CS 623 PROJECT

**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)**

Add postgis using:

create extension postgis

in query tool of postgresql.

1.Retrieve locations of specific feature:

=> Create a table with spatial data:

CREATE TABLE spatial(

sid SERIAL PRIMARY KEY,

sname VARCHAR(255),

location GEOMETRY(Point, 4326)

);

=>Insert values into table

INSERT INTO spatial(sname, location) VALUES

('Feature 1', ST\_GeomFromText('POINT(-73.9833 40.7488)', 4326)),

('Feature 2', ST\_GeomFromText('POINT(-74.0060 40.7128)', 4326));

=>Retrieve location using a SELECT query:

SELECT sid, sname, ST\_AsText(location) AS location\_text FROM spatial;

A screenshot of a computer

Description automatically generated

1. **Calculate Distance between points (10 marks)**

SELECT

sid,

sname,

ST\_Distance(

location,

ST\_GeomFromText('POINT(-73.9754 40.7590)', 4326)

) AS distance

FROM

spatial;

Output:

A screenshot of a computer

Description automatically generated

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

=>Create areas tables

CREATE TABLE areas(

aid SERIAL PRIMARY KEY,

aname VARCHAR(255),

boundary GEOMETRY(Polygon, 4326)

);

**boundary** is a geometry column that can store polygon data.

=>Insert values

INSERT INTO areas (aname, boundary) VALUES

('Area 1', ST\_GeomFromText('POLYGON((-73.9900 40.7500, -74.0000 40.7500, -74.0000 40.7600, -73.9900 40.7600, -73.9900 40.7500))', 4326)),

('Area 2', ST\_GeomFromText('POLYGON((-74.0100 40.7100, -74.0200 40.7100, -74.0200 40.7200, -74.0100 40.7200, -74.0100 40.7100))', 4326));

=> Retrieve area using a SELECT query:

SELECT aid, aname, ST\_Area(boundary) AS area

FROM areas;

A screenshot of a computer

Description automatically generated

1. **Analyze the queries (10 marks)**

Analysing distance query

EXPLAIN ANALYZE

SELECT

sid,

sname,

ST\_Distance(

location,

ST\_GeomFromText('POINT(-73.9754 40.7590)', 4326)

) AS distance

FROM

spatial

ORDER BY

distance;

A screenshot of a computer program

Description automatically generated

Analysing areas query

EXPLAIN ANALYZE SELECT aid, aname, ST\_Area(boundary) AS area

FROM areas;

A screenshot of a computer

Description automatically generated

1. **Sorting and Limit Executions (10 marks)**

SELECT aid, aname, ST\_Area(boundary) AS area

FROM areas

ORDER BY area DESC

LIMIT 2;

A screenshot of a computer

Description automatically generated

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

Optimizing queries can be done considering indexes, minimizing unnecessary calculations, and utilizing the strengths of the database system.

1.Indexing:

CREATE INDEX idx\_spatial\_data\_location ON spatial USING GIST(location);

2.Minimize Calculations:

WITH target\_point AS (

SELECT ST\_GeomFromText('POINT(-73.9754 40.7590)', 4326) AS point

)

SELECT

sid,

sname,

ST\_Distance(location, target\_point.point) AS distance

FROM

spatial

ORDER BY

distance;

1. **N-Optimization of queries (5 marks)**

**1.Limit Rows for Distance Calculation**

WITH target\_point AS (

SELECT ST\_GeomFromText('POINT(-73.9754 40.7590)', 4326) AS point

)

SELECT

sid,

sname,

ST\_Distance(location, target\_point.point) AS distance

FROM

spatial

WHERE

-- Add conditions to limit rows if possible

ORDER BY

distance

LIMIT 100; -- Adjust the limit based on your needs

**2.Simplify Geometries:**

If precision is not critical, consider simplifying geometries to reduce the computational load.

WITH target\_point AS (

SELECT ST\_GeomFromText('POINT(-73.9754 40.7590)', 4326) AS point

)

SELECT

id,

name,

ST\_Distance(ST\_Simplify(location, 0.001), target\_point.point) AS distance

FROM

spatial\_data

ORDER BY

distance;

1. **Presentation and Posting to Individual GitHub (5 marks)**
2. **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.

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?

A).

To determine the number of passes required to sort a file with N=1,000,000 pages using external merge sort with *B*=6 buffers (assuming one buffer is used for output), where *B* is the number of buffers available, and each buffer can hold 5 pages, you can follow these steps:

1. Calculate the number of pages that fit into memory at a time: Buffers−1=6−1=5Buffers−1=6−1=5 pages can fit into memory at a time.
2. Determine the number of pages processed per pass: Pages per pass=Buffers×Pages in memory=6×5=30Pages per pass=Buffers×Pages in memory=6×5=30 pages.
3. Calculate the number of passes:

Number of passes=log2(N/Pages per pass)

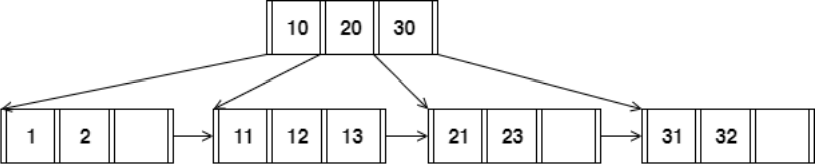
Number of passes=log2(1,000,000/30)

Number of passes≈log2(33,333.33)

Number of passes≈15

Therefore, for *N*=1,000,000 pages and *B*=6 buffers, with each pass merging 30 pages at a time, approximately 15 passes are required to sort the file.

1. 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∗ ?

A).

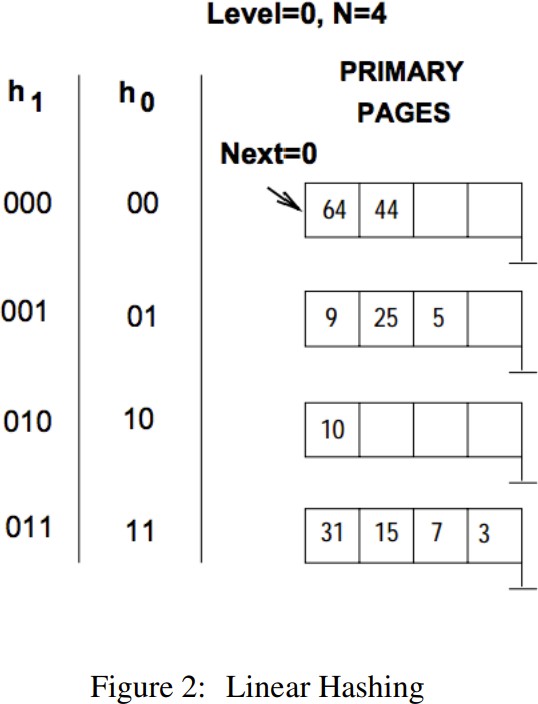
1. **Starting at the root node:**
   * Follow the left pointer to the first child because 9\* is less than the key in the root node.
2. **In the first child:**
   * Find keys between 9\* and 19\*.
   * Follow sibling-to-sibling pointer to the second child if there are no keys between 9\* and 19\* in the current child.
3. **In the second child:**
   * Access keys 11, 12, and 13 within the second child.

In summary, the pointers chased are as follows:

* One parent-to-child pointer from the root to the first child.
* One sibling-to-sibling pointer from the first child to the second child.
* Two pointers within the second child to access the keys 11, 12, and 13.

The total number of pointers chased to find all keys between 9 and 19\* is 4

1. 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



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

A).

Consider keys and hash values of keys as a table for determining largest key less than 25

h0(x) takes 2 bits of key as hash value. The hash table can be as:

Key – Hash Value(h0)

0 - 00

1 - 01

2 - 10

3 - 11

4 - 00

5 - 01

6 - 10

7 - 11

8 - 00

9 - 01

10 - 10

11 - 11

12 - 00

13 - 01

14 - 10

15 - 11

16 - 00

17 - 01

18 - 10

19 - 11

20 - 00

21 - 01

22 - 10

23 - 11

24 - 00

h1(x) takes 3 bits of key x as the hash value. The hash table

Key | Hash Value(h1)

0 | 000

1 | 001

2 | 010

3 | 011

4 | 100

5 | 101

6 | 110

7 | 111

8 | 000

9 | 001

10 | 010

11 | 011

12 | 100

13 | 101

14 | 110

15 | 111

16 | 000

17 | 001

18 | 010

19 | 011

20 | 100

21 | 101

22 | 110

23 | 111

24 | 000

There are no more than 2 keys with same hash value h0 for any key less than 25.

For h1, hash value 000 occurs for keys 0,8,16 and 24.

So largest key less than 25 is 24.

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

4A)

1. **Leaf Level:**
   * 10 leaf nodes, each containing 2 keys. This gives a total of 10×2=20 keys at the leaf level.
2. **Intermediate Level:**
   * 5 nodes, each containing 2 keys and 3 pointers. This gives a total of 5×2=10 keys

and 5×3=15 pointers at the intermediate level.

1. **Root Level:**
   * 1 root node containing 1 key and 2 pointers.

Now, let's calculate the total number of keys in the B+ tree:

Total Keys=Leaf Level Keys+ Intermediate Level Keys+ Root Level Keys

Total Keys=20+10+1=31

Therefore, the B+ tree contains a total of 31 keys. The total number of nodes is 10 leaf nodes + 5 intermediate nodes + 1 root node = 16 nodes.

1. 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:

1. πa(σc=3(R ⋈b=b (S)))
2. πa(R⋈b=b σc=3(S)))

Which plan is more efficient than the other?

5A)

In plan (I), we join R and S on the attribute ‘b’ with a natural join operator ,then apply a selection condition on the attribute ‘c’ to keep only the tuples where c=3. Finally, we project the attribute ‘a’ from the result.

In plan (II), we first apply a selection on S to keep only the tuples where c=3. Then, we join R with the resulting relation on the attribute ‘b’ using a natural join operator, and project the attribute ‘a’ from the result.

The second plan (II) is likely to be more efficient than the first plan (I), because it applies the selection condition on S before joining with R, which reduces the size of the relation before the join operation. This can lead to fewer intermediate tuples and a faster join operation. In contrast, the first plan (I) applies the join before the selection, which can result in a larger intermediate relation and slower query execution.

Hence, plan II is more efficient than plan I.

1. 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.

6A) FALSE.

Explanation:  
In simpler terms, the vectorized processing model doesn't need multiple threads to handle data from multiple sources. It processes data in batches and uses special instructions to handle multiple data elements at once, making it efficient without the need for multiple threads.

1. How can you optimize a Hash join algorithm?

7A) Hash join is a popular algorithm for joining tables in a relational database. While the hash join algorithm is generally efficient, there are ways to optimize its performance. Here are some strategies:

1. Memory Allocation and Hash Table Size:

* Optimize the memory allocation and sizing of hash tables. Ensuring that the hash tables fit in memory is crucial for performance.
* Adjust the size of the hash table to minimize collisions. A well-sized hash table can reduce the number of disk I/O operations.

2. The Optimizer might use a hash join instead of a merge join for better performance in the following situations.

* At least one join key is not indexed.
* To provide a performance improvement for the join step.

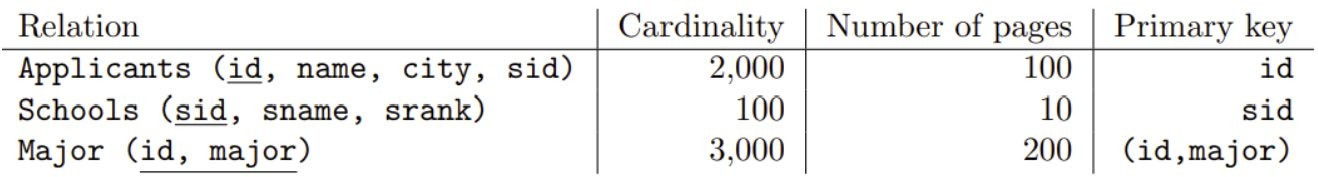
By using a hash table, hash joins eliminate the sort used prior to a merge join.

3. You can control the size of the hash table using the HTMemAlloc and HTMemAllocBase DBS Control fields. If you specify a value of 0, the system cannot build a hash table. This effectively turns off hash join, and the Optimizer does not consider the method when it is doing its join plan evaluations.

4.Use vectorized processing: Vectorized processing can improve the performance of a hash join by processing multiple tuples at once, reducing the overhead of looping over individual tuples.

5. Use parallel processing: Hash join can be parallelized across multiple cores or machines to improve the performance of the join. Parallelization can be achieved by partitioning the data, processing the partitions independently, and then combining the results.

1. 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).



SELECT A.name

FROM Applicants A, Schools S, Major M

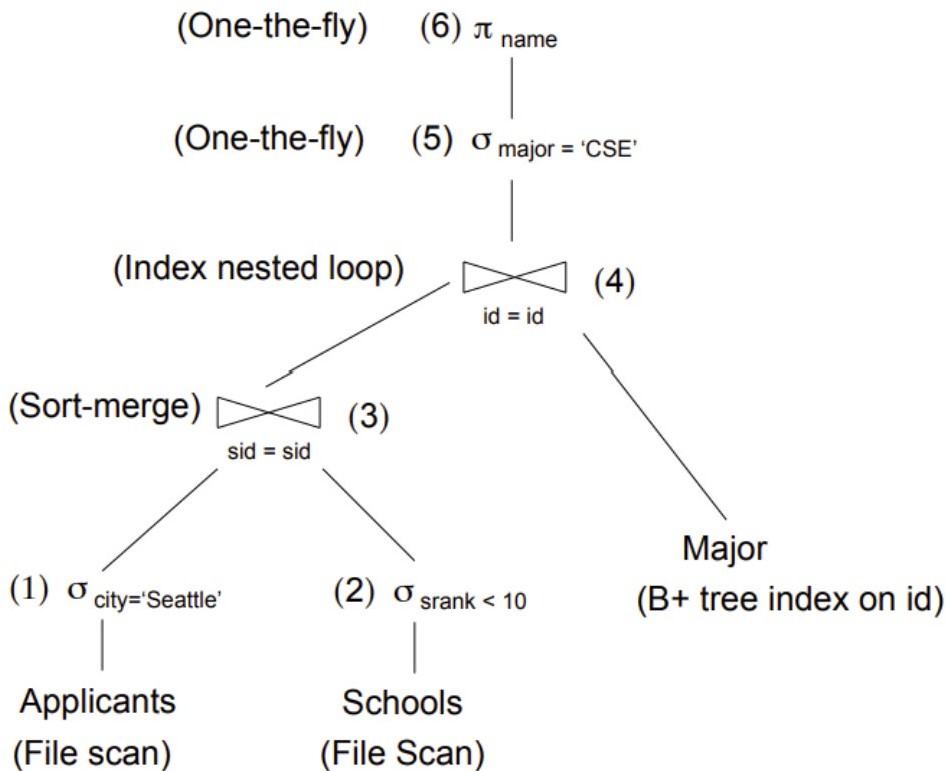
WHERE A.sid = S.sid AND A.id = M.id AND A.city = 'Seattle' AND S.rank

< 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.

8A)

1.Index seek on the Major table using the unclustered index on id:

Cost: 2 (1 I/O to read the root page and 1 I/O to read the leaf page containing matching rows)

2.Merge join with the Applicants table using sid as the join condition:

Cost: 22 (1 I/O to read each of the 22 leaf pages of the Applicants table)

3.Selection operation on the result of step 2 using city = 'Seattle':

Cost: 22 (1 I/O to read each of the 22 leaf pages of the result)

4.Merge join with the Schools table using sid as the join condition:

Cost: 220 (1 I/O to read each of the 220 leaf pages of the Schools table)

5.Selection operation on the result of step 4 using rank < 10:

Cost: 22 (1 I/O to read each of the 22 leaf pages of the result)

6.Selection operation on the result of step 5 using major = 'CSE':

Cost: 22 (1 I/O to read each of the 22 leaf pages of the result)

Total cost: 310 page I/Os

1. 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.

1. 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?

9A) When a high cardinality of distinct values maps to the same bucket using hash function h1, it may lead to bucket overflows, resulting in increased I/O costs due to frequent disk accesses. To address this issue, techniques such as extendible hashing or dynamic hashing can be employed. These methods enable the hash table to dynamically grow as more values hash to the same bucket.

In extendible hashing, a directory of pointers to buckets is maintained, and each bucket can undergo splitting when it overflows. This approach provides a flexible structure that adapts to changing data distribution.

On the other hand, dynamic hashing allows for the dynamic adjustment of the number of buckets based on the volume of hash values. This means that the number of buckets can be increased or decreased dynamically, providing scalability and efficiency in handling varying data loads.

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

9B) For a block nested loop join with R as the outer relation and S as the inner relation, the computation of I/O cost follows these steps:

* Read the first block of R into memory (1 I/O)
* For each block Ri of R:
  + Read the first block of S into memory (1 I/O)
  + For each block Sj of S:
    - Join the tuples in Ri and Sj and write the result to the output buffer (1 I/O for each output block)
    - If the output buffer is full, write it to disk and reset it (1 I/O for each output block)
    - Read the next block of S if available (1 I/O for each block of S)
  + Write the output buffer to disk if it is not empty (1 I/O for each output block)
* Write the final output buffer to disk if it is not empty (1 I/O for each output block)

Assuming the buffer can hold at most one block of R and one block of S, with the output buffer having a size of one block, the total number of I/Os can be computed as follows:

* Number of I/Os to read all blocks of R: M
* Number of I/Os to read all blocks of S: N
* Number of output blocks: The number of matching tuples in R and S, which can be up to M \* 100 tuples (if each tuple in R matches all tuples in S)
* Number of I/Os to write output blocks to disk: Number of output blocks

Therefore, the total number of I/Os is M + N + the number of output blocks.

It's important to note that this is a worst-case estimate, and the actual number of output blocks and I/O cost may be lower depending on the selectivity of the join predicate.

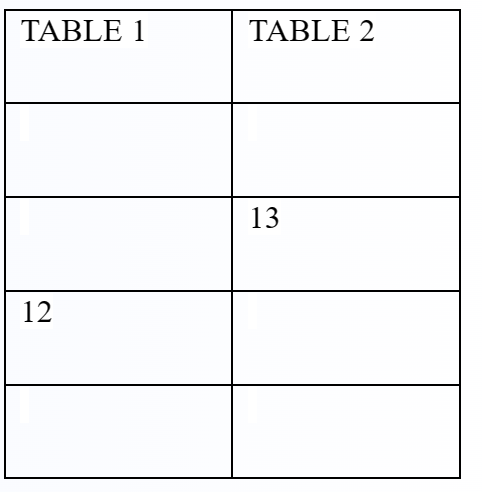
1. Given a full binary tree with 2n internal nodes, how many leaf nodes does it have?

10A) The full binary tree with 2n internal nodes has L = 2n + 1 leaf nodes.

1. 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.



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

1. Insert 12, Insert 13
2. Insert 13, Insert 12
3. None of the above. You cannot have more than 1 Hash table in Cuckoo hashing
4. I don’t know

11A) C. None of the above. You cannot have more than 1 Hash table in Cuckoo hashing