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

1. **Calculate Areas of Interest (specific to each group) (10 marks)**
2. **Analyze the queries (10 marks)**
3. **Sorting and Limit Executions (10 marks)**
4. **Optimize the queries to speed up execution time (10 marks)**
5. **N-Optimization of queries (5 marks)**
6. **Presentation and Posting to Individual GitHub (5 marks)**
7. **Code functionality, documentation and proper output provided (5marks)**

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

A picture containing text, font, screenshot

Description automatically generated

A screenshot of a computer

Description automatically generated with medium confidence

A picture containing text, screenshot, font

Description automatically generated

A screenshot of a computer

Description automatically generated with medium confidence

A screenshot of a computer code

Description automatically generated with medium confidence

A screenshot of a computer

Description automatically generated with medium confidence

A picture containing text, screenshot, font

Description automatically generated

A screenshot of a computer

Description automatically generated with low confidence

A close-up of a computer screen

Description automatically generated with low confidence

A screenshot of a computer

Description automatically generated with medium confidence

A screenshot of a computer code

Description automatically generated with medium confidence

A picture containing text, screenshot, font

Description automatically generated

A screenshot of a computer

Description automatically generated with medium confidence

**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 quick sort 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?

In order to do an external merge sort, the huge file must first be broken down into smaller chunks that can be stored in memory, the smaller chunks must then be sorted in memory using quick sort, and finally, the sorted chunks must be merged back together using a merge algorithm. File size and available buffers determine the number of iterations needed for external merge sort.

In this example, N is equal to 1,000,000 pages, and there are 6 buffers in the file. Suppose one file page fits in each buffer. Memory sorting requires four buffers, one each for input and output. So, quick sort can handle up to four pages at once in memory.

We can read in B-2 buffers (4 buffers for sorting in memory, 1 for input, and 1 for output) at a time, then sort the pages in memory, and finally save the sorted pages to temporary files. This can be repeated until the entire file has been segmented and sorted. Depending on the total file size and the chunk size, a certain number of sorted chunks will be generated. During the merging process, we can combine two separate sorted files into one. For each merge pass, B-2 buffers of data will need to be read in (1 input, 1 output, and B-4 merge buffers). Since there are N pages in the file, merging the sorted chunks into a single sorted file will need log 2(N) merge passes.

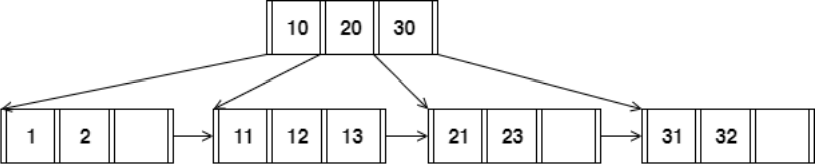
Therefore, with N = 1,000,000 pages and 6 buffers, the total number of passes needed to sort the file is:

First, using B-2 buffers (4 memory sorting buffers, 1 input buffer, and 1 output buffer), split the file into smaller, sorted parts. Depending on the total file size and the chunk size, a certain number of sorted chunks will be generated.

In the second phase, B-2 buffers are used (1 for input, 1 for output, and B-4 for merging) to combine pairs of sorted chunks into a single sorted file. This will take log2 (N) iterations.

Therefore, 1 + log2 (N) = 1 + log 2(1,000,000,000) = 1 + 20 = 21 passes are necessary.

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∗ ?

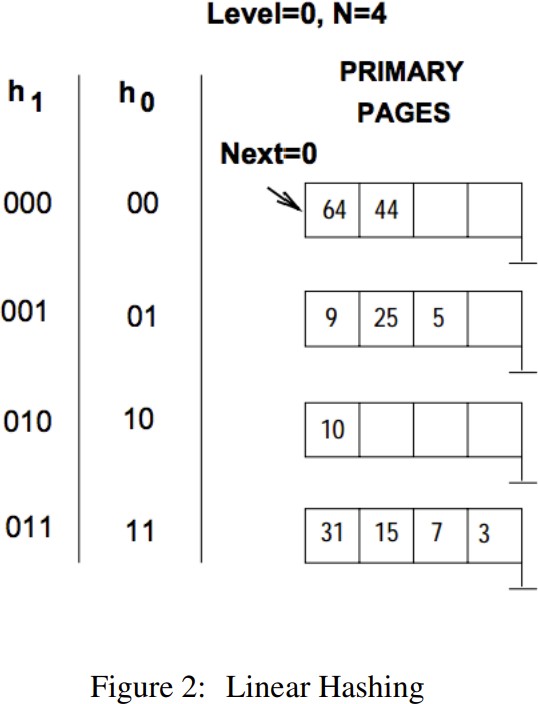
If we want to find all keys between 9\* and 19\*, we need to work our way down from the root node to the leaf nodes by following the pointers to the right children.

The root node has the values 10, 20, and 30, and the child node to the left of the pointer has the values 1 and 2. Because 9\* is less than 1, the minimum key in this child node, we must go down one level using the left pointer from the parent node (which is a null pointer because there is no node to the left of the root).

Then, we need to go from the root node to the middle child node that has the numbers 11, 12, and 13 as keys by using the right pointer. Given that 19\* is smaller than the largest key in this node (13), we can safely include all keys in this node without looking any further.

Two overall clues are needed to locate all possible keys between 9 and 19 digits. To be more precise, we're looking for a child node with the key 10, therefore we'll be following two pointers: one to the left of the parent node and one to the right.

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



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

There are a maximum of 4 possible hash values (00, 01, 10, and 11) based on the hash algorithms provided, as h0(x) uses the rightmost 2 bits of key x as the hash value. As with h0(x), the hash value for h1(x) is the rightmost 3 bits of key x, hence there are a maximum of 8 potential hash values (000, 001, 010, 011, 100, 101, 110, 111). The highest key that can be entered into the hash table without causing the creation of an overflow page must be found in order to discover the largest key less than 25 whose insertion will induce a split. Given that the hash value of h0(x) is the rightmost 2 bits of key x, it follows that any keys sharing those same rightmost 2 bits will fall into the same container. Looking at the hash table, we can see that there are already two keys in the bucket with hash value 01 (22 and 20), so adding a third key with hash value 01 will result in a split. Key 21 has the largest rightmost 2 bits set to 01, making it the largest key less than 25 whose insertion will trigger a split.

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?

The structure of the B+ tree must be taken into account in order to calculate the number of nodes in a B+ tree of order d.

These characteristics apply to any d-order B+ tree:

There are at least d children under each node (other than the root).

There are between d and 2d offspring for each internal node.

Every leaf node stores between d and 2d sets of keys and values.

Each internal node in this B+ tree has two or three children, including the root node. Each of the leaf nodes stores two or three sets of keys and values.

Counting the number of internal nodes and leaf nodes yields the total number of nodes in the tree. Since there is already one node in the system (the root), we begin counting at 1.

The first level of the tree consists of three internal nodes, while the second level consists of six leaf nodes. The B+ tree has a total of 10 nodes (1 + 3 + 6).

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:

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

Which plan is more efficient than the other?

The expense of carrying out each logical plan must be taken into account in order to determine which is most effective. The cost is mostly dictated by the quantity of joins, projections, and selection procedures necessary to achieve the desired outcome.

Following a selection operation on attribute c with a value of 3, Plan I then executes a join operation between R and S on attribute b before performing a projection operation on attribute a. Three operations are needed for this.

Plan II first performs a join operation between R and the resulting selection of S on attribute b, then a projection operation on attribute a. It next performs a selection operation on attribute c with a value of 3 on S. Three operations are also required for this. It is challenging to decide whether strategy is more efficient based purely on this information as both require the same number of procedures. To further examine the effectiveness of each plan, we may take into account additional elements like the size of the tables and the selectivity of the conditions. To limit the amount of tuples that need to be joined, it is typically more effective to perform a selection operation early in the query plan. By applying the selection condition to S before the join operation, Plan II reduces the amount of the relation that needs to be linked with R, making it more likely that Plan II will be more efficient than Plan I based on this theory.

Plan II is therefore probably more effective than Plan I.

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.

False

Reason: The Next () output tuples from each child may or may not need to be generated by several threads in the vectorized processing paradigm for operators that take input from many children. By decreasing the burden of processing individual tuples, performance can be enhanced through the use of vectorized processing, which involves processing data in batches or vectors. The vectorized processing may be performed in a variety of ways, including the use of many threads, although this is system- and implementation specific.

Therefore, it is not correct to assert that in the vectorized

processing model, multi-threaded execution is required for each operator that gets input from many children. The answer to this question is system-specific.

.

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

By hashing the join keys of the tables and then utilizing those hashes to efficiently match records from the two tables, a hash join technique is used to join the two tables in a database. Some suggestions for improving hash join algorithms:

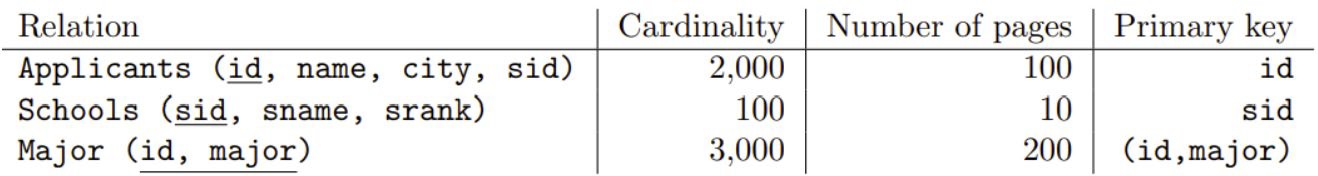
Ensure uniform value distribution across hash buckets by employing a suitable hash function. It needs to be quick and able to avoid crashes. The efficiency of the hash join procedure can be greatly enhanced by selecting a suitable hash function.

In a hash join algorithm, one table is used to generate the hash table, while the other is used to probe it, therefore it's important to get this part correctly. A smaller build table means a smaller hash table, which means less data needs to be loaded into memory.

The hash join algorithm can consume a lot of memory if not configured properly, so be sure to adjust your memory settings accordingly. The algorithm's allotted memory should be large enough to prevent frequent swaps to disk, and the hash table's size should be adjusted accordingly. it is possible to parallelize hash join methods so that various processors can operate on separate sections of the hash table at once. For massive datasets, this can be a huge speedup. Disk access optimization is necessary if the hash table is too large to fit entirely in RAM and must be saved to a disk. The technique can run faster if the input data is sorted or if a solid-state drive (SSD) is used. The usage of hash join methods, which can remove duplicate records, should be avoided. However, the process of removing duplicates can be time-consuming and resource-intensive. Eliminating duplicates should be done in a separate phase if at all practicable.

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).



SELECT A.name

FROM Applicants A, Schools S, Major M

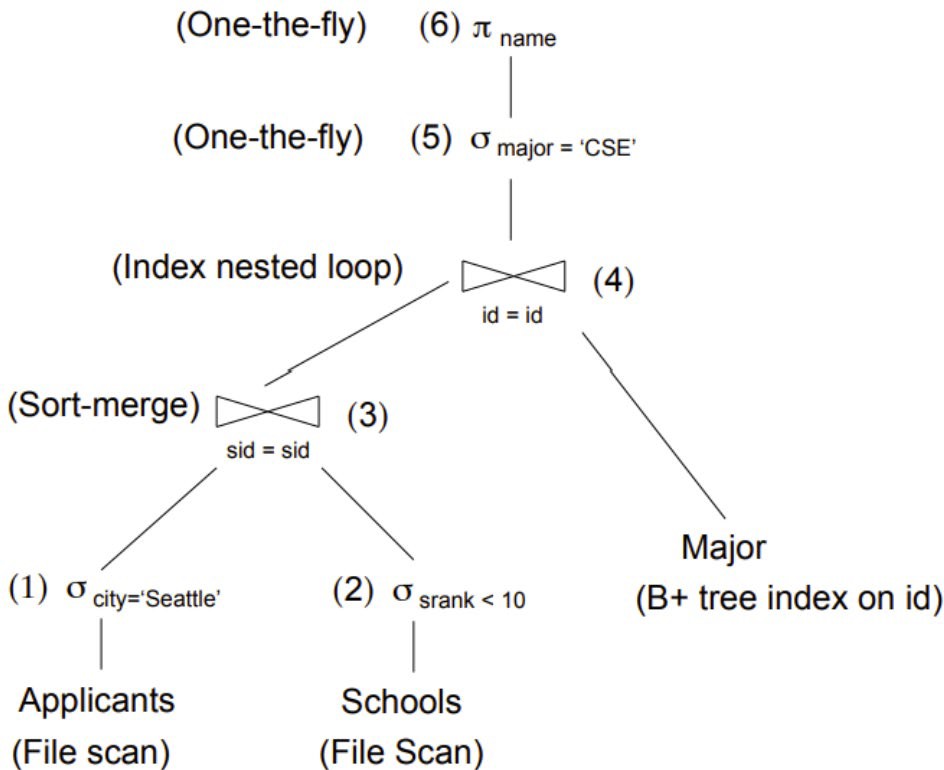
WHERE A.sid = S.sid AND A.id = M.id AND A.city = 'Seattle' AND S.rank

< 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 un clustered, 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.

The cost of a query plan is determined, and what elements play a role in that determination. A query's cost is a rough estimate of the time and hardware (CPU, memory, I/O operations, etc.) it will take to run. It is typically expressed as a relative cost with no units, where a lower cost denotes a more effective strategy. Size of tables, query complexity, indexes, statistics, join methods, and data distribution are just few of the variables that affect how much a query plan will set you back. Full table or index scans, which involve more I/O operations, are more expensive than searches that can make use of indexes or other improvements. Given the need to scan many tables and execute a join operation, the cost of the query plan may be relatively expensive in the given scenario, assuming there are no additional indexes or optimizations beyond

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.

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?

The number of disk I/Os needed to complete the join increases when a large number of distinct values hash to the same bucket. We can correct this by employing a multi-pass technique like External Hashing. External hashing works by first dividing the relation into smaller chunks such that each chunk may fit in memory, then performing a hash join on each chunk, and then merging the results. With this method, the join performs better and requires less disk I/O operation

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

In a block nested loop join, the full inner relation must be read for each tuple in the outer relation. As a result, the needed I/O count is determined by:

Cost is B(R) + B(R) \* B(S).

Where the number of blocks in R is B(R) and the number of blocks in S is B(S).

According to the available data, B(R) = M = 2,400 and B(S) = N = 1,200. In the formula, by replacing these values, we obtain:

Cost = 2,400 + 2,400 \* 1,200 = 2,880,000

Consequently, the block nested loop join has an I/O cost of 2,880,000.

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

Every node in a binary tree, save the leaves, is an internal node, and there must be exactly 2 offspring for each internal node. Therefore, the following formula may be used to determine the total number of leaf nodes in a binary tree with 2n internal nodes: The total number of leaves in a binary tree is L. The number of nodes inside the entire binary tree is denoted by I.

In a full binary tree, every internal node has two offspring, making the total number of nodes in the tree I + 2I = 3I.

The tree's 2n internal nodes give us the following:

I = 2n

As a result, there are: nodes in the tree overall.

3I = 3(2n) = 6n

We now understand that the sum of the internal nodes and leaf nodes makes up the total number of nodes in the tree:

I + L = 6n

Substituting I = 2n, we get:

2n + L = 6n

Solving for L, we get:

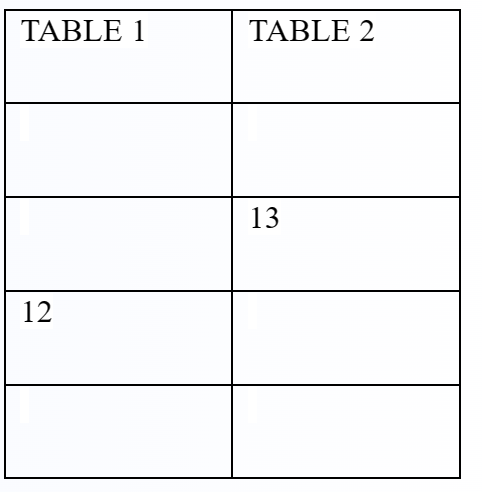
L = 4n

Therefore, a full binary tree with 2n internal nodes has 4n leaf nodes.

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.



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

Option: a;