BACHELOR OF COMPUTER APPLICATIONS

SEMESTER 3

DCA2101

COMPUTER ORIENTED NUMERICAL METHODS

Unit 11

Query Processing and Optimization

Table of Contents

1. INTRODUCTION

In the previous unit, you have learned that normalization is needed to reduce redundancy, and relations are normalized so that when relations in a database are to be altered during the lifetime of the database, we do not lose information or introduce inconsistencies. In this unit, we learn about the query processing and optimization techniques used in Database management systems.

Query processing is a set of activities to obtain the desired information from a database system in a predictable and reliable fashion. These activities are

(i) parsing a query and translating it into the form such that it can be optimized (ii) Optimization of query data and (iii) finally evaluating for an execution plan over physical data model, using operations on file structures, indices, etc. In SQL, queries are expressed in high-level declarative form. So the query has to be processed and optimized so that the query of the internal form gets a suitable execution strategy for processing. This optimization helps get the result in a lesser time.

There can be several possible strategies for processing a query depending upon its complexity. One should select a good strategy for processing a query.

This unit will introduce you to the basic concepts of query processing process and query optimization strategies in the relational database domain.

1.1 Objectives

After studying this unit, you should be able to:

- *Process and optimize query*
- *Explain about the equivalence of expressions in queries*
- *Write basic algorithms for executing query operations*
- *Describe heuristics in query optimization*
- *Explain cost estimates in query optimization*
- *Discuss basic query optimization strategies*

2. QUERY INTERPRETATION

There are three phases that a query passes through during the DBMS processing of that query: (i) Parsing and translation (ii) Optimization

(iii) Evaluation. Figure 11.1 shows the steps in the query processing process.

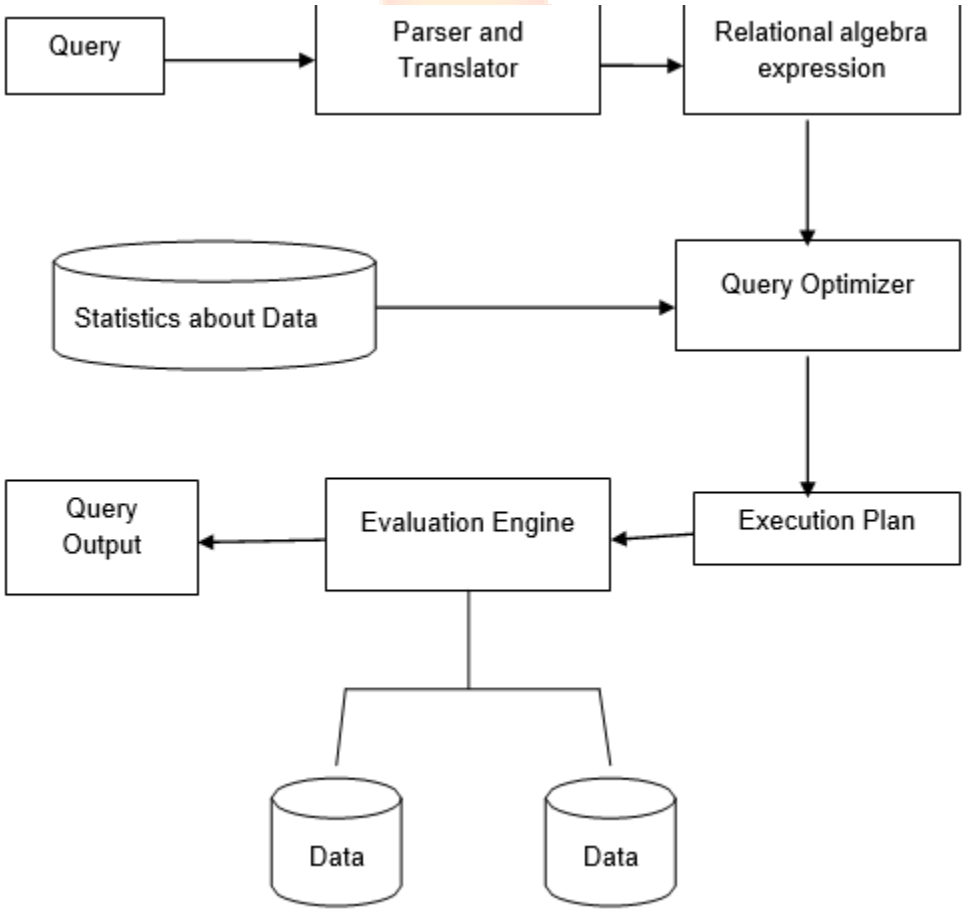

Figure 11.1: Steps in the query processing process

Parsing and translation are done because Query in High-Level language is suitable for human use only. For example in SQL, queries are expressed in high-level declarative form and a high-level relational query is generally non-procedural in nature. It simply informs about "what" rather than informing "how" to get a query. Internally a query should be represented in a more useful form, like relational algebra. So, first, the system must translate the query into

its internal form. After parsing and translating the query into a relational algebra expression, the query is then transformed into a form, usually a query tree or graph, which can be handled by the optimization engine (Query Optimizer).

After the query is been processed and translated into the form that can be optimized, the question arises why we need to optimize such a form of data. A very efficient methodology should be applied to find the result of the query in the existing database structure. If optimization is done then it can be queried using the information in the main memory, with little or no disk access. After optimization, various analyses are performed on the query data, thereby helping in giving solutions for valid evaluation plans for execution of the result. Execution of the query requires disk access. The problem again is that the transfer of data from disk is slow, relative to the speed of the main memory and the CPU. So it is better to think about saving the disk accesses also. Keeping all these limitations in mind, optimization is done to find out the best possible methods.

There are two main approaches to finding the best possible solutions. They are

(i) Rewriting the query in a more effective manner. (ii) Estimating the cost of various execution strategies for the query.

Usually, both strategies are combined because one has to solve it in a very less amount of time.

In the network and hierarchical systems, optimization is left for the most part to the application programmer. It is so because one has to know about the entire application program and it is not easy to transform a hierarchical or network query into another one.

When the optimization phase is over then the detailed strategy is observed and the best evaluation plan is found by the Evaluation engine for processing the query. Here we choose specific indices to use, and the order in which tuples are to be processed, etc.

The best evaluation plan is chosen out of multiple methods of executing a query and then executed to get the result.

Self-Assessment Questions - 1

1. Query is parsed and translated to get relational algebra expression. (True/False)
2. Optimization of query processing is done by _____ .

3. EQUIVALENCE OF EXPRESSIONS

It is always better to find out a query-processing strategy to find a relational algebra expression that is equivalent to the given query. It gives efficiency in execution.

An SQL query is first translated into an equivalent extended relational algebra expression–represented as a query tree data structure–that is then optimized. This can be done by first decomposing the SQL queries into query blocks. This basic unit can be translated into an extended relational algebra expression and then optimized. In fact, Nested queries are not query blocks, but are identified as separate query blocks.

A **query block** contains a **single** SELECT-FROM-WHERE expression (may contain GROUP BY and HAVING)

Let us look at the SQL query in the example of a University database on the FACULTY relation.

FACULTY (FID: string, FNAME: string, LNAME:string, DEPT: string, SALARY: real, ENO: int, DNO: int, SEX:string)

FACULTY	FID	FNAME	LNAME	DEPT	SALARY	DNO	ENO	SEX
	46	Mitra	Guru	IT	$88,000	2	76	M
	47	Lara	Maya	MBA	$88,999	3	85	F

Consider the following SQL query on the Faculty relation

```
SELECT
LNAME, FNAME
    FROM    FACULTY
    WHERE   SALARY >    (SELECT    MAX (SALARY)
                        FROM       FACULTY
                        WHERE      DNO=2);
```

This query includes a nested subquery and hence would be decomposed into two blocks. The inner block is

```
(SELECT     MAX (SALARY)
From        FACULTY
WHERE       DNO= 2);
```

And the outer block is

```
SELECT      LNAME, FNAME
FROM        FACULTY
WHERE       SALARY > c
```

Where **c** represents the result returned from the inner query block. The inner block can be translated into the extended relational algebra expression

δ<MAX SALARY> (σ<DNO=2> (FACULTY))

And the outer block into the expression

π <FNAME, LNAME> (σ<SALARY > c> (FACULTY))

After the SQL query is decomposed then the query optimizer chooses an execution plan for each block. It should be noted that in the above example, the inner block needs to be evaluated only once to produce the maximum salary, which is then used–as the constant c–by the outer block. This is known as an *uncorrelated nested query*. It is difficult to optimize the more *complex correlated nested queries*, where a tuple variable from the outer block appears in the WHERE-clause of the inner block.

Self-Assessment Questions - 2

3. Nested query blocks are not identified as separate query blocks. (True/False)
4. It is difficult to optimize the more complex correlated nested queries, where a tuple variable from the outer block appears in the _____of the inner block.

4. ALGORITHM FOR EXECUTING QUERY OPERATIONS

In this section, we will discuss the algorithm for executing query operations. They are (i) EXTERNAL sorting, (ii) SELECT operation, (iii) JOIN operation,

(iv) PROJECT & SET operation, (v) AGGREGATE operation, and (vi) Outer Join.

4.1 External Sorting

External Sorting is one of the Primary algorithms used in query processing that are suitable for large files of records stored on disk that do not fit entirely in main memory, as most database files do.

Whenever an SQL query specifies an ORDER BY clause, the query result must be sorted. Sorting is also a key component in sort-merge algorithms used for JOIN and other operations (such as UNION and INTERSECTION), and in duplicate elimination algorithms for the PROJECT operation (when an SQL query specifies the DISTINCT option in the SELECT clause).

The typical external sorting algorithm uses a ***sort-merge strategy***. This algorithm consists of two phases: (i) Sorting Phase (ii) Merging Phase.

This sort-merge algorithm also like other database algorithms requires *buffer* space in the main memory, where the actual sorting and merging of the **runs** (portions or pieces of the file) is performed.

In the **sorting phase, runs** (portions or pieces of the file) which can fit in the available buffer space are read into the main memory, which is sorted using an internal sorting algorithm, and written back to disk as temporary sorted subfiles (or runs). The size of a run and the **number of initial runs** are dictated by the **number of file blocks (b)** and the **available buffer space.**

In the **merging phase,** the sorted runs are merged during one or more **passes**. The **degree of merging** is the number of runs that can be merged together in each pass. In each pass, one buffer block is needed to hold one block from each of the runs being merged, and one block is needed for containing one block of the merge result.

4.2 SELECT Operation

This is another form of an algorithm for executing query operations. The execution of the SELECT operation depends on the file having specific access paths and may apply only to certain types of selection conditions.

Search methods for simple selection

There may be a number of search algorithms to select records from a file. These algorithms scan a file and are also known as **file scans** because they scan the records of a file to search for and retrieve records that satisfy a selection condition. A search algorithm may involve an ***index*** for searching and this type of index search is called an ***index scan***. The search methods given below are examples of some of the search algorithms that can be used to implement a select operation:

i) Linear search (brute force): This method is used to retrieve every record in the file, and test whether its attribute values satisfy the selection condition.
ii) Binary search: This method can be used if the selection condition involves an equality comparison on a key attribute on which the file is ordered.
iii) Using a primary index (or hash key): This method is used in the selection condition and involves an equality comparison on a **key attribute** with a primary index (or hash key).
iv) Using a primary index to retrieve multiple records: This method is used in the comparison condition is >, >=, <, or <= on a key field with a primary index.
v) Using a clustering index to retrieve multiple records: It is used in the selection condition that involves an equality comparison on a **non-key attribute** with a clustering index.
vi) Using a secondary (BPlus-tree) index on an equality comparison: This search method can be used to retrieve a single record if the indexing field is a **key** (has unique values) or to retrieve multiple records if the indexing field is **not a key**. This can also be used for comparisons involving >, >=, <, or <=.

Linear search (brute force) applies to any file, but all the other methods depend on having the appropriate access path on the attribute used in the selection condition. Binary search is more efficient than linear search if the selection condition involves an equality comparison

on a key attribute on which the file is ordered. The primary index, Clustering index, and Secondary index can be used to retrieve records in a *certain range*. Queries involving such conditions are called **range queries**.

4.3 Join Operation

The JOIN operation is one of the most time-consuming operations in query processing. There are many possible ways to implement a **two-way join**, which is a join on two files. Joins involving more than two files are called **multiway joins**. The number of possible ways to execute multiway joins grows very rapidly. Let us consider the algorithms for join operations of the form

P⋈A=B Q

For relations P and Q and where *A* and *B* are domain-compatible attributes of P and Q, respectively.

Methods for implementing Joins:

Following are the methods commonly used for implementing Joins:

i) *Nested-loop join (brute force):* In this method, for each record t in P (outer loop), every records from Q (inner loop) has to be retrieved and tested whether the two records satisfy the join condition t[A] = s[B].

ii) *Single-loop join (using an access structure to retrieve the matching records):* In this method, suppose if an index (or hash key) exists for one of the two join attributes, B of Q then each record t in P is retrieved one at a time (single loop), and then the access structure is used to retrieve all matching records s directly from Q that satisfy s[B] = t[A].

iii) *Sort–merge join:* In this method, the records of P and Q has to be *physically sorted (ordered)* by the value of the join attributes A and B, respectively. Then both files are scanned concurrently in order of the join attributes, matching the records that have the same values for A and B. If the files are not sorted, they may be sorted first by using external sorting. Pairs of file blocks are copied into memory buffers in order and the

records of each file are scanned only once each for matching with the other file–unless both A and B are non-key attributes. In such a case, the method needs to be modified slightly. We use P(i) to refer to the ith record in P. A variation of the sort-merge join can be used when secondary indexes exist on both join attributes. The indexes provide the ability to access (scan) the records in order of the join attributes, but the records themselves are physically scattered all over the file blocks. This method may be quite inefficient if every record access involves accessing a different disk block.

iv) *Hash-join:* In this method, the records of files P and Q are both hashed to the same hash file, using the same hashing function on the join attributes A of P and B of Q as hash keys. First, it passes through the first phase known as the **partitioning phase**, a single pass through the file with fewer records (say, P) hashes its records to the hash file buckets. It is called the **partitioning phase** because the records of P are partitioned into the hash buckets. In the second phase, called the **probing phase**, a single pass through the other file (Q) then hashes each of its records to probe the appropriate bucket, and that record is combined with all matching records from P in that bucket. Here we assume that the smaller of the two files fit entirely into memory buckets after the first phase although this type of assumption is not always required.

4.4 PROJECT And Set Operations

Project and set operations are very useful and used in certain cases for query optimization.

PROJECT operation: PROJECT operation $\pi_{<\text{attribute list}>}(R)$ can be implemented if <attribute list> includes a key of relation *P* because in this case, the result of the operation will have the same number of tuples as *P*, but with only the values for the attributes in <attribute list> in each tuple. If <attribute list> does not include a key of P, *duplicate tuples must be eliminated*. This is usually done by sorting the result of the operation and then eliminating duplicate tuples, which appear consecutively after sorting.

Hashing can also be used to eliminate duplicates. Each record is hashed and inserted into a bucket of the hash file in memory and it is checked against those already in the bucket. If it is a duplicate, it is not inserted

Set operations: Set operations are UNION, INTERSECTION, SET DIFFERENCE, and CARTESIAN PRODUCT. But the CARTESIAN

PRODUCT operation like *R x S* is quite expensive because its result includes a record for each combination of records from *R* and *S.* Moreover the attributes of the result include all attributes of R and S. *Hence CARTESIAN PRODUCT* operation is avoided and other equivalent operations are used for query optimizations.

The other three set operations UNION, INTERSECTION, and SET DIFFERENCE apply only to union-compatible relations, which have the same number of attributes and the same attribute domains.

Hashing can also be used to implement *UNION, INTERSECTION, and SET DIFFERENCE*.

4.5 Aggregate Operations

We use aggregate operations to find out the aggregate values for query optimizations. *MIN, MAX, COUNT, AVERAGE,* and *SUM* are the aggregate operators used to compute the aggregate values. But these can be computed by a table scan or by using an appropriate index. For example:

```
SELECT      MAX (SALARY)
FROM        FACULTY
```

If an ascending index on salary exists for the *FACULTY* relation, it can be used (otherwise we can scan the entire table).

The index can also be used for the *COUNT, AVERAGE*, and *SUM* aggregates but the index must be *dense*, that is, there must be an index entry for every record in the main file.

When a *GROUP BY* clause is used in a query, the aggregate operator must be applied separately to each group of tuples.

In order to do this, the table is first partitioned into subsets of tuples, where each partition (group) has the same value for the grouping attributes. In this case, the computation is more complex. Let us consider the following query:

SELECT	DNO, **AVG** (SALARY)
FROM	FACULTY
GROUP BY	DNO:

Usually either sorting or hashing is used on the grouping attributes to partition the file into the appropriate groups and then the algorithm computes the aggregate function for the tuples in each group which have the same grouping attribute values.

4.6 Outer Join

Outer join also can be used for query optimizations. Outer Join can be computed by modifying one of the join algorithms, such as nested-loop join or single-loop join. The sort-merge and hash-join algorithms can also be extended to compute outer joins.

Outer join can also be computed by executing a combination of relational algebra operators. In this case, the cost of the outer join would be the sum of the costs of the associated steps i.e. inner join, projections, and union.

Self-Assessment Questions - 3

5. Sort-merge algorithm requires *buffer space* in _____ memory, where the actual sorting and merging of the runs is performed.
6. The condition where Primary index, Clustering index and Secondary index are used to retrieve records in a certain range are called _____queries.
7. The JOIN operation is a very fast operation in query processing. (True/False).
8. In set operation, which operations is avoided. (Choose correct option)
   a) UNION
   b) INTERSECTION
   c) SET DIFFERENCE
   d) CARTESIAN PRODUCT

5. HEURISTICS IN QUERY OPTIMIZATION

The heuristic rule is applied for Query Optimization by modifying the internal representation of the query. This form of query is generally in the form of query tree or a query graph data structure. Although some optimization techniques were based on query graphs, nowadays, this technique is not applied because query graphs cannot show the order of operation which is needed by the query optimizer for query execution. So this unit will deal mainly with the Heuristic Optimization of the query tree.

A heuristic rule is applied to the initial query expression and produces the heuristically transformed equivalent query expressions. This is performed by transforming an initial expression (tree) into an equivalent expression (tree) which is made more efficient for execution. This rule works well in most cases but is not always guaranteed.

Execution of the query tree consists of executing an internal node operation whenever its operands are available and then replacing that internal node with the relation that results from executing the operation.

Query trees and query graphs

The query tree is a tree data structure that represents the relational algebra expression in the query optimization process. The leaf nodes in the query tree correspond to the input relations of the query. The internal nodes represent the relational algebra operations. The system will execute an internal node operation whenever its operands are available and then the internal node is replaced by the relation which is obtained from executing the operation. The execution is terminated when the root node is executed and produces the result relation for the query.

For example – Consider the following two tables with the following schema:

Emp(EmpID, Name, Sal, DeptID)

Dept(DeptID, Dname, Location)

To find the name of employees working in the Marketing department, a query in relational algebra can be written as shown in Q as well as a query tree as shown in fig. 11.2.

Q: ΠE_{name} ($\sigma_{Dname = "Marketing"}$ (Emp $\bowtie$ Dept))

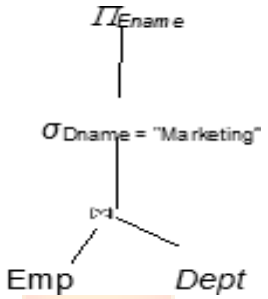

Figure 11.2: Query Tree for Q

The query can also be represented by a **query graph**. In this case, the relations in the query are represented by relational nodes and are represented by single circles. Constant nodes are used to represent constant values and are displayed as double circles or ovals. Selection and join conditions are represented by the graph edges. And finally, the attributes to be retrieved from each relation are displayed in square brackets above each relation. The graph query representation does not give an order of performing the operations because there is only a single graph corresponding to each query.

Hence query trees are better than query graphs because the query optimizer needs to show the order of operations for query execution which is not possible in query graphs.

Heuristic optimization of query trees:

Two relational algebra expressions are said to be equivalent if the two expressions generate two relations of the same set of attributes and contain the same set of tuples although their attributes may be ordered differently.

Hence while doing Heuristic Optimization on Query Trees, generally, many different relational algebra expressions can be found which can be equivalent to correspond to the same query. And for every relational algebra expression a query tree can be drawn. And hence there can be many different query trees to correspond to the same query.

The query parser generates a standard *initial query* tree to correspond to the SQL query without optimizing. When the simple standard form of the query tree is found out then the heuristic query optimizer transforms this initial query tree into a *final query tree* that is efficient to execute.

A general outline of a Heuristic Algebraic Optimization Algorithm

Heuristic rule are generally applied as per the following steps:

1) First of all *SELECT* operations are broken up with conjunctive operations into a cascade of *SELECT* operations.
2) Then *SELECT* operations are moved down far to the query tree as is permitted by the attributes involved in the select condition.
3) Leaf nodes of the tree are rearranged by :
   - positioning the leaf node relation with the most restrictive SELECT operations so they are executed first in the query representation,
   - and making sure that the ordering of leaf nodes does not cause CARTESIAN PRODUCT operation.
4) *CARTESIAN PRODUCT* operations are combined with a subsequent *SELECT* operation in the tree into a *JOIN* operation if the condition represents a join condition.
5) Lists of projection attributes are broken down and moved down the tree as far as possible by creating new *PROJECT* operations as needed.
6) And lastly subtrees are identified that represent groups of operations that can be executed by a single algorithm.

Using the above rules the optimized query Q is shown in fig 11.3. It is optimized as now we need to transfer less data from the Dept table for performing joining with data of Emp table.

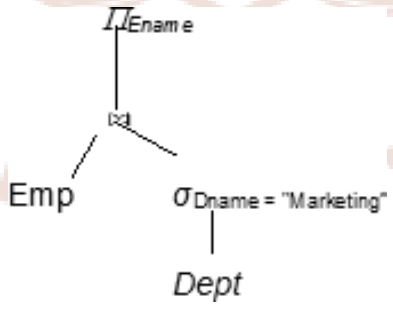

Figure 11.3: Optimized Query Tree for Q

These steps are applied using general transformation rules for relational algebra operations into equivalent ones. While transforming, the meaning of the operations and the resulting relations should not mismatch.

Self-Assessment Questions - 4

9. In query graph Constant nodes are used to represent constant values and are displayed as double circles or_____ .
10. The query parser generates a standard *initial query* tree to correspond to SQL query without _____ . (Choose correct option)
    a) optimizing
    b) parsing
    c) evaluation
    d) sorting

6. SEMANTIC QUERY OPTIMIZATION

This is one of the alternative approaches to query optimization and uses constraints specified on the database scheme.

Suppose there is a query which says to retrieve the names of all students in a University whose age is more than their faculty. And if we had a constraint on the database schema that stated no student is older than faculty. In such a case if the semantic query optimizer checks for the existence of this constraint then it need not to execute the query at all. But searching through many constraints to find constraints applicable to a given query can be quite time-consuming.

7. CONVERTING QUERY TREE TO QUERY EVALUATION PLAN

We have seen above that query optimizers use the equivalence rules to generate a logically equivalent expression to the given query expression such that it is easier for evaluation. Equivalent expression helps in optimizing process easily. And we also observed that the evaluation plan must have a detailed algorithm for each operation in the expression so that the execution of the operations is coordinated one after another.

We saw that the output of the Parsing and Translating step in the query processing is a relational algebra expression. If the query is complex, then the expressions consist of several multiple operations and interact with various relations. The evaluation of the expression becomes very costly in terms of both time and memory space in such a case. So it is very important to consider how to evaluate an expression containing multiple operations. The two approaches used for evaluating expression are materialization and pipelining. Depending upon the situation sometimes Materialization is applicable and sometimes Pipelining is applicable.

Materialization

In the materialization approach of query evaluation, we start from the lowest-level operations in the expression. Using materialized evaluation, every immediate operation produces a temporary relation which is used for the evaluation of higher-level operations. Those temporary relations vary in size and might have to be stored in the disk. Hence if materialization is applicable, the cost for this evaluation is the sum of the costs of all operations plus the cost of writing/reading the result of intermediate results to disk.

Pipelining

Pipelining of different operations can improve query evaluation efficiency by reducing the number of temporary relations that are produced. To achieve this reduction, we can combine several operations into a pipeline of operations. We can implement the pipeline by creating a query execution code. Depending upon different situations, we can use pipelining to reduce the number of temporary files, thus reducing the cost of query evaluation.

Self-Assessment Questions - 5

11. If the query is complex, then the expressions consist of several multiple operations and interact with various _____ .
12. We can combine several operations into a pipeline of operations to improve _____ evaluation.

8. COST ESTIMATES IN QUERY OPTIMIZATION

The strategy of the query optimizer depends on estimation and comparison of the cost of executing different strategies for the query. Query Optimizer should be able to find out the lowest cost estimates. After considering all the strategies, one has to choose the query execution plans with the lowest cost and least execution time. So accurate cost estimates are required which can be compared fairly and realistically.

There are different components involved in the cost of query execution and different types of information needed in cost functions. This information is kept in the DBMS catalog.

8.1 Measure Of Query Cost

The cost of executing a query includes the following components:

i) **Access cost to secondary storage:** This is the cost measured by the search performed while reading, and writing data blocks that reside on secondary storage, mainly on disk. The cost of searching for records in a file depends on the type of access structures on that file, such as ordering, hashing, and primary or secondary indexes. In addition, access cost is affected by the way file blocks are allocated contiguously on the same disk cylinder or scattered on the disk. This cost is usually more important in the case of a large database since disk accesses are slow compared to in-memory operations.

ii) **Storage cost:** This is the cost of storing any intermediate files that are generated by an execution strategy for the query.

iii) **Computation cost:** This is the cost measured by the performance of in-memory operations on the data buffers during query execution. Such operations include searching for and sorting records, merging records for a join, and performing computations on field values. This cost is important when the database is small where almost all data reside in the memory.

iv) **Memory usage cost:** This is the cost pertaining to the number of memory buffers needed during query execution.

v) **Communication cost:** This is the cost of shipping the query and its results from the database site to the site or terminal where the query originated. In the distributed system, this cost should be minimized.

There can be other factors also that may be the cost components in a cost function which may not be so much of importance. Therefore, some cost functions consider only disk access cost as the reasonable measure of the cost of a query-evaluation plan.

8.2 Catalog Information For Cost Estimation Of Queries

As already mentioned, Query optimizers use the statistic information stored in the DBMS catalog to estimate the cost of a plan. The relevant catalog information about the relation includes:

- Number of tuples in a relation r; denoted by nr
- Number of blocks containing tuple of relation r: denoted by br
- Size of the tuple in a relation r (assume records in a file are all of same types): denoted by sr
- Blocking factor of relation r which is the number of tuples that fit into one block: denoted by fr
- V(A,r) is the number of distinct value of an attribute A in a relation r. This value is the same as size of $\pi_A(r)$. If A is a key attribute then V(A,r) = nr
- SC(A,r) is the selection cardinality of attribute A of relation r. This is the average number of records that satisfy an equality condition on attribute A.

Moreover some information about indices is also used along with the relation information. They are:

- A number of levels in index i.
- Number of lowest–level index blocks in index i (number of blocks in the leaf level of the index)

The above given statistical information are the simplified one. In a real database management system, the optimizer may have more information to improve the accuracy of their cost estimates.

This statistical information maintained in the DBMS catalog along with the measures of query cost based on the number of disk accesses helps in estimating the cost for different relational algebra operations

Self-Assessment Questions - 6

13. Memory usage cost is the cost pertaining to the number of _____ buffers needed during query execution.
14. Query optimizers use the statistic information stored in _____ catalog to estimate the cost of a plan.

9. JOIN STRATEGIES FOR PARALLEL PROCESSING

We can use multiple processors for parallel computation to make the processing faster. There are many cases where multiple processors may be available for parallel computation of the join. The architecture may be different, including database machines. We will consider an architecture where all processors have access to all disks, and all processors share main memory.

9.1 Parallel Join

In Parallel-join, pairs to be tested are split over several processors. Each processor computes part of the join, and then the results are assembled (merged).

Ideally, the overall work of computing join is partitioned evenly over all processors. If such a split is achieved without any overhead, a parallel join using N processors will take $1/N$ times as long as the same join would take on a single processor.

The speedups between several processors are more or less similar because the overhead is incurred in partitioning the work among the processors and in collecting the results computed by each processor. If the split is not even then the final result cannot be obtained until the last processor has finished. Speedup also depends upon the processors competing for shared system resources. In such a case for e.g., for relation A and B where A ⋈ B, if each processor uses its own partition of A, and the main memory cannot hold the entire B, the processors need to synchronize the access of B so as to reduce the number of times that each block of B must be read in from the disk.

A parallel hash algorithm is used to reduce memory contention.

9.2 Pipelined Multiway Join

Multiway Join can be pipelined if several joins are computed in parallel over other processors. For example:

1) $r_1 \bowtie r_2 \bowtie r_3 \bowtie r_4$ can be computed by first computing " $t_1 \leftarrow r_1 \bowtie r_2$ " and "$t_2 \leftarrow r_3 \bowtie r_4$", and then " $t_1 \bowtie t_2$ "

2) And, it can be computed in the pipelined way: For $r_1 \bowtie r_2 \bowtie r_3 \bowtie r_4$ One Processor (say P1) can be assigned to process $r_1 \bowtie r_2$, another processor (say P2) to process $r_3 \bowtie r_4$, and other processors (say P3) to process the join of the tuples being generated by P1 and P2.

9.3 Physical Organization

We can also organize the physical resources to make the strategies for query evaluation more efficient and better.

The database can be partitioned over several disks for accessing in parallel to avoid contention between several processors. We must also distribute data among disks to exploit parallel disk access.

For the parallel 2-way join, tuples of individual relations can be split among several disks. This phenomenon is known as *disk striping*. For example in the hash-join algorithm, we can assign tuples to disks based on the hash function value where all groups of tuples that share a bucket are assigned to the same disk. Either each group can be assigned to the same disk, if possible, or the groups can be distributed uniformly among the available disks in order to exploit parallel disk access.

For the pipeline-join, it is better to keep each relation on one disk and the distinct relations be assigned to separate disks to the degree possible.

Although physical organization can be optimized differently for different queries, one must organize the physical resources for a better expectation of result. If the physical organization is done in a proper way then only the query optimizer can choose the best technique by estimating the cost of each technique on the given physical organization.

Self-Assessment Questions - 7

15. In Parallel-join pairs to be tested are split over several processors. (True/False)
16. _____ can be pipelined if several joins are computed in parallel.

10. SUMMARY

When the queries are expressed in the high-level declarative form then the queries have to be processed and optimized so that the query of the internal form gets a suitable execution strategy for processing. To obtain this, several query processing and optimization strategies have to be performed.

Query processing and optimization is a set of activities to obtain the desired information from a database system such that the result is obtained in lesser time with low cost. The queries are parsed and translated in equivalent relational algebra expression which is then easier for optimizing it using different rules and algorithms.

We use External Sorting, Select operation, Join Operation, Project and Set Operation, Aggregate Operations, and Outer Join for executing query operations. We also do Heuristic Optimization of Query Trees or semantic optimization where ever appropriate and then the optimized results are obtained for the query evaluation plan. We convert query trees into a query evaluation plan.

After the query evaluation plan, the cost is estimated to find out the lowest cost possible. Costs are estimated using several components that are included while executing plans and statistical information stored in the DBMS catalog. Finally, the plan is executed over a physical data model, using operations on file structures, indices, etc.

11. TERMINAL QUESTIONS

1. Draw and explain the architecture of Query Processing.
2. Explain sort-merge strategy in external sorting.
3. What is Heuristic Optimization?
4. Explain the measures of query cost in Query Optimization?
5. Explain Join strategies for parallel processing.

12. ANSWERS

Self Assessment Questions

1. True
2. Query optimizer
3. False
4. WHERE clause
5. Main
6. Range
7. False
8. d)-CARTESIAN PRODUCT.
9. ovals
10. a)-optimizing.
11. Relations
12. Query
13. Memory
14. DBMS
15. True
16. Multiway join

Terminal Questions

1. There are three phases that a query passes through during the DBMS processing of that query: (i) Parsing and translation (ii) Optimization
   (iii) Evaluation. (Refer section 2 for detail)
2. A sort-merge strategy in external sorting consists of two phases: Sorting Phase (ii) Merging Phase. In the sorting phase, runs (portions or pieces of the file) which can fit in the available buffer space are read into main memory, which is sorted using an internal sorting algorithm, and written back to disk as temporarily sorted subfiles (or runs). In the merging phase, the sorted runs are merged during one or more passes. (Refer section 4.1 for detail)

3. The heuristic rule is applied for Query Optimization by modifying the internal representation of the query. A heuristic rule is applied to the initial query expression and produces the heuristically transformed equivalent query expressions. This is performed by transforming an initial expression (tree) into an equivalent expression (tree) which is made more efficient for execution. This rule works well in most cases but not always guaranteed. (Refer section 5 for detail)
4. The measure of query cost includes the components like access cost to secondary storage, storage cost, computation cost, memory usage cost, and communication cost. (Refer section 8.1 for detail)
5. Join strategies for parallel processing are Parallel join, pipelined multiway join along with the way physical resources are organized. (Refer section 9 for detail)