**Admission Number:**103428

**NAME:** Fredah Ndila Kioko

**GROUP:** A

**QUESTION: *Write a 700 – 2,000-word essay on how each of the 8 cost estimation techniques (refer to slides 75 to 89) can be combined with various heuristic rules during query optimization by the DBMS and the effect this has on reducing the workload of a DBA***

COST ESTIMATION TECHNIQUES AND HOW THEY ENTERWINE WITH QUERY OPTIMIZATION HEURISTIC RULES

**Introduction**

A query in an advanced database is a question or inquiry to a set of data stored in attributes of a given relation. In SQL (Structured Query Language) to retrieve data from databases. The query uses a plan that consists of a series of ordered steps to access the data in relations. That is the statement needs a SELECT clause to instantiate the data search, a FROM clause to tell it from which table(s) to pull data, and a WHERE clause to restrict/filter the results. Sometimes a relation

May contain many unstructured tables and the query may be run often to ensure that the run time of the query optimization is necessary.

A single query can be executed through different algorithms or re-written in different forms and structures. Hence query optimization determines the most efficient way to execute a given query by considering the possible query plans. The importance of query optimization is to reduce the system resources required to fulfill a query and eventually provide the user with the correct result set faster it also allows the system to service more queries in the same amount of time, because each request takes less time than optimized queries. Thirdly, query optimization ultimately reduces the amount of wear on the hardware components example the hard drive and allows the server to run more efficiently by consuming less power and less memory usage.

Query optimization applies transformation rules to convert one relational algebra expression into an equivalent expression that is known to be more efficient. Hence there are broadly two ways a query can be optimized:

1. Heuristic approach analyzes and transforms equivalent relational expressions: Try to minimize the tuple and column counts of the intermediate and final query processes. This approach to query optimization orders the operations in a query using transformation rules that are known to generate good execution strategies.
2. Cost estimation approach: These underlying algorithms determine how tuples are accessed from the data structures they are stored in, indexing, hashing, data retrieval and hence influence the number of disk and block accesses. The cost estimation approach to query optimization compares different strategies based on their relative costs and selects the one that minimizes resource usage

**Heuristic approach query optimization**

Heuristic approach optimization transforms the query-tree by using a set of rules that typically (but not in all cases) improve execution performance: Perform selection early (reduces the number of tuples) Perform projection early (reduces the number of attributes) Perform most restrictive selection and join operations (i.e. with smallest result size) before other similar operations.

Below are the different query optimization heuristics: -

1. Cascade of selection
2. Commutativity of selection operations
3. In a sequence of projection operations, only the last sequence is required
4. Commutativity of Selection and Projection
5. Commutativity of Theta join (and Cartesian product)
6. Commutativity of Selection and Theta join (or Cartesian product)
7. Commutative projection and Theta join (or Cartesian product)
8. Commutativity of union and Intersection (but not set difference)
9. Commutativity of selection and set operations (union, intersection and set difference)
10. Commutativity of Projection and Union
11. Associativity of Theta join (and Cartesian product)
12. Associativity of Union and Intersection (but not Set difference)

**cost based query optimization**

A cost-based query optimizer works as follows: First, it generates all possible query execution plans. Next, the cost of each plan is estimated. Finally, based on the estimation, the plan with the lowest estimated cost is chosen. Since the decision is made using estimated cost values, the plan chosen may actually not be optimal. The quality of optimizer decisions depends on the complexity and accuracy of cost functions used. It includes different techniques such as use of dynamic programming for deciding the best plan. Its main drawback is that it is very costly. As a result most of the optimizers do not employ this strategy. A cost estimation technique is so that a cost may be assigned to each plan in the search space. Intuitively, this is an estimation of the resources needed for the execution of the plan.

Steps in cost-based query optimization include: -

1. Generating logically equivalent expressions using equivalence rules

2. Annotating resultant expressions to get alternative query plans

3. Choosing the cheapest plan based on estimated cost

Estimation of plan cost based on the statistical information about relations, statistics estimation for intermediate results to compute cost of complex expressions and cost formulae for algorithms, computed using statistics. The Types of searches using Search Keywords and Indices used here include: -

* A linear search looks down a list, one item at a time, without jumping.
* A binary search is when you start with the middle of a sorted list, and see whether that's greater than or less than the value you're looking for, which determines whether the value is in the first or second half of the list
* An equality search is a type of search filter that can be used to identify entries that contain a specific value for a given attribute. The server will use an equality matching rule to make the determination

The given cost-based approach techniques for query optimization include:

* 1. Linear search on a heap file that has no index
  2. Binary search on a sorted file that has no index
  3. Equality search on an attribute that has a hash index
  4. Equality search on an indexed primary key
  5. Inequality search on an indexed primary key
  6. Equality search on an indexed (clustered) non-primary key
  7. Equality search on an indexed (nonclustered) non-primary key
  8. Inequality search on an indexed (B+ Tree) non-primary key

**Objective**

The objective of this essay is to establish how different heuristics can be combined with each cost estimation technique. That is to combine each of the above formulae used to select the lowest cost of query and some rules that improve performance as depicted below: -

1. **Linear search on a heap file that has no index**, assuming that tuples are uniformly distributed into blocks, then on average half the blocks will be searched before the specific tuple is found can use the heuristic approach rule of **commutativity of selection and projection** where the attributes are projected and then selected.
2. **Binary search on a sorted file that has no index**
3. The cost of finding the first tuple that satisfies the predicate will be: [log2(nBlocks(R))] , SCA(R) will give us the average number of tuples expected to satisfy the predicate. These tuples will occupy [SCA(R)/bFactor(R)] blocks, such that the cost of searching the first block has already been catered for in [log2 (nBlocks(R))]. The cost estimate where more than 1 tuple satisfies the predicate will therefore be:[*log2****(nBlocks(R))]* +***[SCA(R)/bFactor(R)] - 1* **.**I propose that the use of Commutativity of Projection and Union
4. **Equality search on an attribute that has a hash index**

The hash function is applied on the attribute to determine the exact location of the target address if there is no overflow, then the estimated cost is 1 If there is an overflow, then additional cost will be incurred in order to traverse through the overflow.Therefore the heuristic I propose the use of **commutativity of selection operations** this is because it is specific .

1. **Equality search on an indexed primary key**

A clustered index can be used to retrieve the tuple that satisfies

the predicate if the predicate involves an equality condition on

the primary key the cost is therefore based on retrieving each level of the index

and retrieving the tuple itself:

*[nLevelsA(I) + 1]* Commutativity of selection and set operations (union, intersection and set difference) Commutativity of union and Intersection (but not set difference)

1. **Inequality search on an indexed primary key**

If the index is sorted, the target tuples can be found by

accessing all tuples before the target, assuming a uniform distribution, then on average half the blocks containing tuples will be searched before the specific tuple is

found the estimated cost would therefore be:

[*nLevelsA(I) + [nBlocks(R)/2]* Commutativity of Projection and Union

1. **Equality search on an indexed (clustered) non-primary key**

A clustered index that has been defined on a non-primary key attribute can be used to locate the attribute’s values SCA(R) will give us the average number of tuples expected to satisfy the predicate. These tuples will occupy [SCA (R)/bFactor(R)] blocks Therefore, the total estimated cost will be:

*[nLevelsA (I) + [SCA (R)/bFactor(R)]* Associativity of Theta join (and Cartesian product)

vii.)**Equality search on an indexed (nonclustered) non-primary key**

A non-clustered index that has been defined on a non-primary

key attribute can be used to locate the attribute’s values

The cost of accessing the index is nLevelsA (I), SCA(R) will give us the average number of tuples expected to satisfy the predicate. Given that it is a non-clustered index and assuming the tuples are on different blocks, the cost for this will be SCA(R) The total cost is therefore *[nLevelsA(I) + [SCA(R)]* therefore I propose that Commutativity of Selection and Theta join (or Cartesian product)

viii.) **Inequality search on an indexed (B+Tree)non-primary key**

Inequality conditions, on attribute A given condition x include:

A<x, A<=x, A>x, A>=x

This would involve a scan from the smallest value to x (for < or

<=) and from x up to the maximum value (for > or >=)

Subsequently resulting in half the leaf blocks being accessed,

and half the actual tuples being accessed. The cost of accessing the B+Tree index is also added. Therefore *[nLevelsA (I) + [nLfBlocksA (I)/2 + nTuples (R)/2*] Therefore I propose that