DISTRIBUTED DATABASES (R22CSE3146)

III B.Tech I Semester (Information Technology)

UNIT – II

Query processing and decomposition: Query processing objectives, characterization of query processors, layers of query processing, query decomposition, localization of distributed data.
Distributed query Optimization: Query optimization, centralized query optimization, distributed query optimization algorithms.

Query : A "Query " refers to the action of retrieving data from the database. Query is expressed by using high level language. (or)

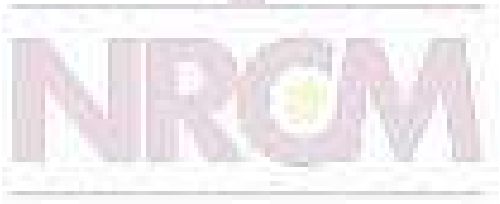
A query is a request for data or information from a database table or combination of tables. In the context of queries in a database, it can be either a select query or an action query. A select query is a data retrieval query, while an action query asks for additional operations on the data, such as insertion, updating or deletion. Most formal queries are written in SQL (Structured Query Language).

A query can either be a request for data results from your database or for action on the data, or for both. A query can give you an answer to a simple question, perform calculations, combine data from different tables, add, change, or delete data from a database.

Query Processor : A Query Processor is a module in the DBMS that performs the tasks to process, to optimize and to generate execution strategy for a high level query.

Query Processing : The process of extracting data from a database is called query processing. It requires several steps to retrieve the data from the database during query processing. The actions involved actions are:

1. Parsing and translation
2. Optimization
3. Evaluation



Distributed Query Processing

- A query is a request for data that involves transmitting data between computers in a network. The process of answering queries on data stored at multiple sites in a computer network is called distributed query processing.
- Distributed query processing is the procedure of answering queries (which means mainly read operations on large data sets) in a distributed environment where data is managed at multiple

sites in a computer network.

- Query processing involves the transformation of a high-level query (e.g., formulated in SQL) into a query execution plan (consisting of lower-level query operators in some variation of relational algebra) as well as the execution of this plan.
- The goal of the transformation is to produce a plan which is equivalent to the original query (returning the same result) and efficient, i.e., to minimize resource consumption like total costs or response time.

- Query Processing is a translation of high-level queries into low-level expression.
- It is a step wise process that can be used at the physical level of the file system, query optimization and actual execution of the query to get the result.
- It requires the basic concepts of relational algebra and file structure.
- It refers to the range of activities that are involved in extracting data from the database.
- It includes translation of queries in high-level database languages into expressions that can be implemented at the physical level of the file system.
- In query processing, we will actually understand how these queries are processed and how they ae optimized.

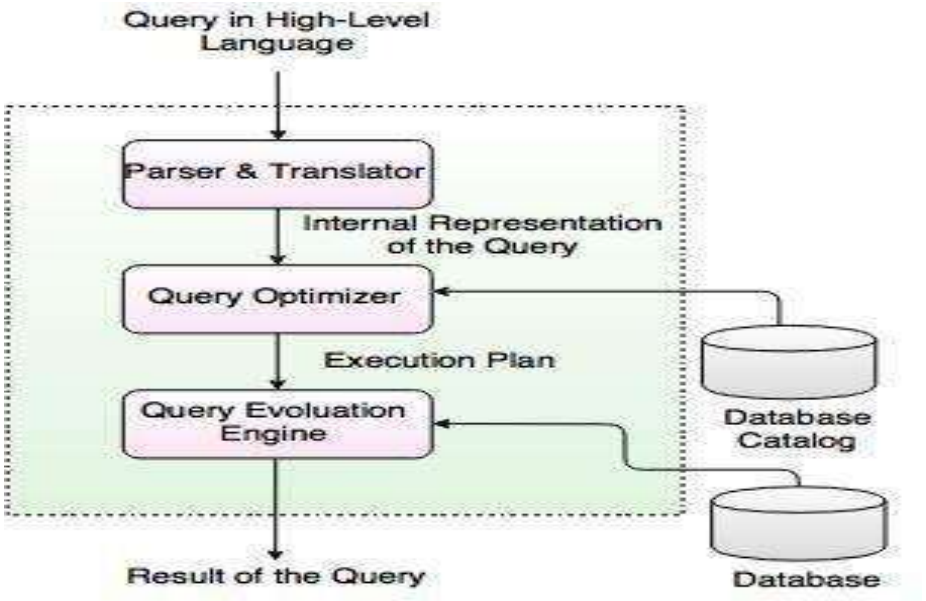


Fig. Query Processing

In the above diagram,

- The first step is to transform the query into a standard form.
- A query is translated into SQL and into a relational algebraic expression. During this process,

Parser checks the syntax and verifies the relations and the attributes which are used in the query.

- The second step is Query Optimizer. In this, it transforms the query into equivalent expressions that are more efficient to execute.
- The third step is Query evaluation. It executes the above query execution plan and returns the result.

Decomposition

- Decomposition is the process of breaking up a single relation into two or more sub-relations.
- This process can help remove redundancy, inconsistencies, and anomalies from a database.
- Decomposition is also dependency preserving and lossless.
- In distributed query processing, decomposition is one of the functions performed by the first three layers of the process.
- These layers map the input query into an optimized distributed query execution plan.
- In query decomposition, a distributed calculus query is mapped into an algebraic query on global relations.

Query Processing Objectives

- The objective of query processing in a distributed context is to transform a high-level query on a distributed database, which is seen as a single database by the users, into an efficient execution strategy expressed in a low-level language on local databases.
- We assume that the high-level language is relational calculus, while the low-level language is an extension of relational algebra with communication operators.
- An important aspect of query processing is query optimization. Because many execution strategies are correct transformations of the same high-level query, the one that optimizes (minimizes) resource consumption should be retained.
- A good measure of resource consumption is the ***total cost***. *The* Total cost is the sum of all times incurred in processing the operators of the query at various sites and in inter site communication.
- Another good measure is the ***response time*** of the query, which is the time elapsed for executing the query. Since operators can be executed in parallel at different sites, the response time of a query may be significantly less than its total cost.
- In a distributed database system, the total cost to be minimized includes : 1. CPU, 2. I/O and 3. Communication costs.

1. **The CPU cost** is incurred when performing operatorson data in main memory.
2. **The I/O cost** is the time necessary for disk accesses. This cost can be minimized by reducing the number of disk accesses through fast accessmethods to the data and efficient use of main memory (buffer management).
3. **The communication cost** is the time needed for exchanging data between sites participated in the execution of the query. This cost is incurred in processing the messages (formatting/de-

formatting), and in transmitting the data on the communication network.

Characterization of Query Processors

- It is quite difficult to evaluate and compare query processors in the context of both centralized systems and distributed systems because they may differ in many aspects.
- Here are some important characteristics of query processors that can be used as a basis for comparison.
  1. Languages
  2. Types of Optimization
  3. Optimization Timing
  4. Statistics
  5. Decision Sites
  6. Exploitation of the Network Topology
  7. Exploitation of Replicated Fragments
  8. Use of Semijoins

  - The first four characteristics hold for both centralized and distributed query processors.
- The next four characteristics are particular to distributed query processors in tightly-integrated distributed DBMSs.

1. Languages

Query processing must perform efficient mapping from the input language to the output language.

2. Types of Optimization

- Query optimization aims at choosing the "best" point in the solution space of all possible execution strategies.
- An immediate method for query optimization is to search the solution space, exhaustively predict the cost of each strategy, and select the strategy with minimum cost.
- The problem is that the solution space can be large; that is, there may be many equivalent strategies, even with a small number of relations.
- Therefore, an "exhaustive" search approach is often used whereby (almost) all possible execution strategies are considered.

3. Optimization Timing

- A query may be optimized at different times relative to the actual time of query execution.
- Optimization can be done *statically* before executing the query or *dynamically* as the query is executed.
- Static query optimization is done at query compilation time.
- Dynamic query optimization proceeds at query execution time.

- The main advantage over static query optimization is that the actual sizes of intermediate relations are available to the query processor, thereby minimizing the probability of a bad choice.

4. Statistics

- The effectiveness of query optimization relies on *statistics* on the database.
- Dynamic query optimization requires statistics in order to choose which operators shouldbe done first.
- Static query optimization is even more demanding since the size of intermediate relations must also be estimated based on statistical information.

5. Decision Sites

When static optimization is used, either a single site or several sites may participate in the selection of the strategy to be applied for answering the query.

- Most systems use the centralized decision approach, in which a single site generates the strategy.
- However, the decision process could be distributed among various sites participating in the elaboration of the best strategy.
- The centralized approach is simpler but requires knowledge of the entire distributed database, while the distributed approach requires only local information.
- Hybrid approaches where one site makes the major decisions and other sites can make local decisions are also frequent.

6. Exploitation of the Network Topology

- The network topology is generally exploited by the distributed query processor.
- With wide area networks (WAN), the cost function to be minimized can be restricted to the data communication cost, which is considered to be the dominant factor.
- With local area networks (LAN), communication costs are comparable to I/O costs. Therefore, it is reasonable for the distributed query processor to increase parallel execution at the expense of communication cost.

7. Exploitation of Replicated Fragments

- A distributed relation is usually divided into relation fragments.
- Distributed queries expressed on global relations are mapped into queries on physical fragments of relations by translating relations into fragments.
- We call this process *localization* because its main function is to localize the data involved in the query

8. Use of Semijoins

- The semijoin operator has the important property of reducing the size of the operandrelation.
- A semijoin is particularly useful for improving the processing of distributedjoin operators as it reduces the size of data exchanged between sites.
- The early distributed DBMSs, such as SDD-1 which were designed for slow wide area networks, make extensive use of semijoins. Some later systems, such as R*, assume faster networks and do not employ semijoins.

Layers of Query Processing

- There are Four main layers are involved in distributedquery processing.

1. Query Decomposition
2. Data Localization
3. Global Query Optimization
4. Distributed Query Execution

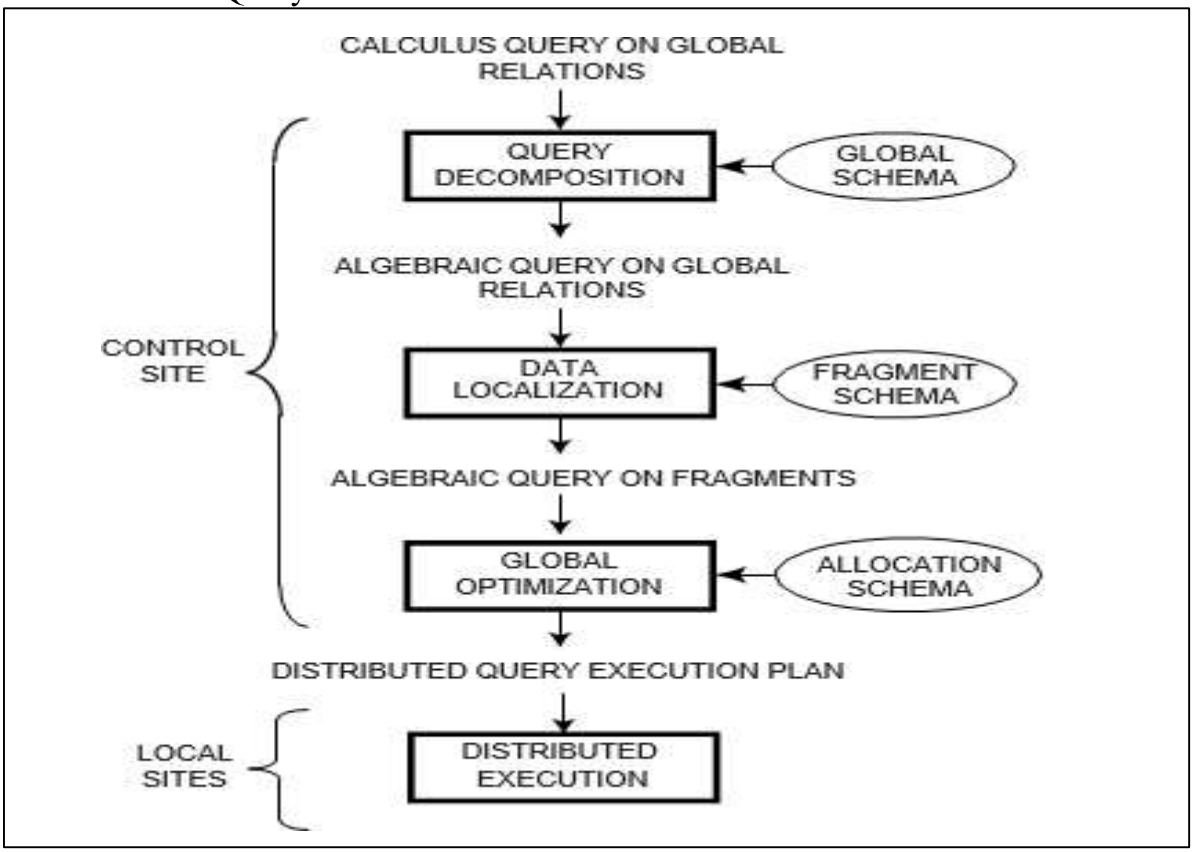


Fig : Generic Layering Scheme for Distributed Query Processing

- The input is a query on global data expressed in relational calculus.

- This query is posed on global (distributed) relations,meaning that data distribution is hidden.
- Four main layers are involved in distributed query processing.
- The first three layers map the input query into an optimized distributed query execution plan.
- They perform the functions of *query decomposition*, *data localization*, and *global query optimization*.
- Query decomposition and data localization correspond to query rewriting.
- The fourth layer performs *distributed query execution* by executing the plan and returnsthe answer to the query.
- It is done by the local sites and the control site.

1. Query Decomposition

- The first layer decomposes the calculus query into an algebraic query on global relations.
- The information needed for this transformation is found in the global conceptual schema describing the global relations.
- Query decomposition can be viewed as four successive steps.

i. Normalization
ii. Analyzation
iii. *Simplified (*eliminate redundant)
iv. *Restructured / rewriting*

Query decomposition is the first phase of query processing that transforms a relational calculus query into a relational algebra query. Both input and output queries refer to global relations, without knowledge of the distribution of data. Therefore, query decomposition is the same for centralized and distributed systems.

i. Normalization

- It is the goal of normalization to transform the query to a normalized form to facilitate further processing.
- There are two possible normal forms for the predicate,
- The *conjunctive normal form* is a conjunction (∧ predicate)
- The *disjunctive normal form* is a disjunctions (∨ predicates)
- one giving precedence to the AND (∧) and the other to the OR (∨).

$$(p_{11} \vee p_{12} \vee \cdots \vee p_{1n}) \wedge \cdots \wedge (p_{m1} \vee p_{m2} \vee \cdots \vee p_{mn})$$

where p_{ij} is a simple predicate. A qualification in *disjunctive normal form*, on the other hand, is as follows:

$$(p_{11} \wedge p_{12} \wedge \cdots \wedge p_{1n}) \vee \cdots \vee (p_{m1} \wedge p_{m2} \wedge \cdots \wedge p_{mn})$$

The transformation of the quantifier-free predicate is straightforward using thewell-known equivalence rules for logical operations (∧, ∨, and ¬):

1. $p_1 \wedge p_2 \Leftrightarrow p_2 \wedge p_1$

2. $p_1 \vee p_2 \Leftrightarrow p_2 \vee p_1$
3. $\neg(p_1 \vee p_2) \Leftrightarrow \neg p_1 \wedge \neg p_2$

Example: Let us consider the following query on the engineering database that we have been referring to: "Find the names of employees who have been working on project P1 for 12 or 24 months"

The query expressed in SQL is

```
SELECT ENAME FROM    EMP, ASG
WHERE  EMP.ENO = ASG.ENO
AND    ASG.PNO = "P1"
AND    DUR = 12 OR DUR = 24
```

The qualification in conjunctive normal form is

$$EMP.ENO = ASG.ENO \wedge ASG.PNO = \text{“P1”} \wedge (DUR = 12 \vee DUR = 24)$$

while the qualification in disjunctive normal form is

$$(EMP.ENO = ASG.ENO \wedge ASG.PNO = \text{“P1”} \wedge DUR = 12) \vee$$
$$(EMP.ENO = ASG.ENO \wedge ASG.PNO = \text{“P1”} \wedge DUR = 24)$$

In the latter form, treating the two conjunctions independently may lead to redundant work if common subexpressions are not eliminated.

ii. Analysis

- Query analysis enables rejection of normalized queries for which further processing is either impossible or unnecessary. The main reasons for rejection are that the query is *type incorrect* or *semantically incorrect.*
- When one of these cases is detected, the query is simply returned to the user with an explanation. Otherwise, query processing is continued.

Example: The following SQL query on the engineering database is type incorrect for two reasons.

```
SELECT E# FROM    EMP   WHERE  ENAME > 200
```

First, attribute E# is not declared in the schema.

Second, the operation ">200" is incompatible with the type string of ENAME.

iii. Elimination of Redundancy

- Here, the query should be simplified means eliminating the redundant predicates.

Example :

```
SELECT TITLE FROM EMP

WHERE  (NOT (TITLE = "Programmer")
AND (TITLE = "Programmer"  OR    TITLE = "Elect. Eng.")
AND NOT (TITLE = "Elect. Eng.")) OR ENAME = "J. Doe"
```

can be simplified using the previous rules to become

```
SELECT TITLEFROM  EMP  WHERE  ENAME = "J. Doe"
```

The simplification proceeds as follows. Let p_1 be TITLE = "Programmer", p_2 beTITLE = "Elect. Eng.", and p_3 be ENAME = "J. Doe". The query qualification is

$$(\neg p_1 \wedge (p_1 \vee p_2) \wedge \neg p_2) \vee p_3$$

The disjunctive normal form for this qualification is

$$(\neg p_1 \wedge ((p_1 \wedge \neg p_2) \vee (p_2 \wedge \neg p_2))) \vee p_3$$

iv. Rewriting

- The last step of query decomposition rewrites the query in relational algebra.
- To represent the relational algebra query graphically by an *operator tree*.
- An operator tree is a tree in which a leaf node is a relation stored in the database, and a non-leaf node is an intermediate relation produced by a relational algebra operator. The sequence of operations is directed from the leaves to the root, which represents the answer to the query.

Example : The query "Find the names of employees other than J. Doe who worked on the CAD/CAM project for either one or two years" whose SQL expression is

```
SELECT  ENAME
FROM        PROJ,     ASG,     EMP     WHERE
ASG.ENO = EMP.ENO
AND  ASG.PNO = PROJ.PNO
AND  ENAME!= "J. Doe"
AND  PROJ.PNAME = "CAD/CAM"
AND   (DUR = 12 OR DUR = 24)
```

- Select operation (σ) : It selects the tuples that satisfy the given predicate from a relation.
- Project operation (π) : It projects the columns that satisfy the given predicate.
- Cartesian Product (x) : Combines information of two different relations into one.

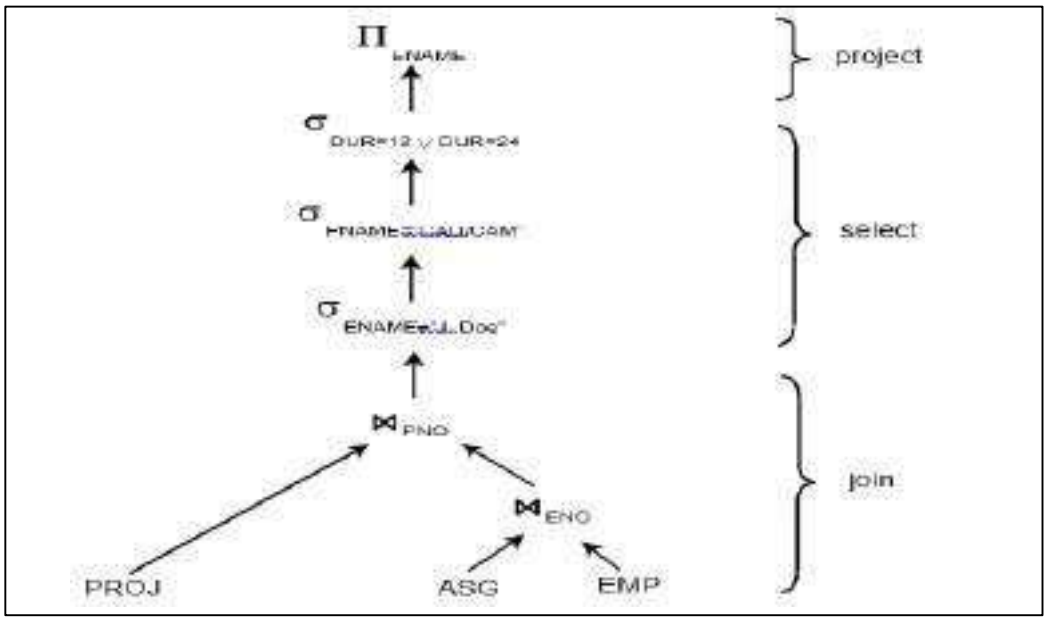


Fig : Example of Operator Tree

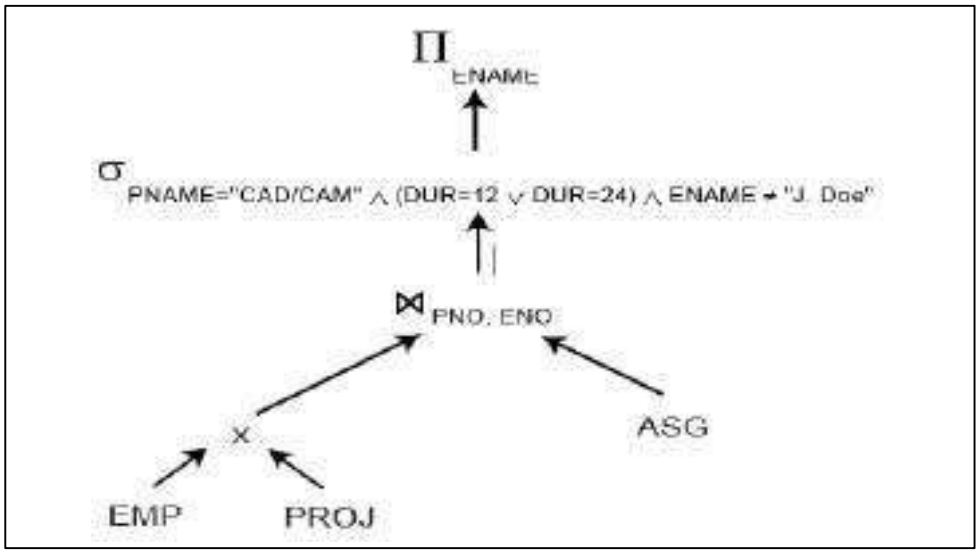


Fig : Equivalent Operator Tree

2. Data Localization

- The input to the second layer is an algebraic query on global relations. The main role of the second layer is to localize the query's data using data distribution information in the fragment schema.
- This layer determines which fragments are involved in the query and transforms the distributed query into a query on fragments.
- A global relation can be reconstructed by applying the fragmentation rules, and then deriving a program, called a localization program, of relational algebra operators, which then act on fragments.

3. Global Query Optimization

- The input to the third layer is an algebraic query on fragments. The goal of query optimization is to find an execution strategy for the query which is close to optimal.
- The previous layers have already optimized the query, for example, by eliminating redundant expressions. However, this optimization is independent of fragment characteristics such as fragment allocation and cardinalities.
- Query optimization consists of finding the "best" ordering of operators in the query, including communication operators that minimize a cost function.
- The output of the query optimization layer is a optimized algebraic query with communication operators included on fragments. It is typically represented and saved (for future executions) as a distributed query execution plan.

4. Distributed Query Execution

- The last layer is performed by all the sites having fragments involved in the query.
- Each sub query executing at one site, called a local query, is then optimized using the local schema of the site and executed.

Localization of Distributed Data

- The localization layer translates an algebraic query on global relations into an algebraic query expressed on physical fragments. Localization uses information stored in the fragment schema.
- The general techniques for decomposing and restructuring queries expressed in relational calculus.
- These global techniques apply to both centralized and distributed DBMSs and do not take into account the distributionof data. This is the role of the localization layer.
- the localization layer translates an algebraic query on global relations into an algebraic query expressed on physical fragments. Localization uses information stored in the fragment schema.

- Fragmentation is defined through fragmentation rules, which can be expressed as relational queries.
- a global relation can be reconstructed by applying the reconstruction (or reverse fragmentation) rules and deriving a relational algebra program whose operands are the fragments. We call this a *localization program*.
- In each type of fragmentation, we present *reduction techniques* that generate simpler and optimized queries.
  - ❖ Reduction for Primary Horizontal Fragmentation
    - Reduction with Selection
    - Reduction with Join
  - ❖ Reduction for Vertical Fragmentation
  - ❖ Reduction for Derived Fragmentation
  - ❖ Reduction for Hybrid Fragmentation

- **The Horizontal fragmentation** function distributes a relation based on selection predicates.

Example : Relation EMP(ENO, ENAME, TITLE) of Figure 2.3 can be split into three horizontal fragments EMP_1, EMP_2, and EMP_3, defined as follows:

$EMP_1 = \sigma_{ENO \leq "E3"}(EMP)$

$EMP2 = \sigma "E3" < ENO \leq "E6"(EMP)$

$EMP_3 = \sigma_{ENO > "E6"}(EMP)$

The localization program for an horizontally fragmented relation is the union of the fragments. In our example we have

$EMP = EMP_1 \cup EMP_2 \cup EMP_3$

- **The vertical fragmentation** function distributes a relation based on projection attributes. Since the reconstruction operator for vertical fragmentation is the join, the localization program for a vertically fragmented relation consists of the join of the fragments on the common attribute.

Example : Relation EMP can be divided into two vertical fragments where the key attribute ENO is duplicated:

$EMP1 = \Pi\ ENO, ENAME(EMP)$

$EMP2 = \Pi\ ENO, TITLE(EMP)$

The localization program is

$EMP = EMP_1\ 0_{ENO}\ EMP_2$

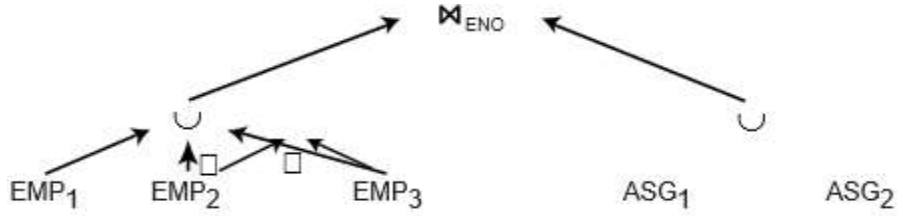
- **The Derived horizontal fragmentation** is another way of distributing two relations so that the joint processing of select and join is improved.

Example : Given a one-to-many relationship from EMP to ASG, relation ASG(ENO, PNO, RESP, DUR) can be indirectly fragmented according to the following rules:

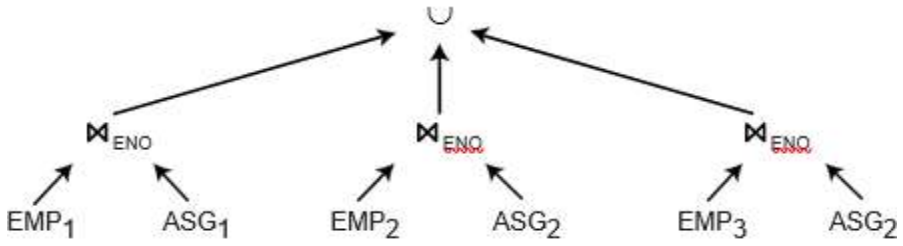
$$ASG_1 = ASG \ n_{ENO} \ EMP_1 \ ASG_2 = ASG \ n_{ENO} \ EMP_2$$

The localization program for a horizontally fragmented relation is the union of the fragments. In our example, we have

$$ASG = ASG_1 \cup ASG_2.$$



(a) Localized query



(b) Reduced query

- **The Hybrid fragmentation** is obtained by combining the fragmentation (horizontal & vertical) functions. The goal of hybrid fragmentation is to support, efficiently, queries involving projection, selection, and join.

  hybrid fragmentation based on selection-projection will make selection only, or projection only, less efficient than with horizontal fragmentation (or vertical fragmentation). The localization program for a hybrid fragmented relation uses unions and joins of fragments.

Example : Here is an example of hybrid fragmentation of relation EMP:

$$EMP1 = \sigma ENO \leq \text{"E4"}(\Pi ENO,ENAME(EMP))$$

$$EMP2 = \sigma ENO > \text{"E4"}(\Pi ENO,ENAME(EMP))$$

$$EMP3 = \Pi ENO,TITLE(EMP)$$

In our example, the localization program is

$$EMP = (EMP_1 \cup EMP_2) \ 0_{ENO} \ EMP_3$$

Queries on hybrid fragments can be reduced by combining the rules used, respectively, in primary horizontal, vertical, and derived horizontal fragmentation.

These rules can be summarized as follows:

1. Remove empty relations generated by contradicting selections on horizontal fragments.
2. Remove useless relations generated by projections on vertical fragments.
3. Distribute joins over unions in order to isolate and remove useless joins.

Distributed Query Optimization

Query Optimization

- Query optimization refers to the process of producing a query execution plan (QEP) which represents an execution strategy for the query.
- This QEP minimizes an objective cost function. A query optimizer, the software module that performs query optimization.
- usually it consisting of three components:
    - a search space,
    - a cost model, and
    - a search strategy
- The ***search space*** is the set of alternative execution plans that represent the input query.
- These plans are equivalent, in the sense that they yield the same result, but they differ in the execution order of operations and the way these operations are implemented, and therefore in their performance.
- The ***cost model*** predicts the cost of a given execution plan. To be accurate, the cost model must have good knowledge about the distributed execution environment.
- The ***search strategy*** explores the search space and selects the best plan, using the cost model. It defines which plans are examined and in which order.
- The details of the environment (centralized versus distributed) are captured by the search space and the cost model.
- Query execution plans are typically abstracted by means of operator trees, which define the order in which the operations are executed.
- They are enriched with additional information, such as the best algorithm chosen for each operation.
- For a given query, the search space can thus be defined as the set of equivalent operator trees that can be produced using transformation rules.
- To characterize query optimizers, it is useful to concentrate on *join trees*, which are operator trees whose operators are join or Cartesian product.
- This is because permutations of the join order have the most important effect on performance of relational queries.

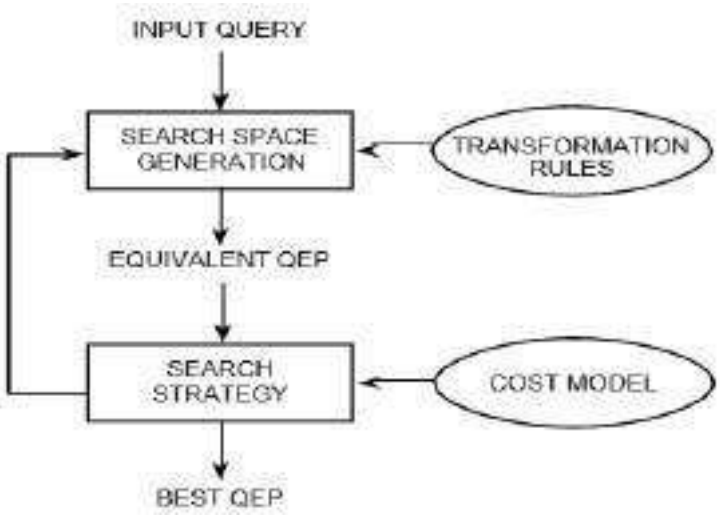


Fig : Query Optimization Process

Example : Consider the following query:

```
SELECT ENAME,  RESP FROM EMP, ASG, PROJ
WHERE  EMP.ENO=ASG.ENO
AND  ASG.PNO=PROJ.PNO
```

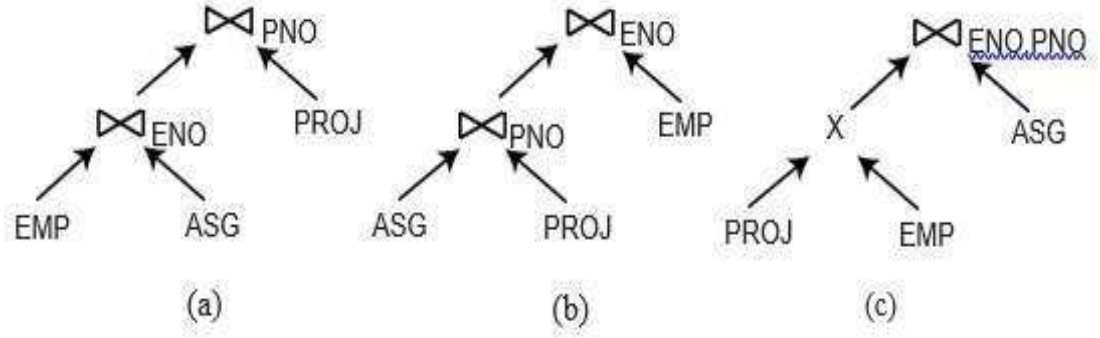


Fig : Equivalent Join Trees

- Each of these join trees can be assigned a cost based on the estimated cost of each operator.
- Join tree (c) which starts with a Cartesian product may have a much higher cost than the other join trees.
- If a query is large we use restrictions.
- The first restriction is to use heuristics. The most common heuristic is to perform selection and projection when accessing base relations.
- Another commonheuristic is to avoid Cartesian products that are not required by the query.
- operator tree (c) would not be part of the search space considered bythe optimizer.
- Another important restriction is with respect to the shape of the join tree.
- Two kinds of join trees are usually distinguished: linear versus bushy trees.

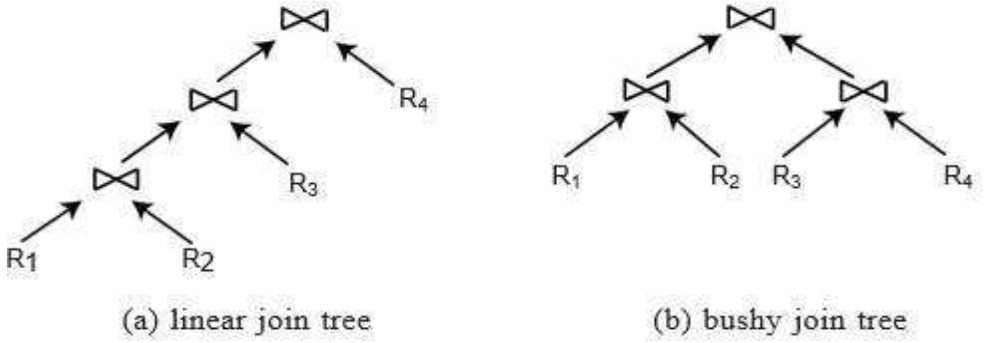


Fig : The Two Major Shapes of Join Trees

- A *linear tree* is a tree such that at least one operand of each operator node isa base relation.
- A *bushy tree* is more general and may have operators with no base relations as operands (i.e., both operands are intermediate relations).
- By considering only linear trees, the size of the search space is reduced to $O(2^N)$.
- In a distributed environment, bushy trees are useful in exhibiting parallelism.
- For example, in join tree operations R_1 *0* R_2 and R_3 *0* R_4 can be done in parallel.

Cost Functions

- The cost of a distributed execution strategy can be expressed with respect to eitherthe total time or the response time.
- The cost is in terms of execution time, so a cost function represents the execution time of a query.
- The total time is the sum of all time (also referred to as cost) components, while the response time is the elapsed time from the initiation to the completion of the query.
- A general formula for determining the total time canbe specified as follows :

$$\textbf{\textit{Total time}} = T_{CPU} * \#insts + T_{I/O} * \#I/Os + T_{MSG} * \#msgs + T_{TR} * \#bytes$$

- The two first components measure the local processing time.
- T_{CPU} is the time of a CPU instruction and $T_{I/O}$ is the time of a disk I/O. The communication time is depicted by the two last components.
- T_{MSG} is the fixed time of initiating and receiving a message.
- T_{TR} is the time it takes to transmit a data unit from one site to another.
- The data unit is given here in terms of bytes (*#bytes* is the sum of the sizes of all messages), but could be in different units (e.g., packets).
- A typical assumption is that T_{TR} is constant. This might not be true for wide area networks, where some sites are farther away than others. However, this assumption greatly simplifies query optimization.
- Thus the communication time of transferring *#bytes*of data from one site to another is assumed to be a linear function of *#bytes*:

$$CT(\#bytes) = T_{MSG} + T_{TR} * \#bytes$$

- Costs are generally expressed in terms of time units, which in turn, can be translated into other units (e.g., dollars).

Centralized Query Optimization

- There are three reasons to understanding distributed query optimization.
- First, a distributed query is translated into local queries, each of which is processed in a centralized way.
- Second, distributed query optimization techniques are often extensions of the techniques for centralized systems.
- Finally, centralized query optimization is a simpler problem
- The minimization of communication costs makes distributed query optimization more complex.
- The optimization timing, which can be dynamic, static or hybrid, is a good basis for classifying query optimization techniques

Dynamic Query Optimization

- Dynamic query optimization combines the two phases of query decomposition and optimization with execution.
- The QEP is dynamically constructed by the query optimizer which makes calls to the DBMS execution engine for executing the query's operations.
- Thus, there is no need for a cost model.
- Dynamic query optimization, done at run-time
- The most popular dynamic query optimization algorithm is that of INGRES

Static Query Optimization

- With static query optimization, there is a clear separation between the generation of the QEP at compile-time and its execution by the DBMS execution engine.
- Thus, anaccurate cost model is key to predict the costs of candidate QEPs.
- Static query optimization, done at compilation time
- The most popular static query optimization algorithm is that of System R.

Hybrid Query Optimization

- Hybrid query optimization attempts to provide the advantages of static query optimization while avoiding the issues generated by inaccurate estimates.
- The approach is basically static, but further optimization decisions may take place at run time.

Distributed Query Optimization Algorithms

Dynamic Approach :

INGRES Algorithm

- The algorithm of Distributed **INGRES** illustrates the **dynamic approach**.
- It recursively breaks a query into smaller pieces
- The objective function of the algorithm is to minimize a combination of both the communication time and the response time. The algorithm also takes advantage of fragmentation, but only horizontal fragmentation is handled for simplicity.
- INGRES is a popular relational DB system and it has a distributed version whose optimization algorithms are extensions of the centralized versions.
- It uses a dynamic query optimization algorithm that recursively breaks-up a calculus query into smaller pieces.
- INGRES combines calculus-algebra decomposition and optimization.
- A query is first decomposed into a subsequence of queries having a unique relation in common.
- Then each mono relation query is processed by a **One - Variable Query Processor (OVQP).**
- The OVQP optimizes the access to a single-relation by selecting the best access method to that relation.
- (Eg : index, sequential scan).
- Given an N - relation query q, the INGRES query processor decompose q into n subqueries
- q1->q2->…..->qi.
- This decomposition uses two basic techniques: detachments and substitutions.
- Each qi is a mono relation query; the output of qi is consumed by qi+1. Detachment is used first.
- There is a processor that can efficiently process mono-relation queries optimizes each query independently for the access to single relation.

Static Approach :

R* Algorithm

- It uses **static optimization algorithm** based on an exhaustive search of solution space.
- It uses database statistics to determine the total cost involved.
- Under this strategy all the candidate tree are given particular cost and lowest cost tree is retained.
- The total number of trees possible is determined using dynamic programming under which those paths, which are not optimal are not considered and also which includes Cartesian product is not considered.
- The disadvantage of this algorithm is that it is costly and does not deal with fragments.

Semijoin Algorithm:

Hill-Climbing query optimization algorithm

- Refinements of an initial feasible solution are recursively computed until no more cost improvements can be made
- Semijoins, data replication, and fragmentation are not used
- Devised for wide area point-to-point networks
- The first distributed query processing algorithm

SDD-1 Algorithm

The SDD-1 query optimization algorithm improves the Hill-Climbing algorithm in a number of directions:

– Semijoins are considered

– More elaborate statistics

– Initial plan is selected better

– Post-optimization step is introduced

Hill Climbing using Semijoin

Initialization

Step 1: In the execution strategy (call it ES), include all the local processing

Step 2: Reflect the effects of local processing on the database profile

Step 3: Construct a set of beneficial semijoin operations (BS) as follows :

BS = Ø

For each semijoin SJi

BS ← BS ∪ SJi if cost(SJi) < benefit(SJi)

Iterative Process

Step 4: Remove the most beneficial SJi from BS and append it to ES

Step 5: Modify the database profile accordingly

Step 6: Modify BS appropriately

– compute new benefit/cost values

– check if any new semijoin needs to be included in BS

Step 7: If BS ≠ Ø, go back to Step 4

Assembly Site Selection

Step 8: Find the site where the largest amount of data resides and select it as the assembly site

Postprocessing

Step 9: For each Ri at the assembly site, find the semijoins of ⋈ the type Ri Rj.

where the total cost of ES without this semijoin is smaller than the cost with it and remove the semijoin from ES.

Step 10: Permute the order of semijoins, if doing so would improve the total cost of ES.