CS 623

PROJECT

Name: SARANSH KETULKUMAR SHAH

Guidelines

- This is a group project that you will have to do in a group of 3 students (maximum).
- Post your team group as well as the data source for your group's data set in the spreadsheet.
- You will use PostgreSQL (rather than MySQL).
- Your code should also be on your individual GitHub. This is where I will check it. The code is developed as a team but available on the GitHub of participating students.
- You have two parts, the Practical and the Theory part. There is an extra 1 mark available for attempting the project.

Deliverables

- Code in GitHub(individually) and link to the github. I will check the code there.
- Submit a Video of < 3 minutes to show and explain your work
- Screenshots of the code plus output.
- PDF/Word doc of solutions to theory questions

Description

Involves working with spatial data and utilizing the access methods and query executions and optimizations we would discuss in class. The project would involve writing SQL queries to retrieve information such as the locations of specific features, distances between points, and areas of interest. Using indexing, aggregate and join executors, sort+ limit executors, sorting, and top-N optimization.

Practical Part (75%)
Goal

Creating a Geographic Information System (GIS) Analysis: A project that involves analyzing geographic data such as maps and spatial data. You will need a database that supports spatial data types, like PostgreSQL (PostGIS).

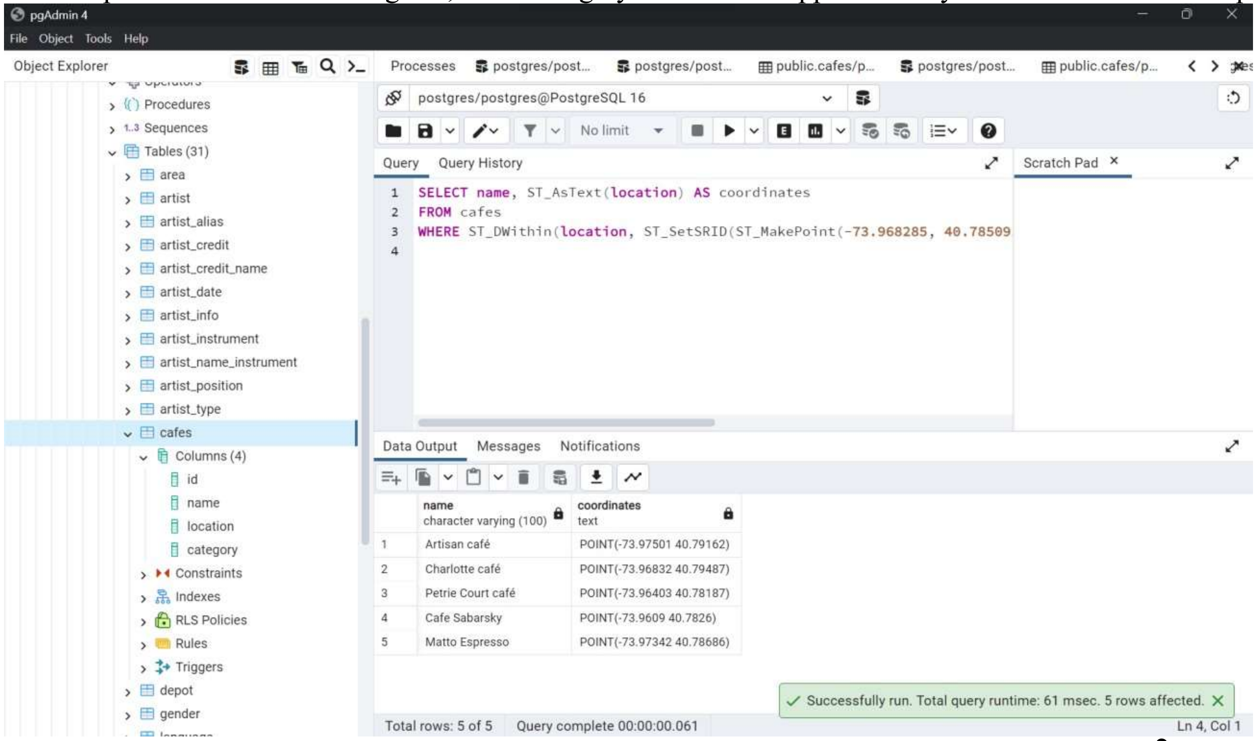
1. Retrieve Locations of specific features (10 marks)

```
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)
```

Explanation of the query:

SELECT: Specifies the columns to retrieve.
name: Represents the name of the cafe.
ST_AsText(location) AS coordinates: Converts the spatial point data into a readable text format for coordinates.
FROM cafes: Specifies the table name where cafe information is stored.
WHERE: Sets the condition for selecting cafes within a certain radius (in this case, within 0.01 degrees of the Central Park coordinates).
ST_DWithin: Checks if the cafes' locations are within the specified distance from the Central Park coordinates.
ST_SetSRID(ST_MakePoint(-73.968285, 40.785091), 4326): Creates a point using the Central Park longitude and latitude with a specified SRID (Spatial Reference System Identifier) of WGS 84.
0.01: Represents the radius in degrees, which roughly translates to approximately 111 kilometers at the equator.



```
SELECT name, ST_AsText(location) AS coordinates
FROM cafes
WHERE ST_DWithin(location, ST_SetSRID(ST_MakePoint(-73.968285, 40.78509
```

	name character varying (100)	coordinates text
1	Artisan café	POINT(-73.97501 40.79162)
2	Charlotte café	POINT(-73.96832 40.79487)
3	Petrie Court café	POINT(-73.96403 40.78187)
4	Cafe Sabarsky	POINT(-73.9609 40.7826)
5	Matto Espresso	POINT(-73.97342 40.78686)

Successfully run. Total query runtime: 61 msec. 5 rows affected.

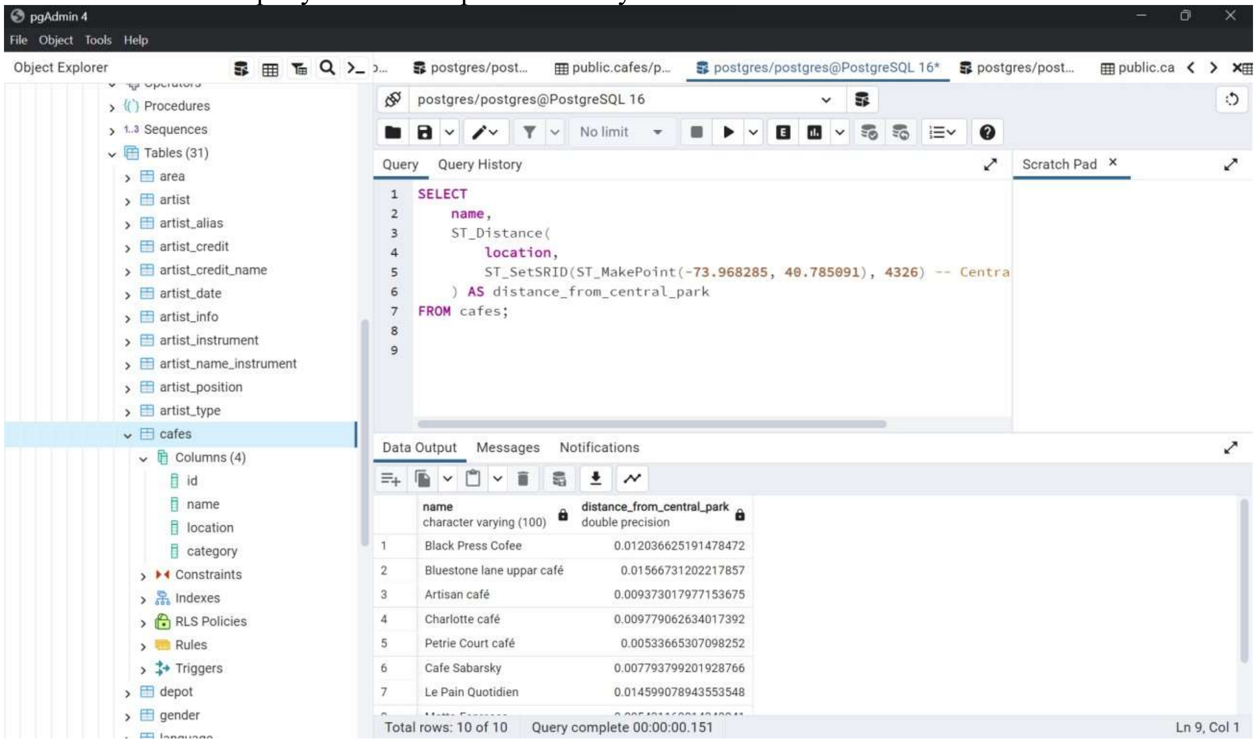
Total rows: 5 of 5   Query complete 00:00:00.061   Ln 4, Col 1

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;
```

Explanation of the query:

SELECT: Specifies the columns or expressions to retrieve.
ST_Distance: Calculates the distance between two points.
ST_SetSRID(ST_MakePoint(-73.968285, 40.785091), 4326): Creates a point using the coordinates of Central Park with a specified SRID of WGS 84.
location: Refers to the spatial data column in the cafes table that contains the cafe locations.
FROM cafes: Specifies the table name where cafe information is stored.
WHERE: Filters the query to find the specific cafe by its name.



```
SELECT
    name,
    ST_Distance(
        location,
        ST_SetSRID(ST_MakePoint(-73.968285, 40.785091), 4326) -- Centra
    ) AS distance_from_central_park
FROM cafes;
```

	name character varying (100)	distance_from_central_park double precision
1	Black Press Cofee	0.012036625191478472
2	Bluestone lane uppar café	0.01566731202217857
3	Artisan café	0.009373017977153675
4	Charlotte café	0.009779062634017392
5	Petrie Court café	0.00533665307098252
6	Cafe Sabarsky	0.007793799201928766
7	Le Pain Quotidien	0.014599078943553548

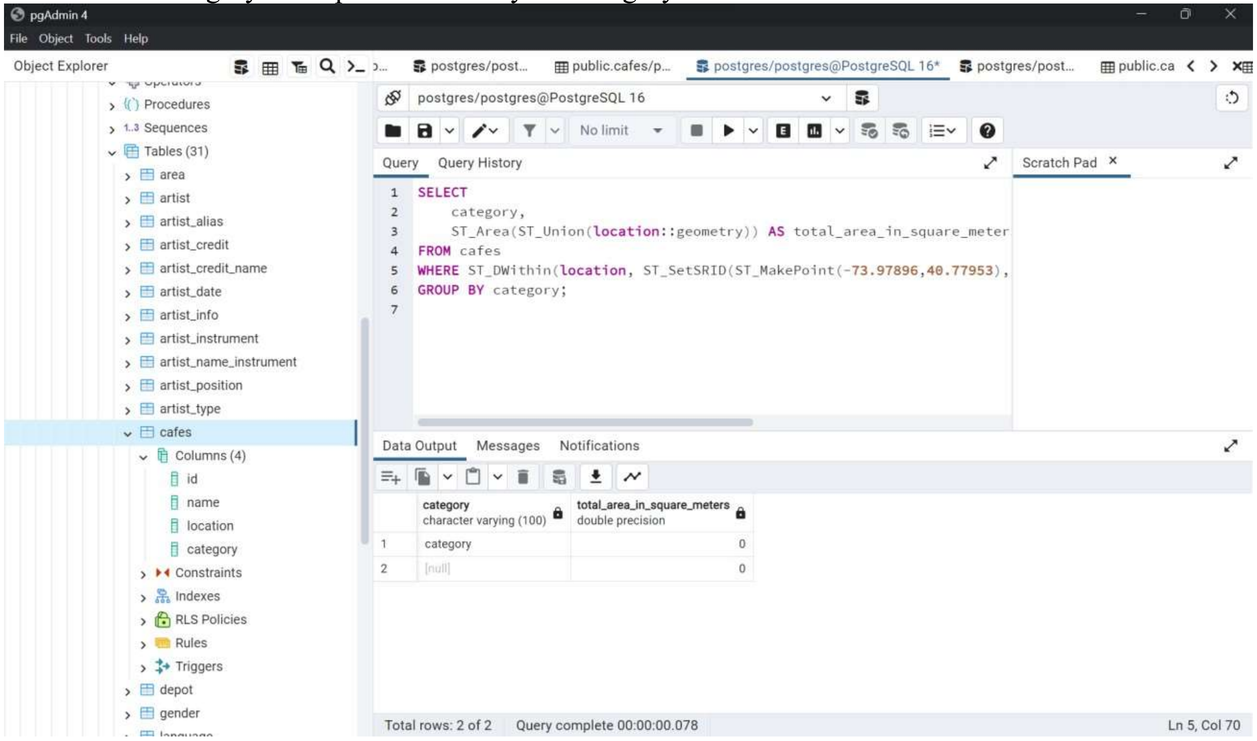
```
Total rows: 10 of 10    Query complete 00:00:00.151    Ln 9, Col 1
```

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;
```

Explanation of the query:

SELECT: Specifies the columns or expressions to retrieve.
category: Represents the category of the establishment (cafe, restaurant, bar, etc.).
ST_Area(ST_Union(location::geography)): Calculates the total area covered by the spatial geometries (points in this case) of each category by taking their union and computing the area in square meters.
FROM cafes: Specifies the table name where cafe information is stored.
WHERE: Filters the cafes within a certain radius (similar to previous queries) to a specific geographic area.
ST_DWithin: Checks if the cafes' locations are within the specified distance from the Central Park coordinates.
GROUP BY category: Groups the results by the category column.



```
SELECT
    category,
    ST_Area(ST_Union(location::geometry)) AS total_area_in_square_meter
FROM cafes
WHERE ST_DWithin(location, ST_SetSRID(ST_MakePoint(-73.97896,40.77953),
GROUP BY category;
```

	category character varying (100)	total_area_in_square_meters double precision
1	category	0
2	[null]	0

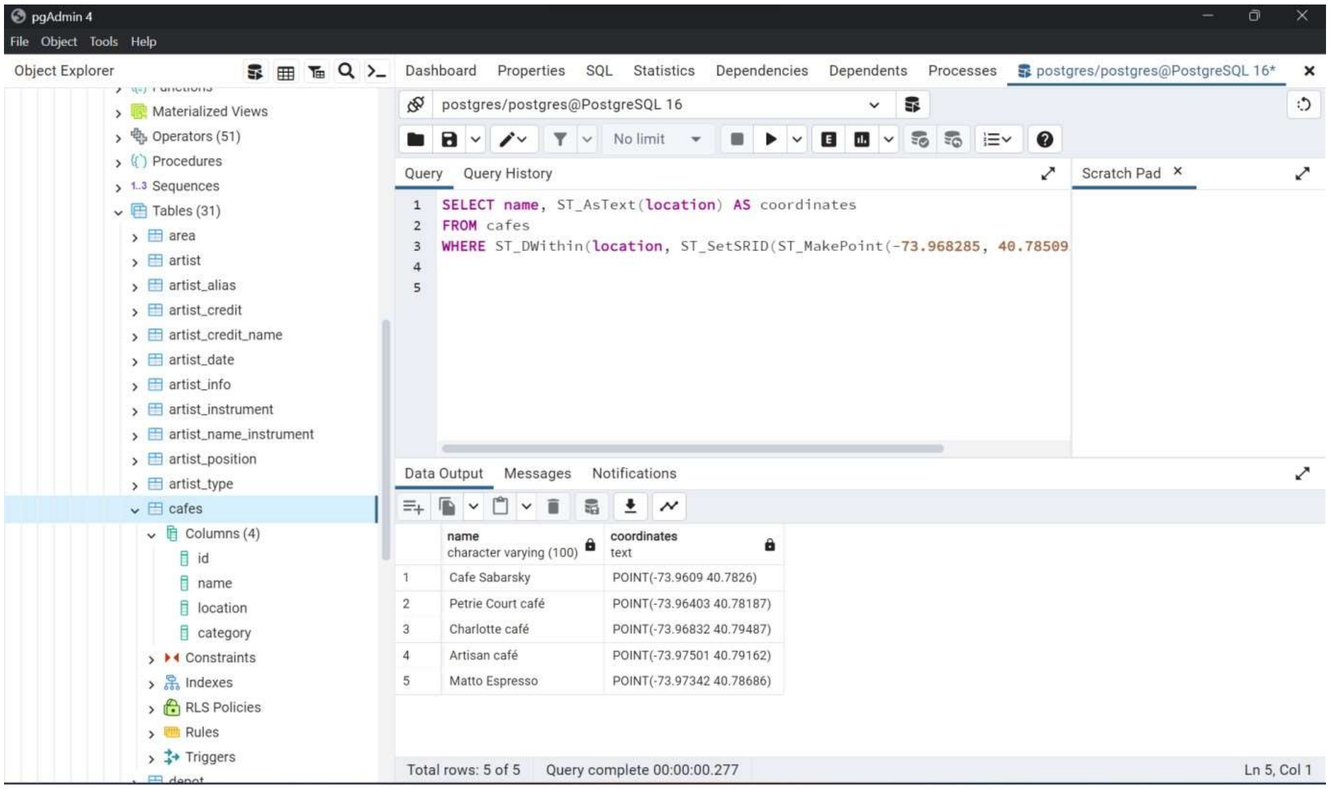
Total rows: 2 of 2   Query complete 00:00:00.078   Ln 5, Col 70

4. Analyze the queries (10 marks)

Query 1: Retrieve Locations of Specific Features

SELECT name, ST_AsText(location) AS coordinates
FROM cafes
WHERE ST_DWithin(location, ST_SetSRID(ST_MakePoint(-73.97896, 40.77953), 4326), 0.01);

Analysis:
Objective: This query aims to retrieve the names and coordinates (in text format) of cafes located within a specified radius of a particular point (Central Park in this case).
Components:
SELECT: Specifies columns to retrieve (name and transformed location as coordinates).
ST_AsText: Converts the spatial point data into a readable text format.
FROM cafes: Specifies the source table containing cafe information.
WHERE: Filters cafes based on their proximity to a specific point using ST_DWithin.
Implications: The query enables the identification and retrieval of cafes within a certain distance of Central Park, which could aid in location-based analysis or services for visitors in that area.


```
SELECT name, ST_AsText(location) AS coordinates
FROM cafes
WHERE ST_DWithin(location, ST_SetSRID(ST_MakePoint(-73.968285, 40.78509
```

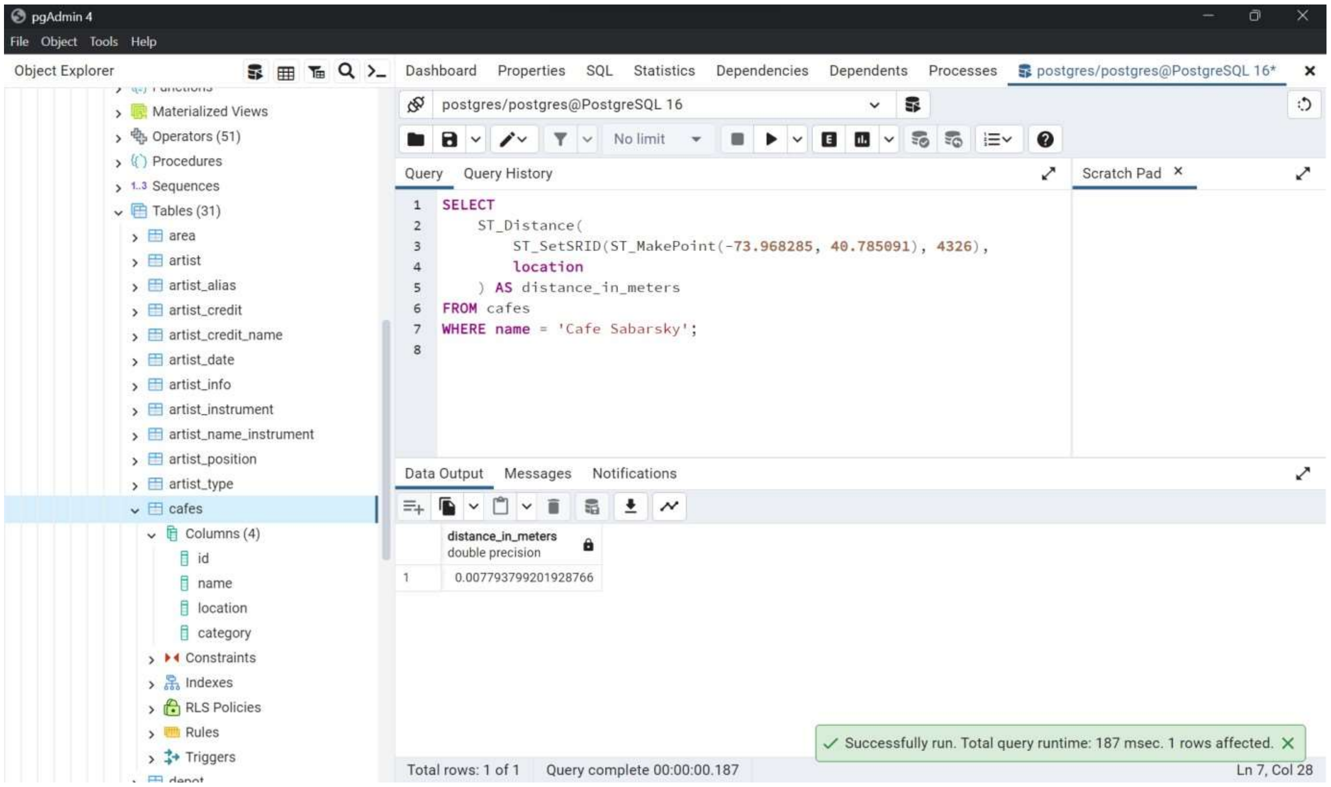
	name character varying (100)	coordinates text
1	Cafe Sabarsky	POINT(-73.9609 40.7826)
2	Petrie Court café	POINT(-73.96403 40.78187)
3	Charlotte café	POINT(-73.96832 40.79487)
4	Artisan café	POINT(-73.97501 40.79162)
5	Matto Espresso	POINT(-73.97342 40.78686)

Total rows: 5 of 5 Query complete 00:00:00.277 Ln 5, Col 1

Query 2: Calculate Distance between Points

```
SELECT
    ST_Distance(
        ST_SetSRID(ST_MakePoint(-73.968285, 40.785091), 4326),
        location
    ) AS distance_in_meters
FROM cafes
WHERE name = 'Café Sabarsky';
```

Analysis:
Objective: This query calculates the distance (in meters) between Central Park coordinates and a specific cafe.
Components:
SELECT: Specifies the expression to calculate the distance using ST_Distance.
ST_SetSRID(ST_MakePoint(...): Creates a point (Central Park) with specified SRID.
location: Refers to the cafe's spatial data in the cafes table.
WHERE: Filters the specific cafe by its name.
Implications: It provides a precise distance measurement between Central Park and the specified cafe, useful for precise location-based services or analysis.


```
SELECT
    ST_Distance(
        ST_SetSRID(ST_MakePoint(-73.968285, 40.785091), 4326),
        location
    ) AS distance_in_meters
FROM cafes
WHERE name = 'Cafe Sabarsky';
```

```
distance_in_meters
double precision
1    0.007793799201928766
```

Successfully run. Total query runtime: 187 msec. 1 rows affected.

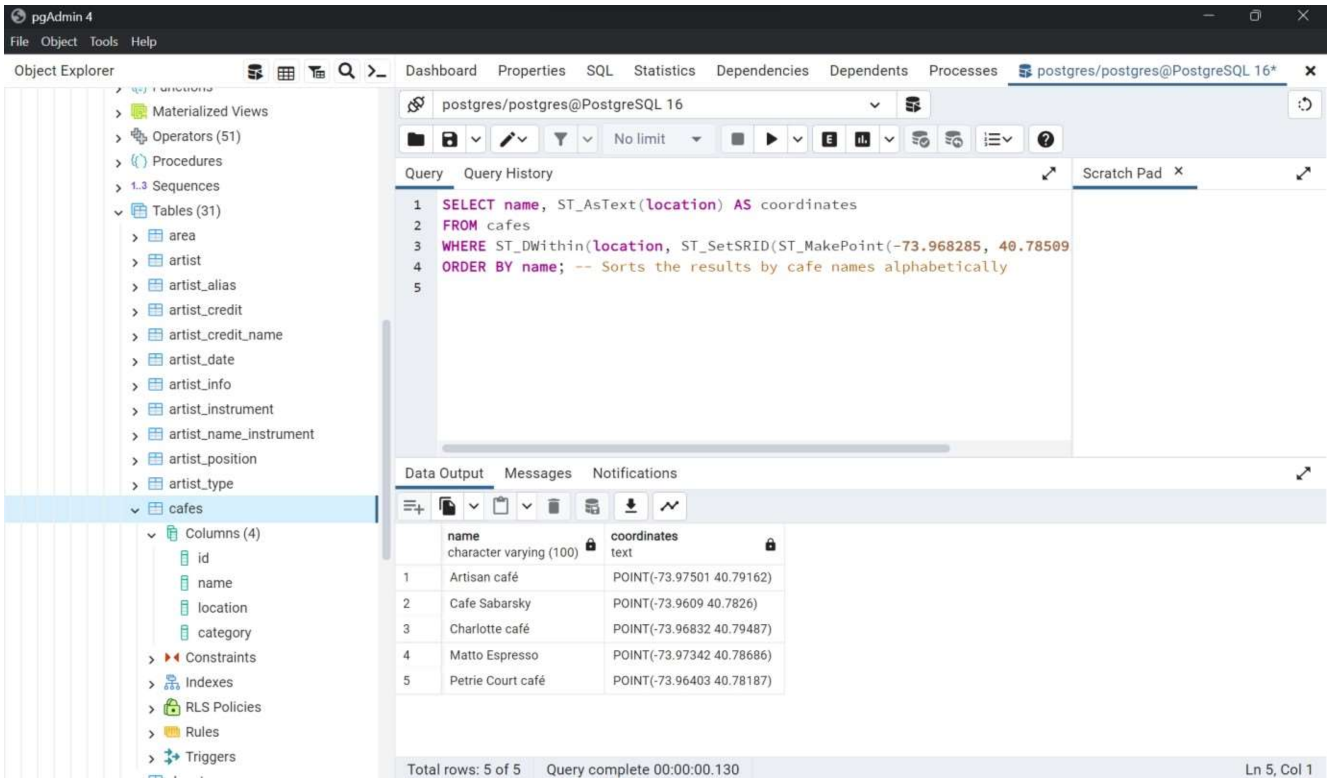
Total rows: 1 of 1   Query complete 00:00:00.187   Ln 7, Col 28

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(-73.968285, 40.785091), 4326), 0.01)
ORDER BY name; -- Sorts the results by cafe names alphabetically

Analysis:
Addition: Included ORDER BY name to sort the retrieved cafes by their names in ascending order.
Implications: This modification will present the cafes near Central Park in alphabetical order based on their names.



```
SELECT name, ST_AsText(location) AS coordinates
FROM cafes
WHERE ST_DWithin(location, ST_SetSRID(ST_MakePoint(-73.968285, 40.78509
ORDER BY name; -- Sorts the results by cafe names alphabetically
```

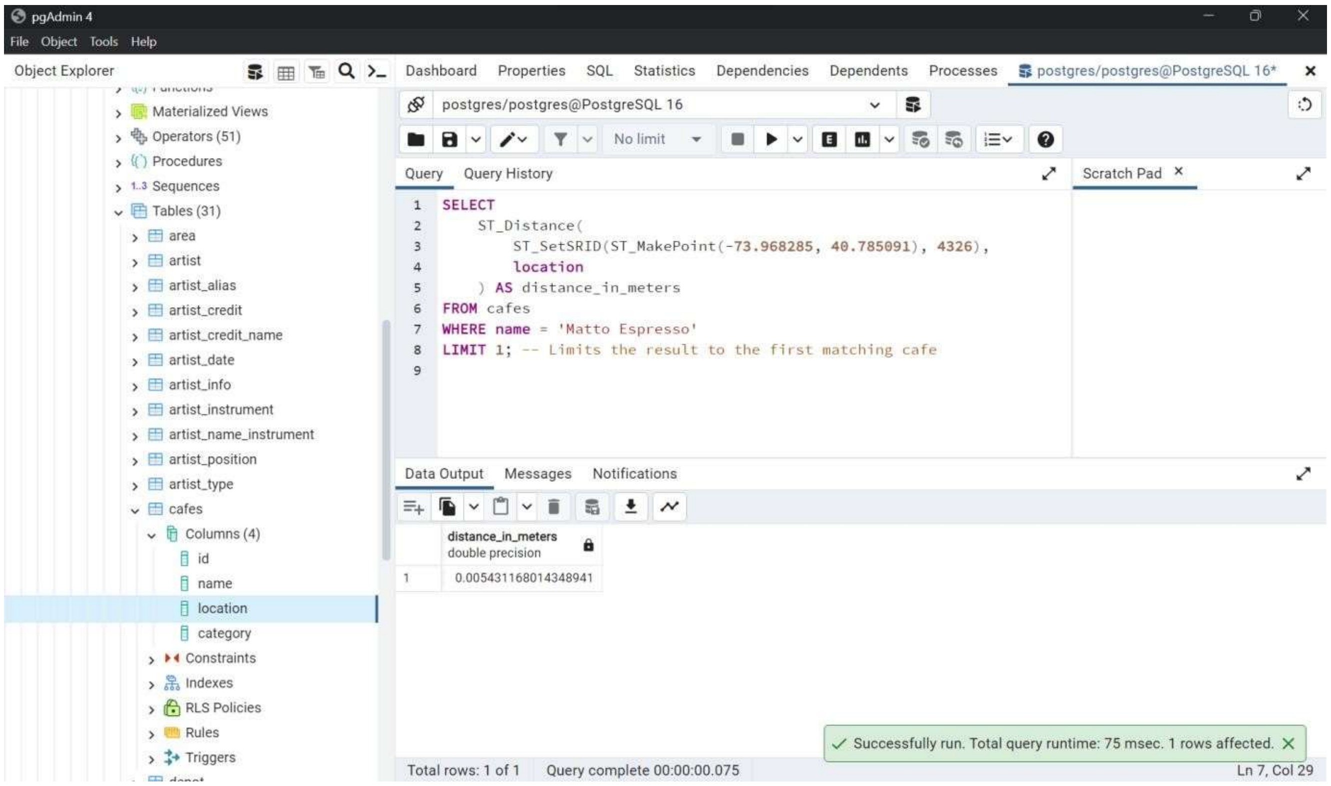
```
   name                      coordinates
   character varying (100)   text
1  Artisan café              POINT(-73.97501 40.79162)
2  Cafe Sabarsky             POINT(-73.9609 40.7826)
3  Charlotte café            POINT(-73.96832 40.79487)
4  Matto Espresso            POINT(-73.97342 40.78686)
5  Petrie Court café         POINT(-73.96403 40.78187)

Total rows: 5 of 5    Query complete 00:00:00.130    Ln 5, Col 1
```

Query 2: Calculate Distance between Points with Limiting Results

```
SELECT
    ST_Distance(
        ST_SetSRID(ST_MakePoint(-73.968285, 40.785091), 4326),
        location
    ) AS distance_in_meters
FROM cafes
WHERE name = 'Matto Espresso'
LIMIT 1; -- Limits the result to the first matching cafe
```

Analysis:
Addition: Included LIMIT 1 to restrict the output to only the first matching cafe.
Implications: This alteration ensures that the distance calculation is performed for a single cafe (in this case, the first matching one), potentially useful when multiple cafes share the same name.



```
SELECT
    ST_Distance(
        ST_SetSRID(ST_MakePoint(-73.968285, 40.785091), 4326),
        location
    ) AS distance_in_meters
FROM cafes
WHERE name = 'Matto Espresso'
LIMIT 1; -- Limits the result to the first matching cafe
```

```
   distance_in_meters
   double precision
1  0.005431168014348941
```

```
Successfully run. Total query runtime: 75 msec. 1 rows affected.
Total rows: 1 of 1    Query complete 00:00:00.075    Ln 7, Col 29
```

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

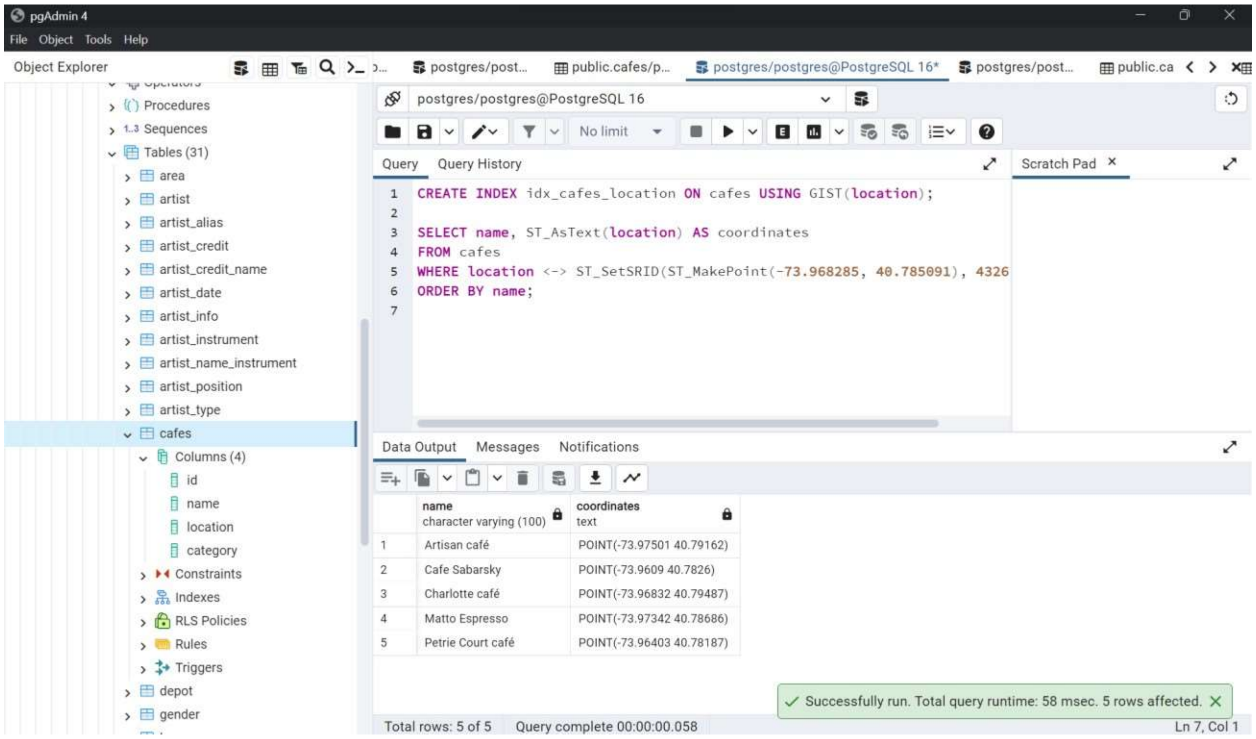
Optimizing SQL queries, especially those involving spatial data in a Geographic Information System (GIS) context, often involves employing indexing, appropriate spatial functions, and efficient data retrieval strategies. Here are ways to optimize the previous queries:

Query 1: Retrieve Locations of Specific Features with Sorting

To optimize the query fetching cafes near Central Park:

```
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;
```

Optimization Techniques:
Spatial Indexing: Created a GiST index on the location column to speed up spatial queries.
Distance Operator (<->): Used the <-> operator for faster distance calculations.



```
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
ORDER BY name;
```

	name character varying (100)	coordinates text
1	Artisan café	POINT(-73.97501 40.79162)
2	Cafe Sabarsky	POINT(-73.9609 40.7826)
3	Charlotte café	POINT(-73.96832 40.79487)
4	Matto Espresso	POINT(-73.97342 40.78686)
5	Petrie Court café	POINT(-73.96403 40.78187)

Successfully run. Total query runtime: 58 msec. 5 rows affected.

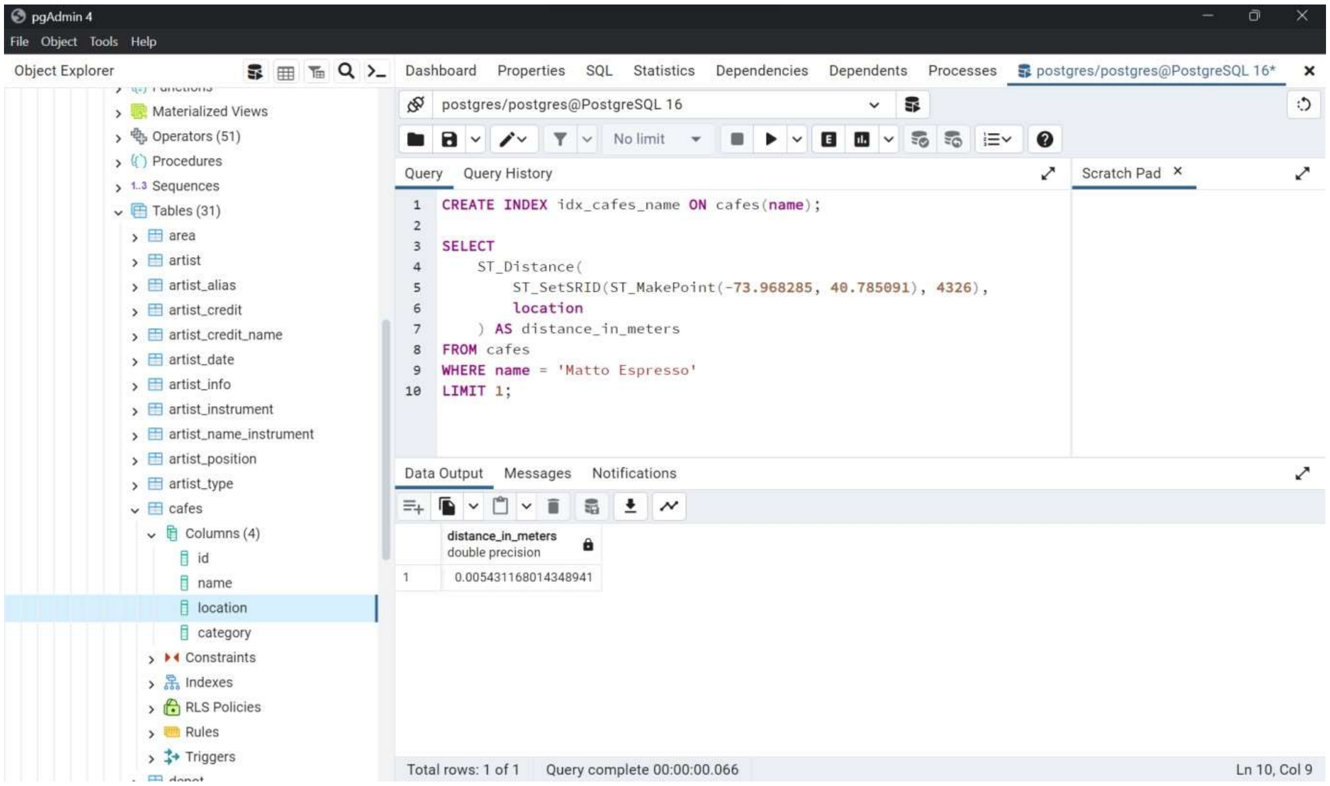
Total rows: 5 of 5 Query complete 00:00:00.058 Ln 7, Col 1

Query 2: Calculate Distance between Points with Limiting Results

```
CREATE INDEX idx_cafes_name ON cafes(name);

SELECT
    ST_Distance(
        ST_SetSRID(ST_MakePoint(-73.968285, 40.785091), 4326),
        location
    ) AS distance_in_meters
FROM cafes
WHERE name = 'Matto Espresso'
LIMIT 1;
```

Optimization Techniques:
Indexing: Created an index on the name column for quicker searching by cafe names.
Limiting Results: Kept the LIMIT 1 clause to restrict the output to a single row, avoiding unnecessary calculations for multiple rows.



```sql
CREATE INDEX idx_cafes_name ON cafes(name);

SELECT
    ST_Distance(
        ST_SetSRID(ST_MakePoint(-73.968285, 40.785091), 4326),
        location
    ) AS distance_in_meters
FROM cafes
WHERE name = 'Matto Espresso'
LIMIT 1;
```

```
distance_in_meters
double precision
1    0.005431168014348941

Total rows: 1 of 1    Query complete 00:00:00.066
```

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)

Ans: Final Project GitHub Link

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

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

Each member of the team posts the code of the project in GitHub. (INDIVIDUAL)

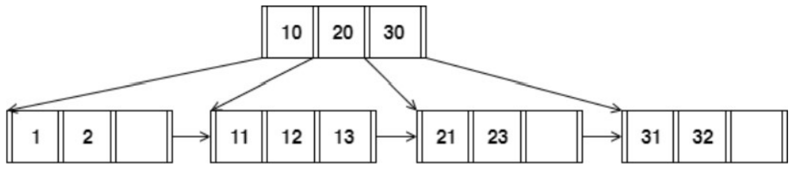
THEORY PART (24%)

You have 12 Theory questions, each with 2 marks.

1.) We have a file with a million pages (N = 1,000,000 pages), and we want to sort it using external merge sort. Assume the simplest algorithm, that is, no double buffering, no blocked I/O, and quicksort for in-memory sorting. Let B denote the number of buffers. How many passes are needed to sort the file with N = 1,000,000 pages with 6 buffers?

Ans: To sort 1,000,000 pages using external merge sort with 6 buffers, we perform an initial pass to create sorted runs using 5 buffers (since one is for output. In each pass, one buffer is used for output, leaving B−1=5 buffers for input). Each buffer sorts a portion of the file, resulting in 200,000 runs. In subsequent passes, 5 runs are merged at a time (one output buffer). The total passes needed are determined by how many times you can merge 5 runs into 1, repeating this until only one sorted run remains. This process takes a total of 9 passes, including the initial sorting pass and the merging passes.

2.) Consider the following B+tree.



When answering the following question, be sure to follow the procedures described in class and in your textbook. You can make the following assumptions:

- A left pointer in an internal node guide towards keys < than its corresponding key, while a right pointer guides towards keys ≥.
- A leaf node underflows when the number of keys goes below [(d−1)/ 2] e.
- An internal node(root node) underflows when the number of pointers goes below d /2 .

How many pointers (parent-to-child and sibling-to-sibling) do you chase to find all keys between 9 ∗ and 19∗ ?

Ans: 1 to get to the leaf node. 2 more until it sees a key > 19∗
So in total we need 3 pointers.

3.) Answer the following questions for the hash table of Figure 2. Assume that a bucket split occurs whenever an overflow page is created. h0(x) takes the rightmost 2 bits of key x as the hash value, and h1(x) takes the rightmost 3 bits of key x as the hash value

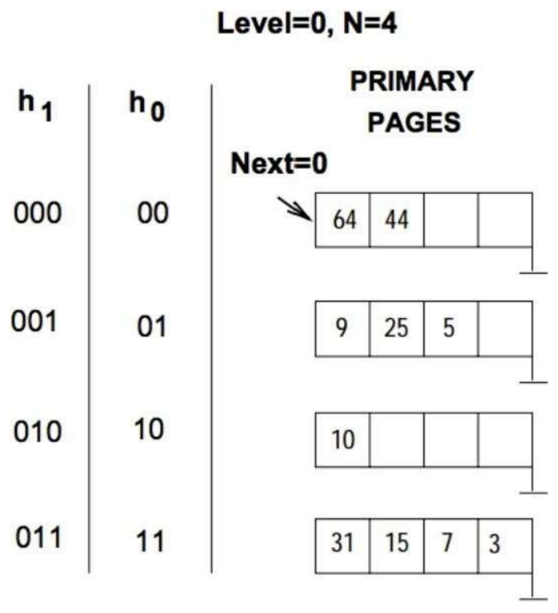


Figure 2: Linear Hashing

What is the largest key less than 25 whose insertion will cause a split?

Ans: In the given linear hashing structure, when inserting a key, the hash function being used at Level=0 is h0 (x), which considers the rightmost 2 bits of the key x. If we consider the binary representations of 24 ('11000') and 23 ('10111'), the rightmost 2 bits are '00' and '11', respectively. Inserting 24, which ends in '00', into the bucket indexed by '00' would not cause a split if there's still space left in the '00' bucket, which in this case there is, as it only contains the keys 64 and 44.
On the other hand, inserting 23, which ends in '11', into the bucket indexed by '11' would indeed cause a split if the '11' bucket is already full, as the keys 31, 15, 7, and 3 are already occupying the slots, assuming the buckets have a capacity of 3 keys and that an overflow page is created when a fourth key is inserted.
Given the current state of the hash table and the rules provided, inserting key 23 would cause a split because the bucket '11' is already at its maximum capacity and cannot accommodate another key.
Therefore, the largest key less than 25 whose insertion will cause a split is indeed 23.

4.) Consider a sparse B+ tree of order d = 2 containing the keys 1 through 20 inclusive. How many nodes does the B+ tree have?

Ans: In a sparse B+ tree of order d=2, each node carries the minimum number of keys, which is 2, except for the root node which can have at least 1 key. With keys ranging from 1 to 20:

- There are 10 leaf nodes since each one holds exactly 2 keys.
- The next level up has 5 nodes, with each parent node connecting to 2 leaf nodes.
- The process of halving continues up the tree levels, resulting in 3 nodes at the next level.
- The root node is a single node, as it points to the 3 nodes below it.
- Adding these nodes across all levels, the sparse B+ tree has **19** nodes in total.

5.) Consider the schema R(a,b), S(b,c), T(b,d), U(b,e).

Below is an SQL query on the schema:
SELECT R.a
FROM R, S,
WHERE R.b = S.b AND S.b = U.b AND U.e = 6
For the following SQL query, I have given two equivalent logical plans in relational algebra such that one is likely to be more efficient than the other:
I. $\pi a(\sigma c=3(R \bowtie b=b\ (S)))$
II. $\pi a(R \bowtie b=b\ \sigma c=3(S)))$

Which plan is more efficient than the other?

Ans: I. $\pi a(\sigma e=6(R \bowtie b=b(S \bowtie b=bU)))$ This plan first performs a join between S and U on their common attribute b and then filters the results where U.e equals 6. After that, it joins this result with R ontheir common attribute b and then projects a attribute.

I. $\pi a(R \bowtie b=b\sigma e=6(S \bowtie b=bU))$ This plan first joins S and U on their common attribute b, filters the results where U.e equals 6, and then immediately joins the filtered result with R on their common attribute b.
After the join, it projects a attribute.

The more efficient plan is typically the one that reduces the size of intermediate results as early as possible in the query execution. Filtering as early as possible (as soon as the necessary columns are available) generally reduces the number of tuples involved in subsequent joins.
In this case, Plan II is likely to be more efficient because it applies the selection condition $\sigma e=6$ on the join between S and U before joining with R. This means that R is joined with a potentially smaller subset of the S and U join, leading to fewer operations than joining first and then applying the selection, which is what Plan I does.

To summarize, II is likely to be more efficient than I because with the select operator applied first, fewer tuples need to be joined.

6.) In the vectorized processing model, each operator that receives input from multiple children requires multi-threaded execution to generate the Next() output tuples from each child. True or False? Explain your reason.

Ans: False.
In the vectorized processing model, operators efficiently handle inputs from various sources without the need for multi-threaded execution. This model works by processing data in groups or batches, allowing for quicker data handling compared to processing each item individually.
Contrastingly, in the iterator model, the process is more integrated. The Next () function is key here – it not only fetches the next batch of data but also directs the flow of operations. It's akin to getting both the information and instructions on what to do next in one go. This method intertwines data retrieval with the control flow, yet interestingly, it can be effectively managed using just a single thread.
To sum up, while the vectorized model excels in processing large data batches swiftly, the iterator

model combines data processing with sequential control, efficiently executable within a single-threaded environment.

7.) How can you optimize a Hash join algorithm?

Ans: Optimizing a hash join algorithm involves several key strategies: First, manage memory effectively to accommodate the hash table in-memory and minimize disk I/O. Choose the smaller of the two tables for constructing the hash table, reducing its size and the likelihood of overflowing memory. Implement a well-distributed hash function to minimize key collisions. Take advantage of parallel processing to distribute the workload across multiple CPUs. Use bloom filters for rapid checks of row correspondence, and process data in batches to improve cache efficiency. For large datasets, partition the data to ensure it fits in memory. In cases where data must spill to disk, optimize the read/write processes. Finally, tailor the optimization to your specific hardware setup, considering factors like cache size and disk speed.

8.) Consider the following SQL query that finds all applicants who want to major in CSE, live in Seattle, and go to a school ranked better than 10 (i.e., rank < 10).

Relation	Cardinality	Number of pages	Primary key
`Applicants (id, name, city, sid)`	2,000	100	`id`
`Schools (sid, sname, srank)`	100	10	`sid`
`Major (id, major)`	3,000	200	`(id,major)`

SELECT A.name

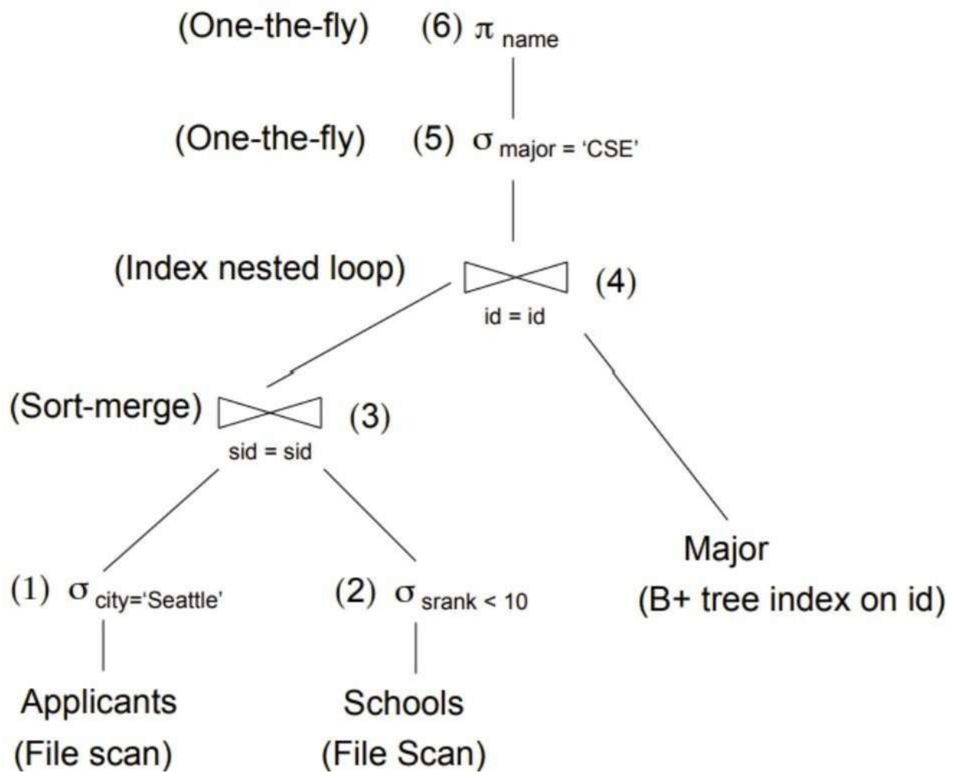
FROM Applicants A, Schools S, Major M

WHERE A.sid = S.sid AND A.id = M.id AND A.city = 'Seattle' AND S.rank < 10 AND M.major = 'CSE'

Assuming:

- Each school has a unique rank number (srank value) between 1 and 100.
- There are 20 different cities.
- Applicants.sid is a foreign key that references Schools.sid.
- Major.id is a foreign key that references Applicants.id.
- There is an unclustered, secondary B+ tree index on Major.id and all index pages are in memory.

You as an analyst devise the following query plan for this problem above:



What is the cost of the query plan below? Count only the number of page I/Os.

Ans: The query execution plan involves scanning the Applicants table, which consumes 100 I/Os, and scanning the schools table, which uses 10 I/Os. These scans generate 100 and 9 tuples, respectively. A sort-merge join on these results operates in memory, so it doesn't add to the I/O cost, and outputs 9 tuples. Following this, an index-nested loop join on the merged output requires 9 additional I/Os. The final two steps of the plan are executed without incurring further I/O costs. Summing up the I/Os from all these operations gives a total cost of 119 I/Os for the entire query plan.

9.) Consider relations R(a, b) and S(a, c, d) to be joined on the common attribute a. Assume that there are no indexes available on the tables to speed up the join algorithms. • There are B = 75 pages in the buffer

• Table R spans M = 2,400 pages with 80 tuples per page

• Table S spans N = 1,200 pages with 100 tuples per page

Answer the following question on computing the I/O costs for the joins. You can assume the simplest cost model where pages are read and written one at a time. You can also assume that you will need one buffer block to hold the evolving output block and one input block to hold the current input block of the inner relation.

A.) Assume that the tables do not fit in main memory and that a high cardinality of distinct values hash to the same bucket using your hash function h1. What approach will work best to rectify this?

Ans: To address the issue of high cardinality values leading to hash collisions in join operations, consider three main strategies. Firstly, refine the hash function to ensure a more uniform distribution of keys, decreasing the likelihood of collisions. Secondly, use a partition-based hash join like the Grace Hash Join, which breaks down the data into smaller, manageable chunks that fit in memory, thus cutting down on I/O costs. Lastly, if feasible, expand the buffer size to hold more data in memory, which can further reduce collisions. Or Create hashtables for the innner and outer relation using h1 and rehash into an embedded hash table using h2 != h1 for large buckets.

B.) I/O cost of a Block nested loop join with R as the outer relation and S as the inner relation

Ans: M + [M/B-2] * N = 2400 + [2400/73] * 1200 = 2400 + 39600 = 42000

10.) Given a full binary tree with 2n internal nodes, how many leaf nodes does it have?

Ans: In a full binary tree with 2n internal nodes:
The total number of nodes T is the sum of the number of internal nodes I and the number of leaf nodes L.
Each internal node contributes two children, leading to the equation T=I+L.
In full binary trees, the number of leaf nodes is always one more than the number of internal nodes, so L=I+1.
Given I=2n, we find that the number of leaf nodes L is 2n+1.

11.) Consider the following cuckoo hashing schema below:

Both tables have a size of 4.The hashing function of the first table returns the fourth and third least significant bits: h1(x) = (x >> 2) & 0b11.The hashing function of the second table returns the least significant two bits: h2(x) = x & 0b11.

When inserting, try table 1 first. When replacement is necessary, first select an element in the second table. The original entries in the table are shown in the figure below.

TABLE 1	TABLE 2
	13
12	

What sequence will the above sequence produce? Choose the appropriate option below:

a.) Insert 12, Insert 13
b.) Insert 13, Insert 12
c.) None of the above. You cannot have more than 1 Hash table in Cuckoo hashing
d.) I don't know

Ans:

a.) Insert 12, Insert 13

In cuckoo hashing, when inserting keys into two hash tables, if a key encounters a collision, it displaces the existing key, which then seeks a place in the other table. With keys 12 and 13, using hash functions h1 and h2, both hash to position 3 in Table 1. Inserting 12 first and then 13 causes 12 to be kicked out. 12 then goes to its alternate position, 0 in Table 2, determined by h2, where it fits without issue. This sequence ensures both keys are placed successfully: 12 in Table 2 and 13 in Table 1, making option a (Insert 12, Insert 13) the correct outcome.