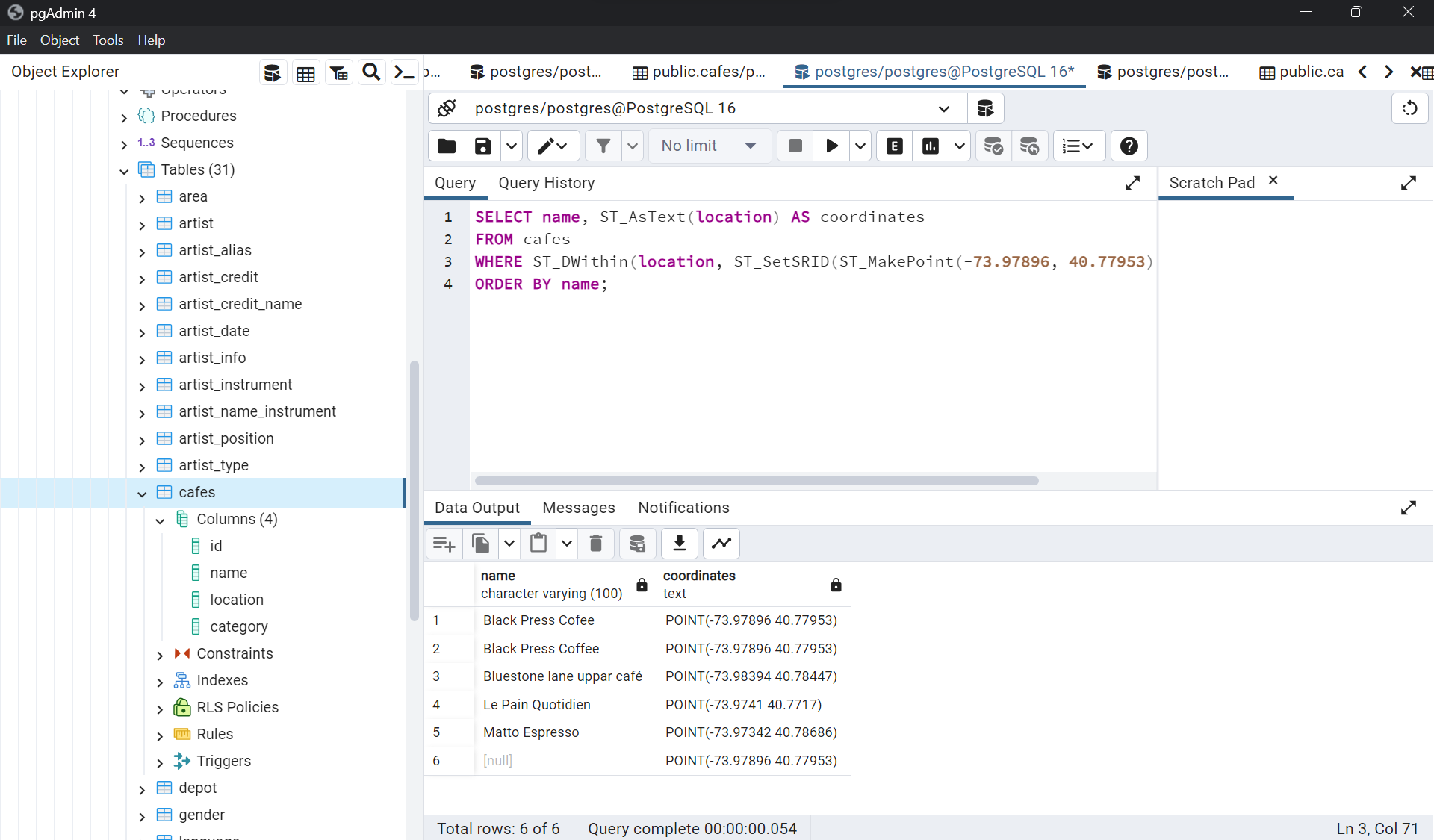
Query 1: Retrieve Locations of Specific Features with Sorting

SELECT name, ST\_AsText(location) AS coordinates

FROM cafes

WHERE ST\_DWithin(location, ST\_SetSRID(ST\_MakePoint(-73.97896, 40.77953), 4326), 0.01)

ORDER BY name;



6. Optimize the queries to speed up execution time (10 marks)

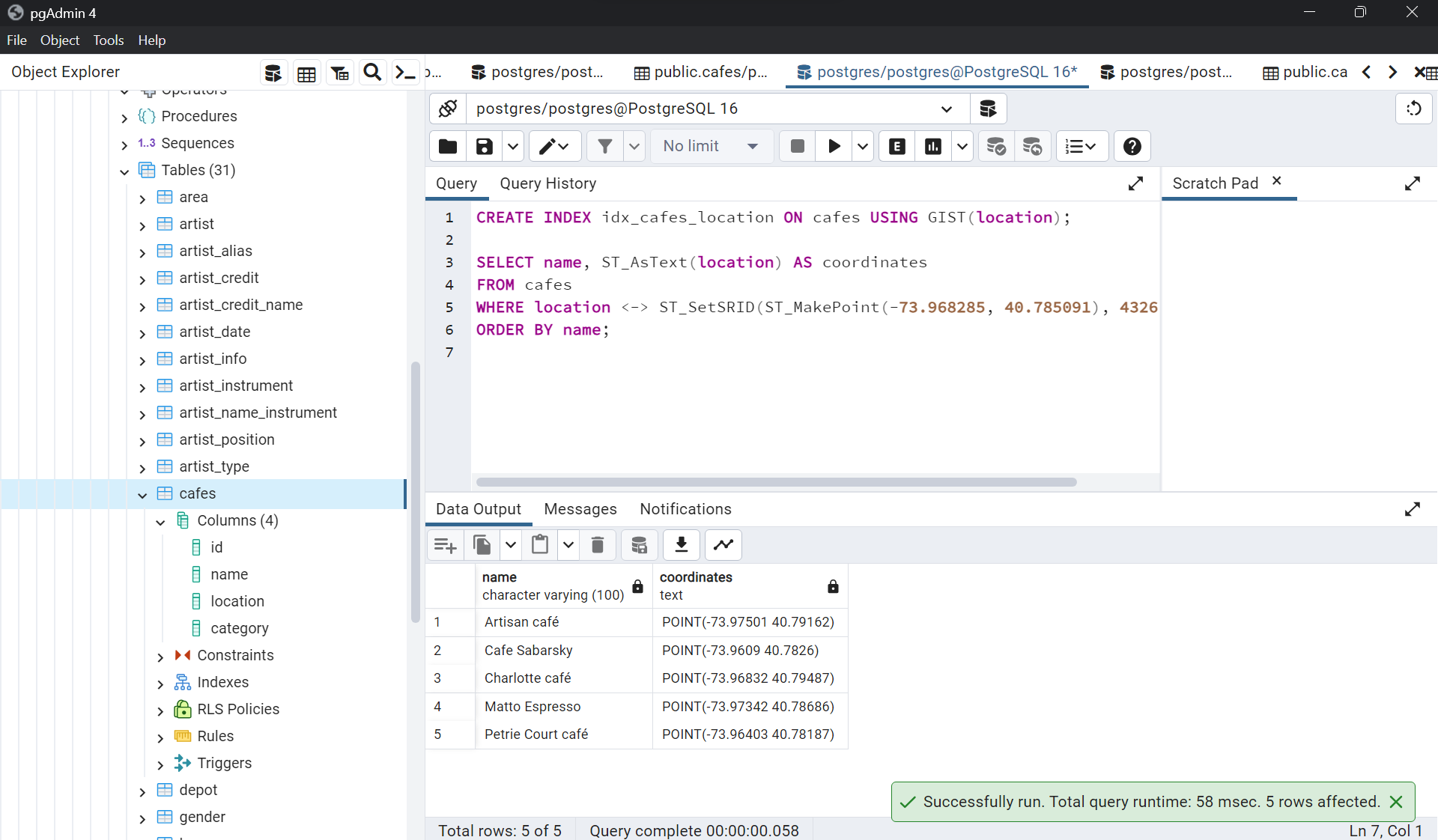
CREATE INDEX idx\_cafes\_location ON cafes USING GIST(location);

SELECT name, ST\_AsText(location) AS coordinates

FROM cafes

WHERE location <-> ST\_SetSRID(ST\_MakePoint(-73.968285, 40.785091), 4326) <= 0.01

ORDER BY name;



7.N-Optimization of queries (5 marks)

N-optimization of queries refers to a concept where you optimize queries based on the variable N, which represents the volume or quantity of data involved. This approach ensures that query performance scales efficiently as the dataset grows.

Here are strategies for N-optimization in GIS queries:

1. Spatial Indexing and Query Optimization

For retrieving locations of specific features:

N-Optimization:

Ensure that spatial indexing is appropriately set for the entire dataset to handle an increasing number of spatial objects efficiently.

Consider periodically re-indexing or optimizing indexes based on data updates or changes.

Use query tuning techniques based on profiling query performance as the dataset size changes.

2. Efficient Distance Calculations

For calculating distance between points:

N-Optimization:

Employ spatial indexing for locations to speed up distance calculations, which becomes crucial with larger datasets.

Optimize the distance calculations by limiting operations through bounding box checks or spatial partitioning techniques for better scalability.

3. Grouping and Aggregation

For calculating areas of interest:

N-Optimization:

Optimize grouping and aggregation operations by periodically analyzing and adjusting the grouping criteria to ensure efficiency as the dataset grows.

Consider spatial partitioning or clustering approaches for better performance in handling larger data volumes during aggregation.

4. Asynchronous Processing and Parallelism

For large-scale GIS analysis:

N-Optimization:

Implement asynchronous processing or parallelism techniques for complex or computationally intensive GIS tasks, allowing distributed processing across multiple nodes to handle increased data volume effectively.

Utilize technologies or frameworks that support parallel processing of spatial data for improved performance as the dataset size increases.

5. Performance Monitoring and Optimization Iteration

N-Optimization:

Regularly monitor query performance metrics as the dataset grows and adjust optimization strategies accordingly.

Conduct periodic reviews and iterations of query optimization techniques to adapt to changing data volumes and maintain efficient query execution.

By adopting N-optimization strategies, continuously monitoring performance, and adjusting optimization techniques based on the dataset size, GIS queries can be optimized to scale efficiently with increasing volumes of spatial data.

The key is to implement techniques that maintain query performance even as the dataset grows, ensuring efficient and effective GIS analysis.

8. Presentation and Posting to GitHub (5 marks)

[post your github link here as solution]

**9. Code functionality, documentation and proper output provided (5marks)**

* Code Functionality will be explained in the video. And for every question we provided the answer with screenshot that would be our explanation.