Characteristics of Query processor

→ Important characteristics of query Processor

- → Languages
- → Types of optimization
- → Optimization Timing
- → Statistics
- → Decision sites
- → Exploitation of the Network Topology
- → Exploitation of Replicated fragments
- → Use of Semi join.

① Languages:-

Input language to the query Processor is based on high level relational DBMs.

i.e Input is in calculus query.

On the other hand output language is based on lower level relational DBMs. i.e output is in algebraic query.

The operations of the output language are implemented directly in the computer system.

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

Types of optimization:

Among all possible strategies for executing query. the one in which less time and space are required is the best solution for the optimization query.

A comparative study for all the strategies should perform such that how much cost will be required for each solution. that strategy should be choosen in which there will be less cost. but keep in mind that choosing the low cost strategy may require much space. so care should be taken in this case also.

Optimization Timing:

The actual time required to optimize the execution of a query is an important factor. If less time is required, then it is the best solution for query Processing.

Statistics:

The Effectiveness of query optimization relies on statistical information of the database. i.e how many fragments query will be needed. which operation should be done first.

size and number of distinct values of each attribute of the relation, histograms of the attributes for minimizing probability error etc.

Decision sites:-

In a distributed database, several sites may participate for answering the query. Most systems use centralized decision approach. In which a single site generates the strategy. However, the decision process could be distributed among various sites.

Exploitation of Network Topology:-

Distributed query processor uses computer Network. So its performance depends also on which topology. It is using. The cost function, speed, utilization of various network resources are important factors for executing query processor in a distributed Environment.

Exploitation of Replicated fragments.

A distributed relation is usually divided in to relation fragments

Distributed queries expressed on global relations are mapped in to queries on Physical fragments of

relations by translating relations in to fragments. We call this process localization because its main function is to localize the data involved in the query.

For higher reliability and better read performance, it is useful to have fragments replicated at different sites.

Use of Semi join operation:-

In Distributed system, it is better to use Semi-join operation instead of join operation for minimizing data communication.

Step 2 – Data Localization

Input: Algebraic query on distributed relations

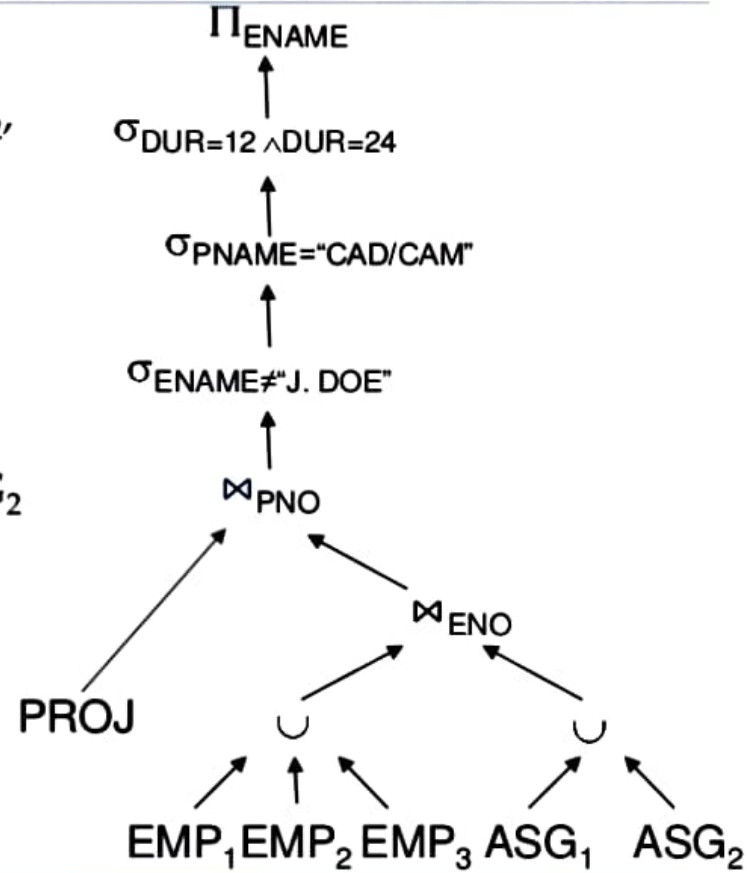
- Determine which fragments are involved
- Localization program
  - ➡ substitute for each global query its materialization program
  - ➡ optimize

Example

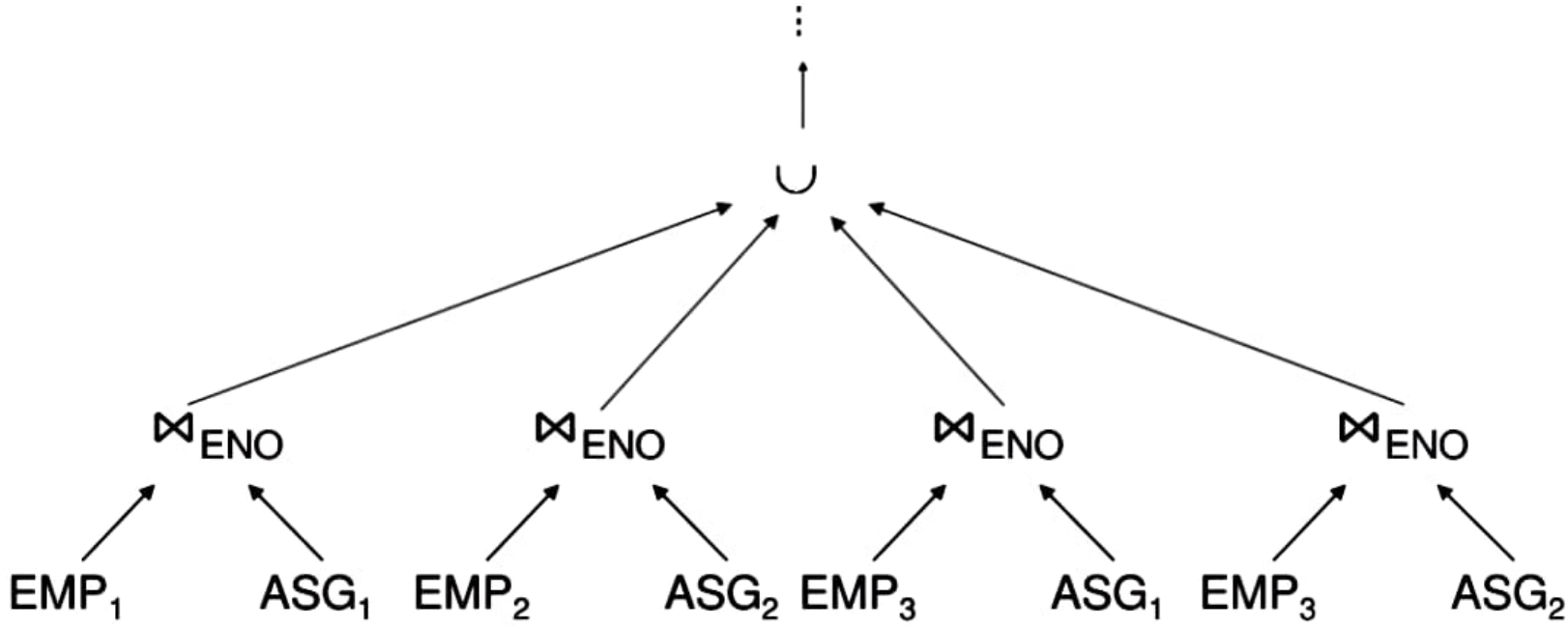
Assume

- EMP is fragmented into EMP_1, EMP_2, EMP_3 as follows:
  - $EMP_1 = \sigma_{ENO \leq \text{"E3"}}(EMP)$
  - $EMP_2 = \sigma_{\text{"E3"} < ENO \leq \text{"E6"}}(EMP)$
  - $EMP_3 = \sigma_{ENO \geq \text{"E6"}}(EMP)$
- ASG fragmented into ASG_1 and ASG_2 as follows:
  - $ASG_1 = \sigma_{ENO \leq \text{"E3"}}(ASG)$
  - $ASG_2 = \sigma_{ENO > \text{"E3"}}(ASG)$

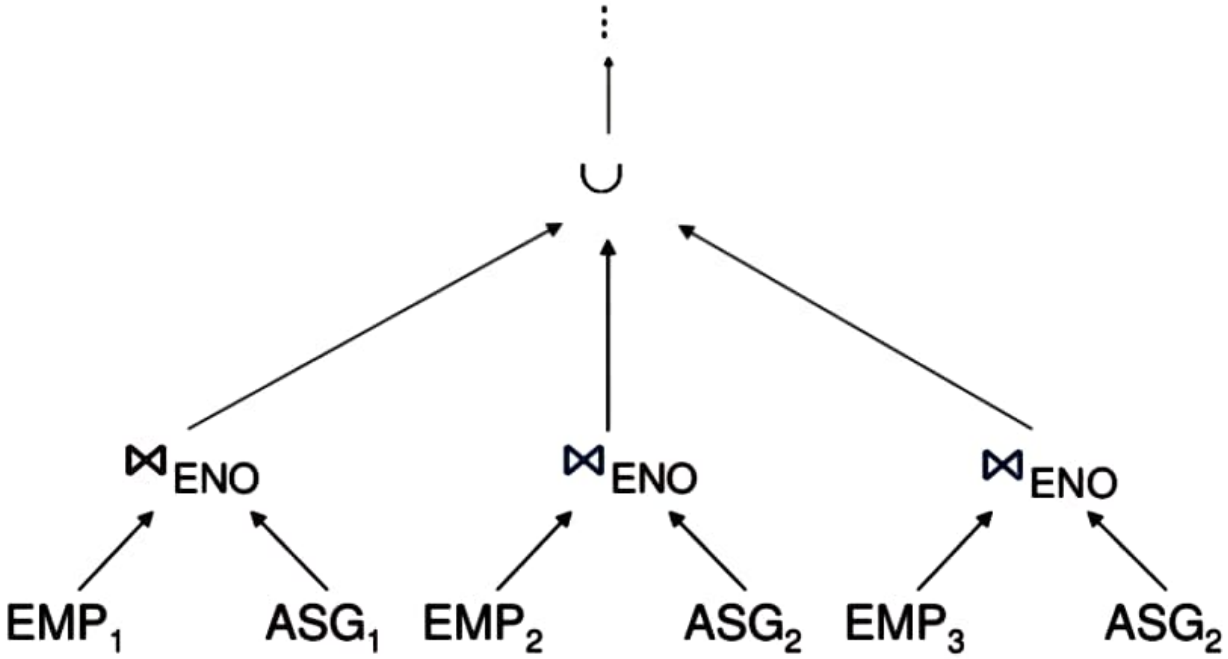
Replace EMP by $(EMP_1 \cup EMP_2 \cup EMP_3)$ and ASG by $(ASG_1 \cup ASG_2)$ in any query



Provides Parallellism



Eliminates Unnecessary Work

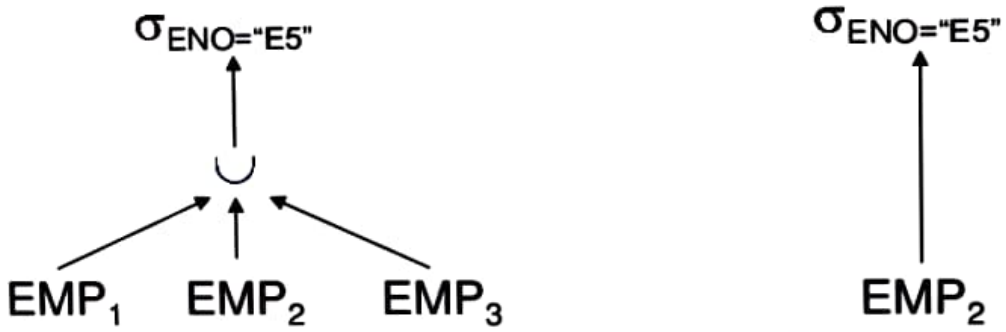


Reduction for PHF

- Reduction with selection
  - Relation R and $F_R=\{R_1, R_2, \ldots, R_w\}$ where $R_j=\sigma_{p_j}(R)$

    $\sigma_{p_i}(R_j)=\varnothing$ if $\forall x$ in R: $\neg(p_i(x)\wedge p_j(x))$
  - Example

```
SELECT    *
FROM      EMP
WHERE     ENO="E5"
```



Reduction for PHF

- Reduction with join
  - Possible if fragmentation is done on join attribute
  - Distribute join over union

$$(R_1 \cup R_2) \bowtie S \Leftrightarrow (R_1 \bowtie S) \cup (R_2 \bowtie S)$$

  - Given $R_i = \sigma_{p_i}(R)$ and $R_j = \sigma_{p_j}(R)$

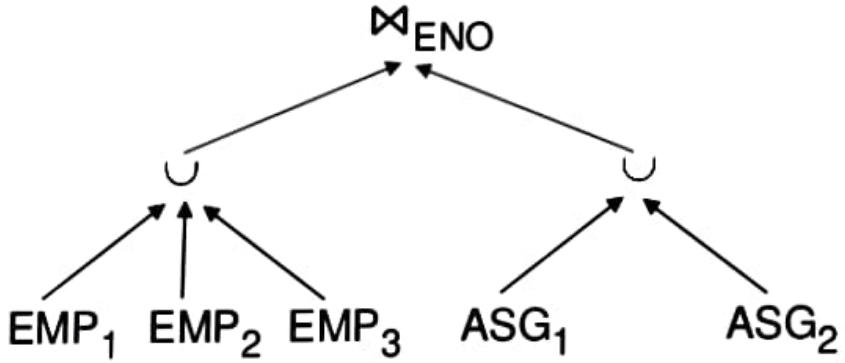
$$R_i \bowtie R_j = \varnothing \text{ if } \forall x \text{ in } R_i,\ \forall y \text{ in } R_j: \neg(p_i(x) \wedge p_j(y))$$

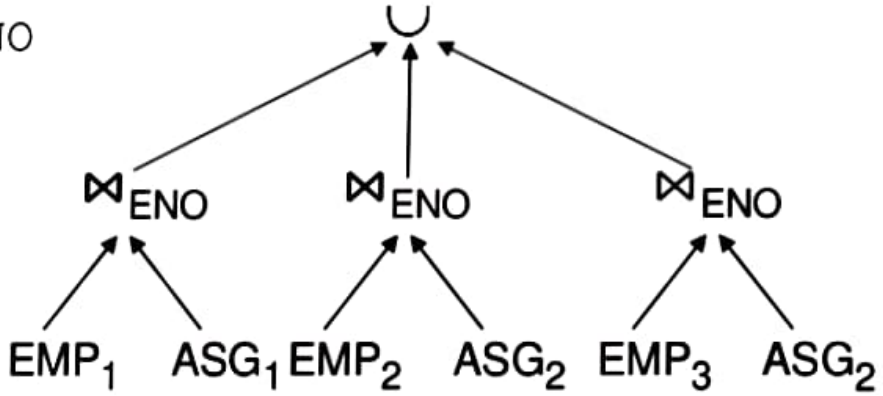
Reduction for PHF

- Assume EMP is fragmented as before and
  - ASG_1: $\sigma_{ENO \leq "E3"}(ASG)$
  - ASG_2: $\sigma_{ENO > "E3"}(ASG)$
- Consider the query

```
SELECT   *
FROM     EMP, ASG
WHERE    EMP.ENO=ASG.ENO
```

- Distribute join over unions
- Apply the reduction rule





Reduction for VF

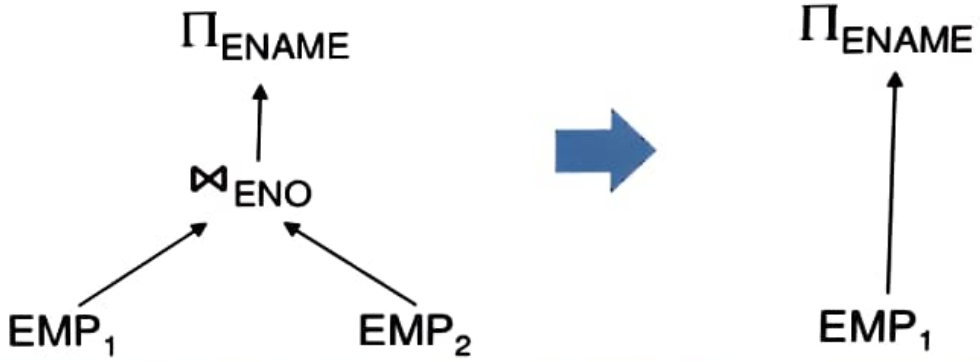
- Find useless (not empty) intermediate relations

  Relation R defined over attributes $A = \{A_1, ..., A_n\}$ vertically fragmented as $R_i = \Pi_{A'}(R)$ where $A' \subseteq A$:

  $\Pi_{D,K}(R_i)$ is useless if the set of projection attributes D is not in A'

  Example: $EMP_1 = \Pi_{ENO,ENAME}(EMP)$; $EMP_2 = \Pi_{ENO,TITLE}(EMP)$

```
SELECT  ENAME
FROM                EMP
```



Reduction for DHF

- Rule :
  - Distribute joins over unions
  - Apply the join reduction for horizontal fragmentation
- Example

  ASG_1: $ASG \ltimes_{ENO} EMP_1$

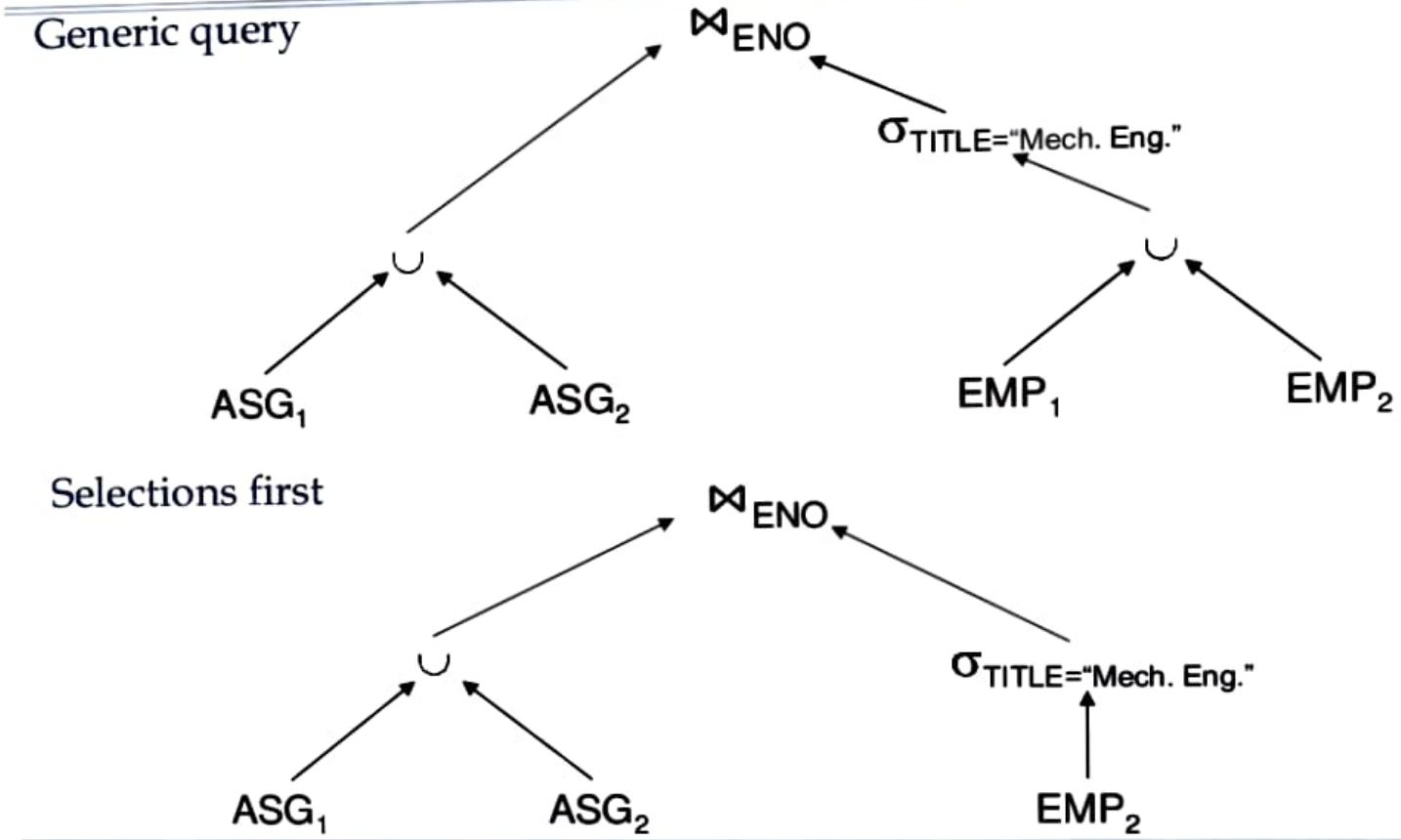
  ASG_2: $ASG \ltimes_{ENO} EMP_2$

  EMP_1: $\sigma_{TITLE=\text{“Programmer”}}(EMP)$

  EMP_2: $\sigma_{TITLE=\text{“Programmer”}}(EMP)$
- Query

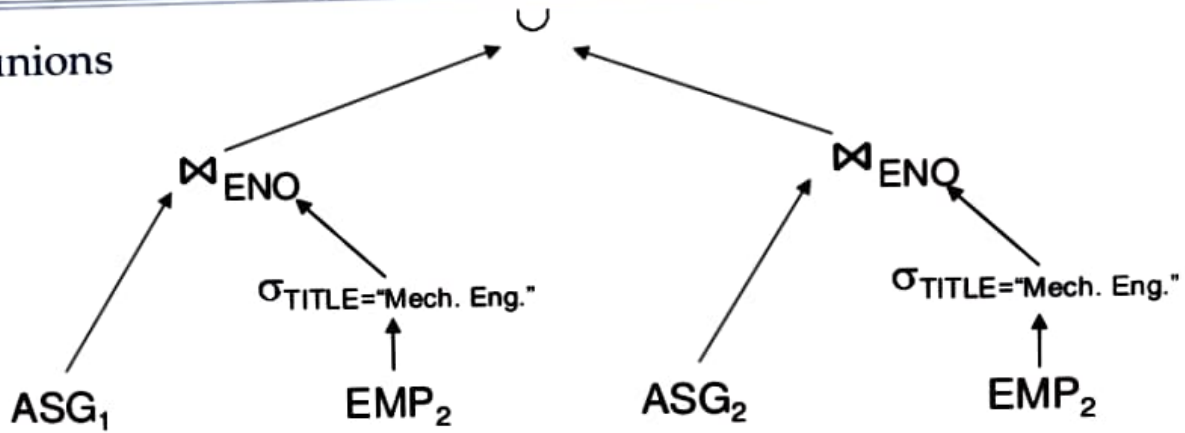
```
SELECT    *
FROM      EMP, ASG
WHERE ASG.ENO = EMP.ENO
AND       EMP.TITLE = "Mech. Eng."
```

Reduction for DHF



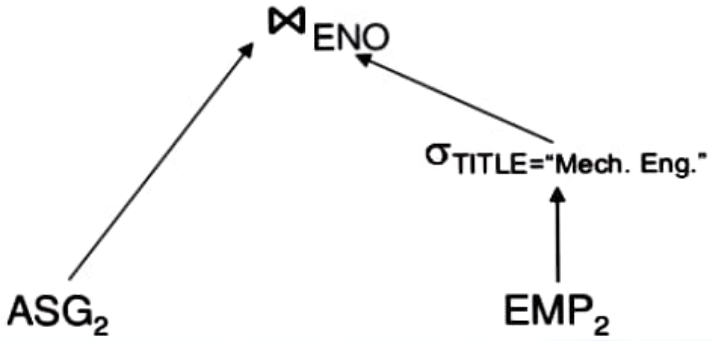
Reduction for DHF

Joins over unions



Elimination of the empty intermediate relations

(left sub-tree)



Reduction for Hybrid Fragmentation

- Combine the rules already specified:
  - Remove empty relations generated by contradicting selections on horizontal fragments;
  - Remove useless relations generated by projections on vertical fragments;
  - Distribute joins over unions in order to isolate and remove useless joins.

Reduction for HF

Example

Consider the following hybrid fragmentation:

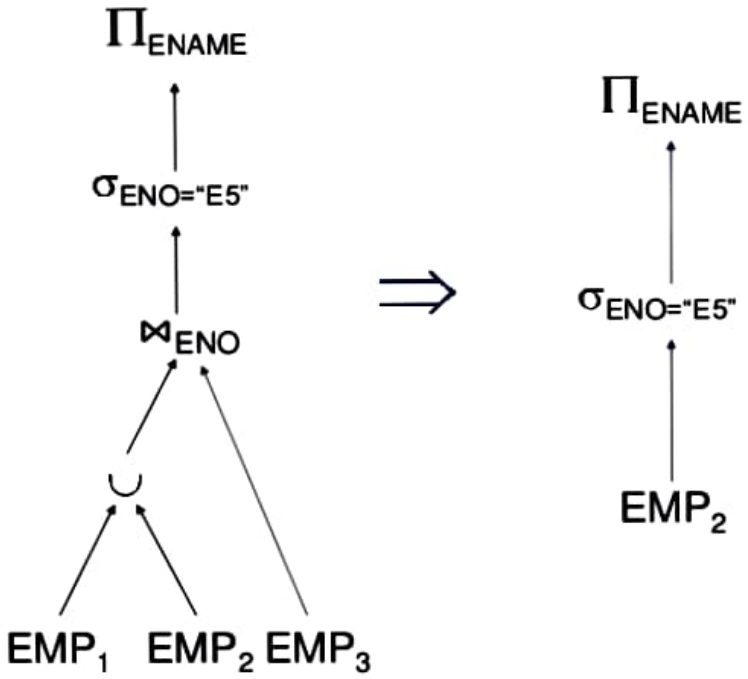
$EMP_1 = \sigma_{ENO \leq "E4"} (\Pi_{ENO,ENAME} (EMP))$

$EMP_2 = \sigma_{ENO > "E4"} (\Pi_{ENO,ENAME} (EMP))$

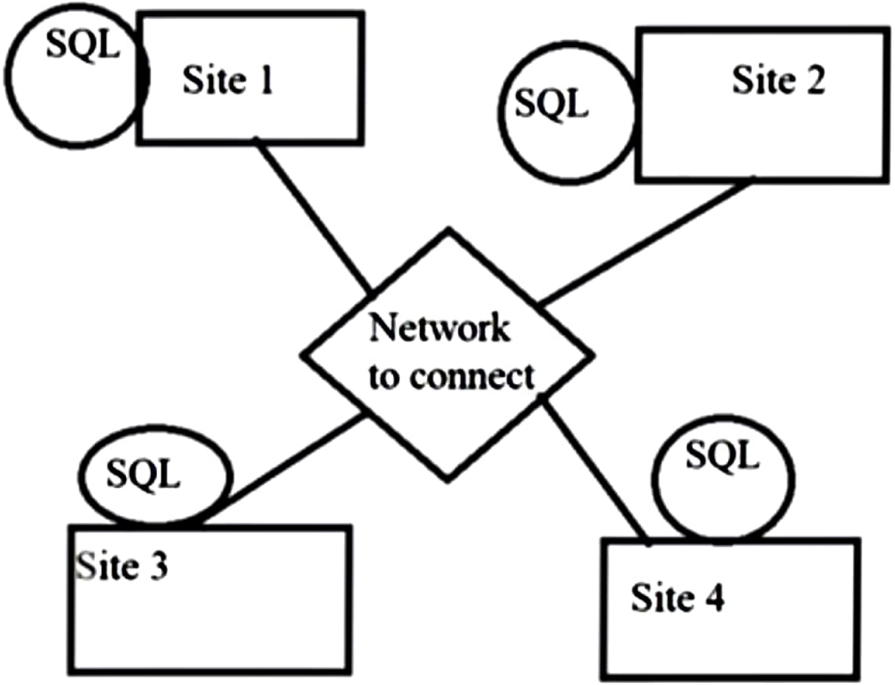
$EMP_3 = \sigma_{ENO,TITLE} (EMP)$

and the query

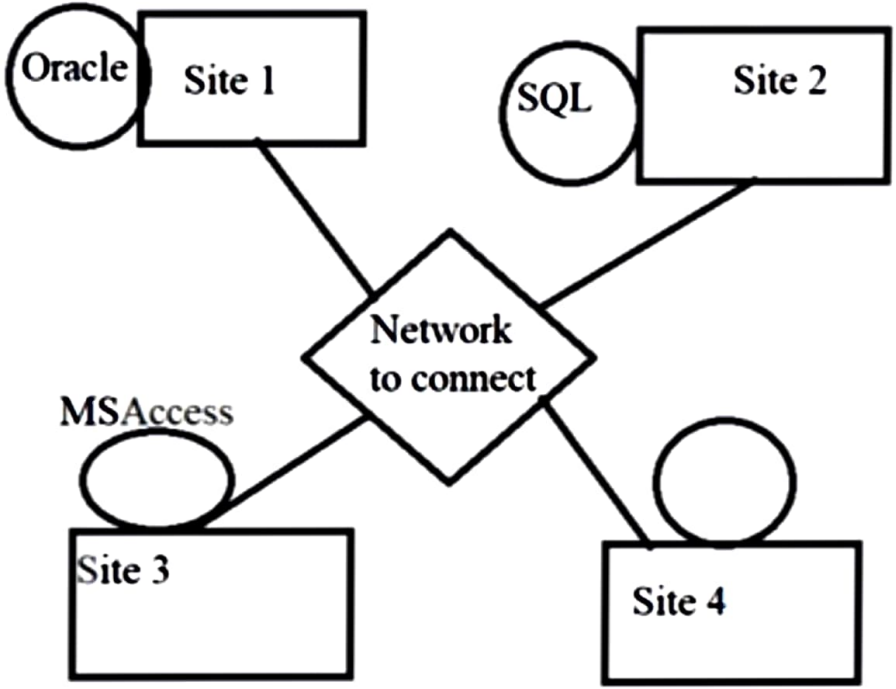
```
SELECT ENAME
FROM   EMP
WHERE  ENO="E5"
```



Distributed Query Processing



Homogeneous Distributed Database



Heterogeneous Distributed Database

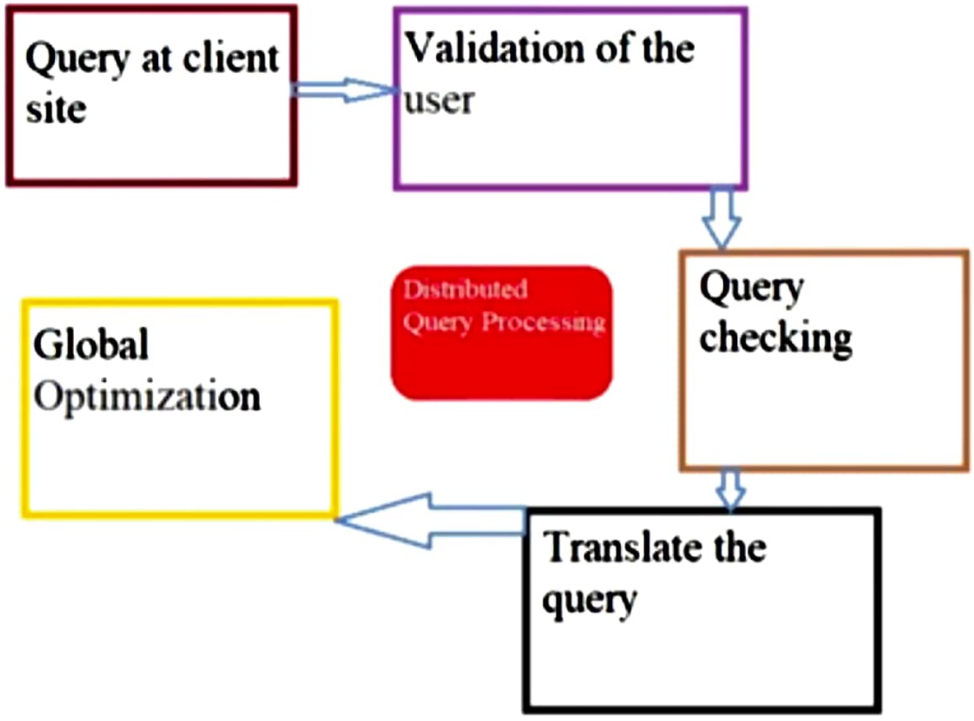

Query at client site
Validation of the user
Query checking
Distributed Query Processing
Global Optimization
Translate the query

Table T1, Site Y
Records=100

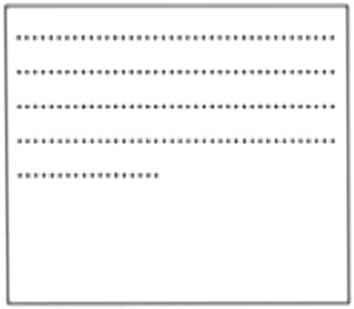


Table T2, Site Z
Records=10,000

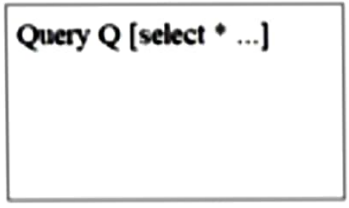


Table T1, Site Y
Records=100

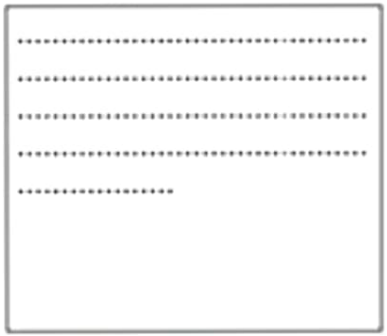
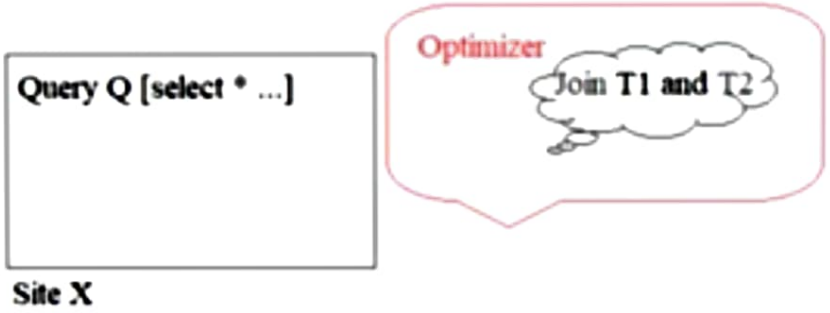


Table T2, Site Z
Records=10,000



Site X

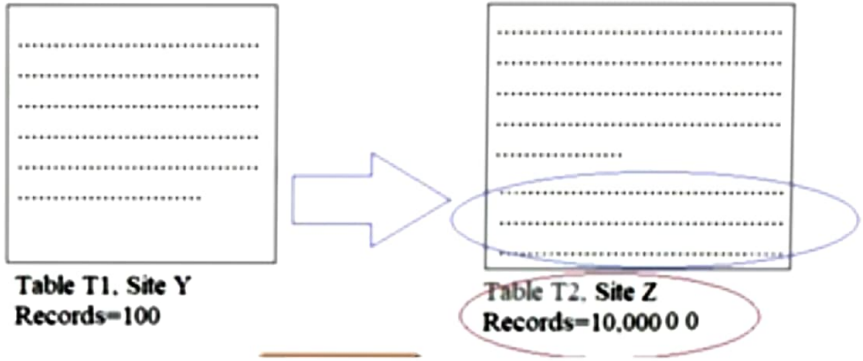


Table T1. Site Y
Records=100
Table T2. Site Z
Records=10,000 0 0

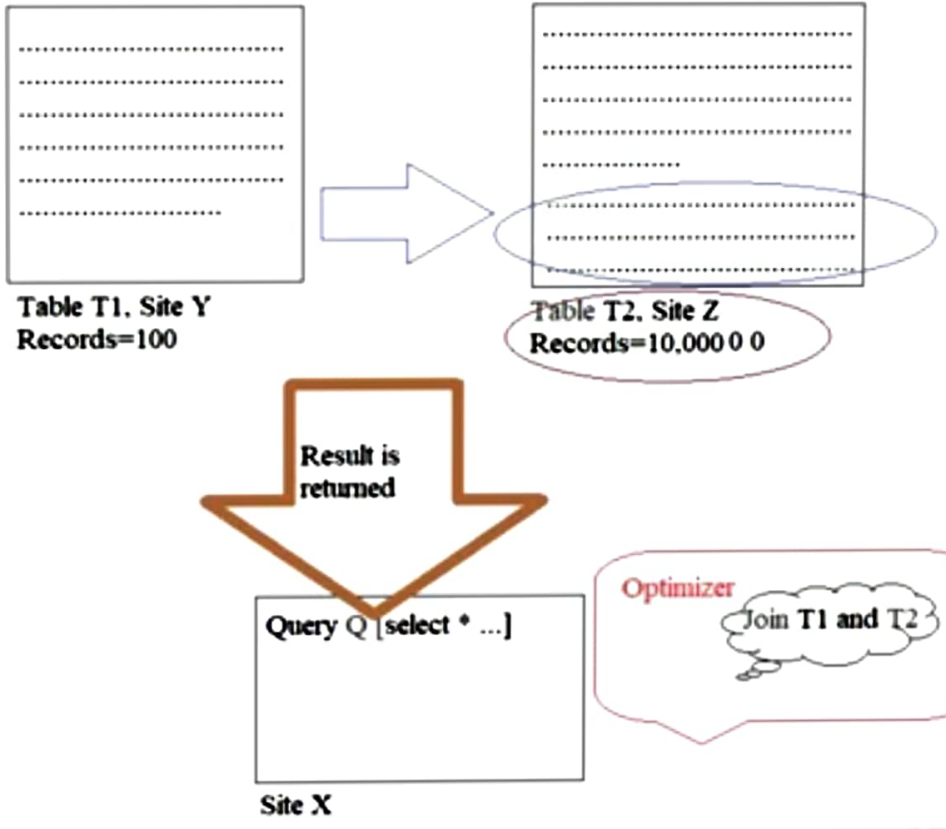
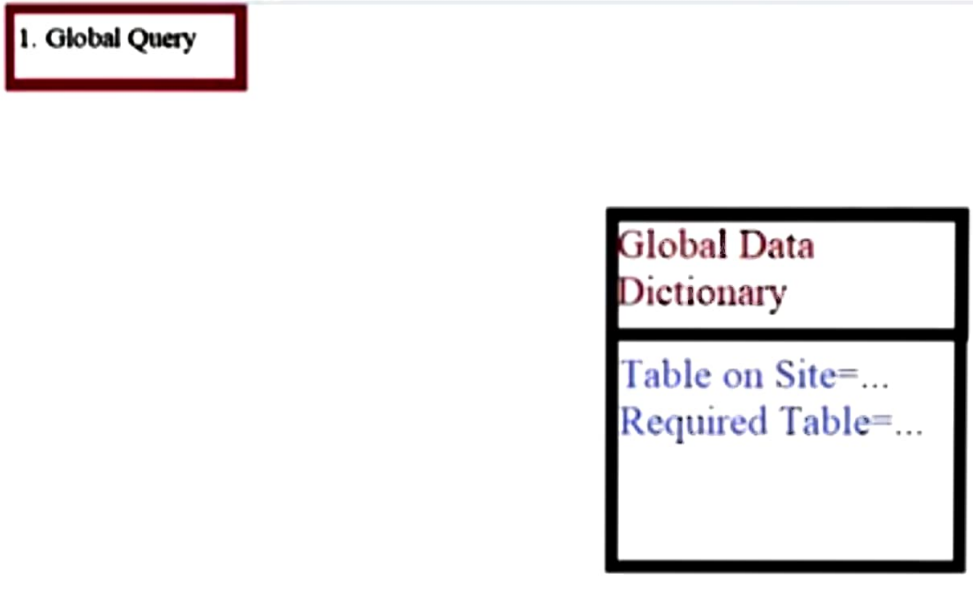
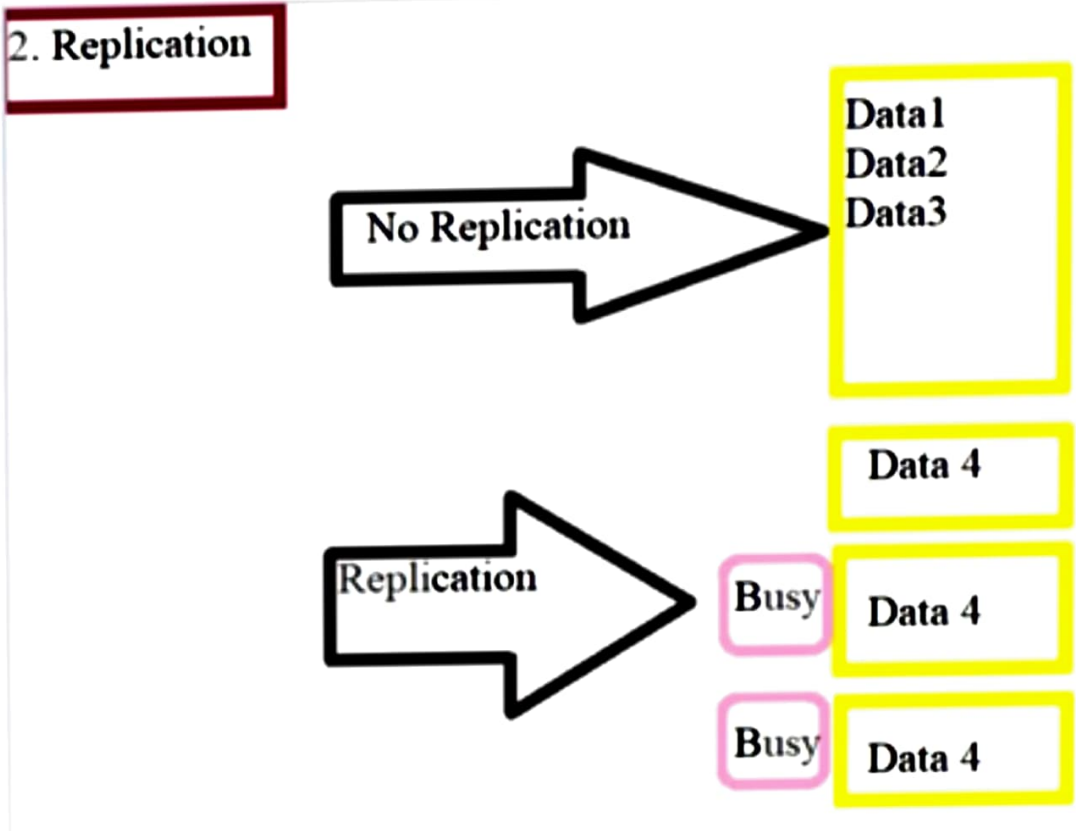


Table T1, Site Y
Records=100
Table T2, Site Z
Records=10,000 0 0
Result is returned
Query Q [select * ...]
Site X
Optimizer
Join T1 and T2


1. Global Query
Global Data Dictionary
Table on Site=...
Required Table=...

2. Replication



3. Execution Plan

Based on the knowledge gathered for query execution and optimization and to return the asked result

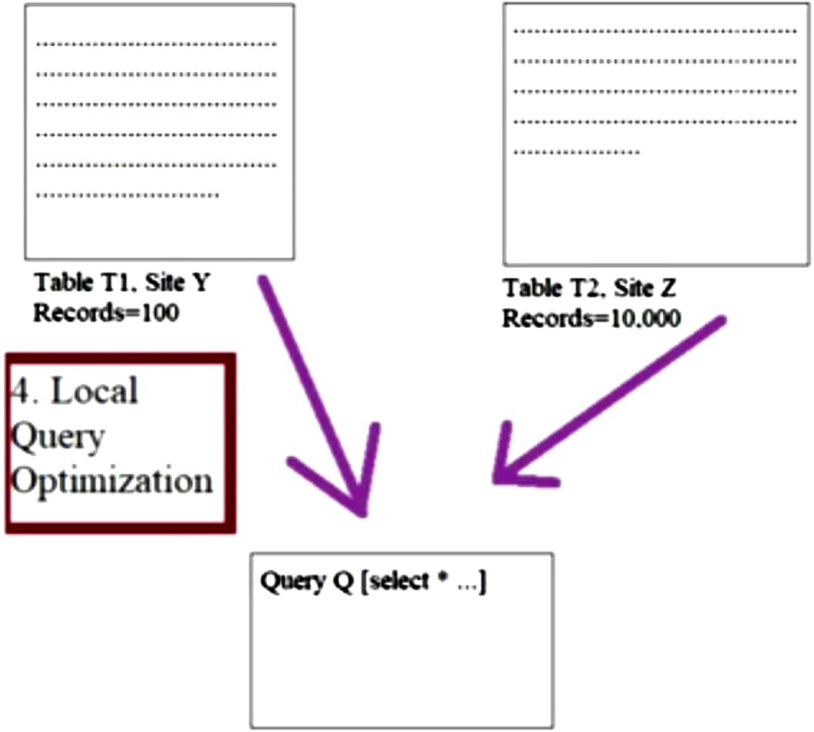
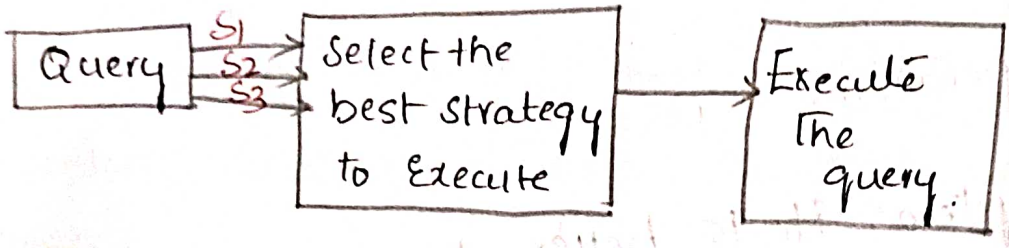


Table T1. Site Y
Records=100
Table T2. Site Z
Records=10.000
4. Local Query Optimization
Query Q [select * ...]

Query Optimization:

Query optimization is the process of selecting the most efficient query evaluation plan among the many strategies.



Student

Roll.No	Name	Add.
01	Rama	china
02	Krishna	Mumbai
03	Suresh	Hydrabad
04.	Naresh	kolkata.

Result

Roll. NO	Sub id	Marks
1	S_1	90
2	S_2	60
3	S_1	75
4	S_3	85

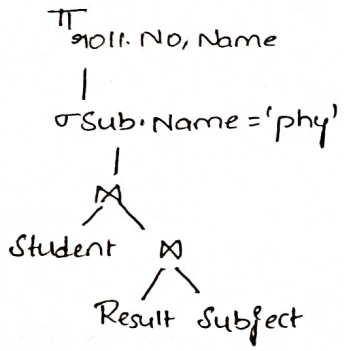
Subject

Sub id	Sub Name
S_1	Phy
S_2	Chemistry
S_3	maths

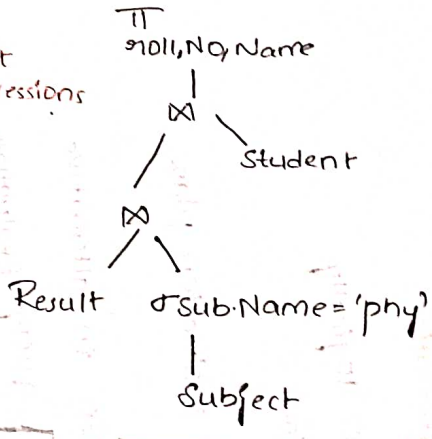
⇒ List the roll.No, Name of the students who have Enrolled in physics.

$$\Pi_{roll, Name}\left(\sigma_{subName = 'phy'}(Student \bowtie Result \bowtie Subject)\right)$$

This Expression constructs a large intermediate table. However, we can intrested in only a few records of this relation



Equivalent Expressions ⟺



Operator tree.

Roll. No.	Name	Add	Sub Id	Marks	Sub Name.
01	Rama	China	S1	90	Phy
02	Krishna	Mumbai	S2	60	che
03	Suresh	Hyd	S1	75	Phy
04	Naresh	Kolkata	S2	85	Maths.

Roll. No	Sub Id	Marks	Sub Name
01	S1	90	Phy
02	S2	60	che
03	S1	75	Phy
04	S3	85	maths.

1	Rama
3	Suresh

Roll.	Name	Sub Id	Marks	Sub Name
1	Rama	S1	90	Phy
3	Suresh	S1	75	Phy.

Roll.	Sub Id	Marks	Sub Name
1	S1	90	Phy
3	S1	75	Phy

Sub Id.	Sub Name
S1	Phy.

Given a relational algebra expression. it is the job of the query optimizer to come up with a query Evaluation plan that compute the same result as the given expression. and is the least costly way of generating result.

Steps of query Evaluation plan generation

① Generating expression that are logically Equivalent to the given expression.
By Equivalence rule

② Estimating the cost of each evaluation plan.
By collecting statistical info of tables.

③ Annotating the result expression in alternative ways to generate alternative query - Evaluation Plan.

Centralized Query Optimization:-

In this we present the main query optimization techniques for centralized systems.

This presentation is a prerequisite to understanding distributed query optimization for three reasons.

First, a distributed query is translated in to 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.

In a centralized system, query processing is done with the foll. aim —

* Minimizing of ~~resource~~ ~~tt~~ response time of query (Time taken to produce the results to user's query)
* Maximize system throughput (the number of requests that are processed in a given amount of time)
* Reduce the amount of memory and storage required for processing.
* Increase parallelism.

Centralized Query optimization has divided in to 3 approaches.

① Static Query optimization
② Dynamic " "
③ Hybrid " "

Dynamic Query optimization:-

Dynamic Query opt combines the two phases of query decomposition and optimization with execution.

The most popular dynamic query opt algorithm is that of INGRES one of the first relational DBms.

This algorithm recursively breaks up a query expressed in relational calculus (i.e, SQL) in to smaller pieces which are executed along the way. the query is first decomposed in to a sequence of queries having a unique relation in common. then each mono relation query is Processed by selecting based on the predicate, the best access method to that relation (eg, index, sequential scan)

INGES Algorithm

Question: To illustrate the detachment technique, we apply it to the foll. query.

"Names of employees working on the CAD/CAM project"

Answer: This query can be expressed in SQL by the foll query q1 on the engineering database.

```
q1: SELECT  Emp.ENAME
    FROM    EMP, ASG, PROJ
    WHERE   Emp.ENO = ASG.ENO
      AND   ASG.PNO = PROJ.ENO
      AND   PNAME = "CAD/CAM"
```

Static Query optimization:-

- Simple (i.e mono relation) queries are executed according to the best access path
- Execute joins
  - Determine the possible ordering of joins
  - Determine the cost of each ordering
  - Choose the join ordering with minimal cost.

For joins, two alternative algorithms

→ Nested loops.

for each tuple of external relation (cardinality n_1)

for each tuple of erno/ relation (cardinality n_2)

joins two tuples if the join predicate is true

end

end

• complexity : $n_1 * n_2$

→ Merge join

• Sort relations

• Merge relations

complexity : $n_1 + n_2$ if relations are previously sorted and equijoin.

Hybrid Query optimization :-

It is mixed of static and dynamic approaches, the approach is mainly static, but dynamic query optimization may take place when high difference between predicated and actual sizes is detected.

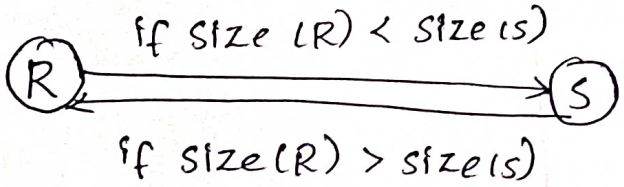
Distributed Query optimization

Join ordering is an important aspect in centralized DBMS. and it is even more important in a DDBMS. since joins between fragments that are stored at different sites may increase the communication cost.

- Two approches exist.
- Optimize the ordering of joins directly.
  - INGRES and distributed INGRES
  - System R and System R*
- Replace joins by combinations of semijoins in order to minimize the communication costs.
  - Hill climbing alg.

Direct join ordering of two relation/fragments located at different sites.

- Move the smaller relation to the other site
- We have to Estimate the size of R and S.



Natural Join operation ⋈

When we combine rows at two or more tables based on a common column between them. this operation is called "joining"

A Natural join is a type of join operation that creates an implicit join by combining tables based on columns with the same same name and data type.

Example:-

$R_1(A, B)$

A	B
a_1	b_1
a_2	b_1
a_2	b_2
a_3	b_2

$R_2(B, C)$

B	C
b_1	c_1
b_1	c_2
b_2	c_3

$R_1 \bowtie R_2$

A	B	C
a_1	b_1	c_1
a_1	b_1	c_2
a_2	b_1	c_1
a_2	b_1	c_2
a_3	b_2	c_3

⋉ Semi Join Matches the rows of two relations and then show the matching rows of the relation whose name is mentioned to the left side of ⋉ Semi join operation.

ID	Designation	Salary.
501	Asst prof	50,000
502	Asst prof	50,000
503	Lecturer	10,000

Table 1 : Teacher

ID	Roll.No	Marks.
503	2017 - Q1	60
504	2017 - Q2	90
505	2017 - Q3	70

Table 2 : Student

ID	Roll.No	Marks
503	2017 - Q1	60.

Table 3 : Student ⋉ Teacher OR student Semi join Teacher

if Teacher ⋉ Student or teacher Semi-join Student

ID	Designation	salary
503	Lecturer	10,000.

Semi-join Based algorithm:-

The join of two relations R and S over attribute A, stored at sites 1 and 2 respectively.

Cost Analysis.

$(R \bowtie_A S)$ Vs $(R \ltimes_A S) \bowtie S$, assuming that size(R) < size(S)

1. R → site 2
2. site 2 computes $R \bowtie S$.

R

A	C
1	6
2	7
4	8
5	9

site1

S

A	B
1	4
2	5
3	6
3	7

site2

S

A	B
1	4
2	5
3	6
3	7

$R \bowtie S$

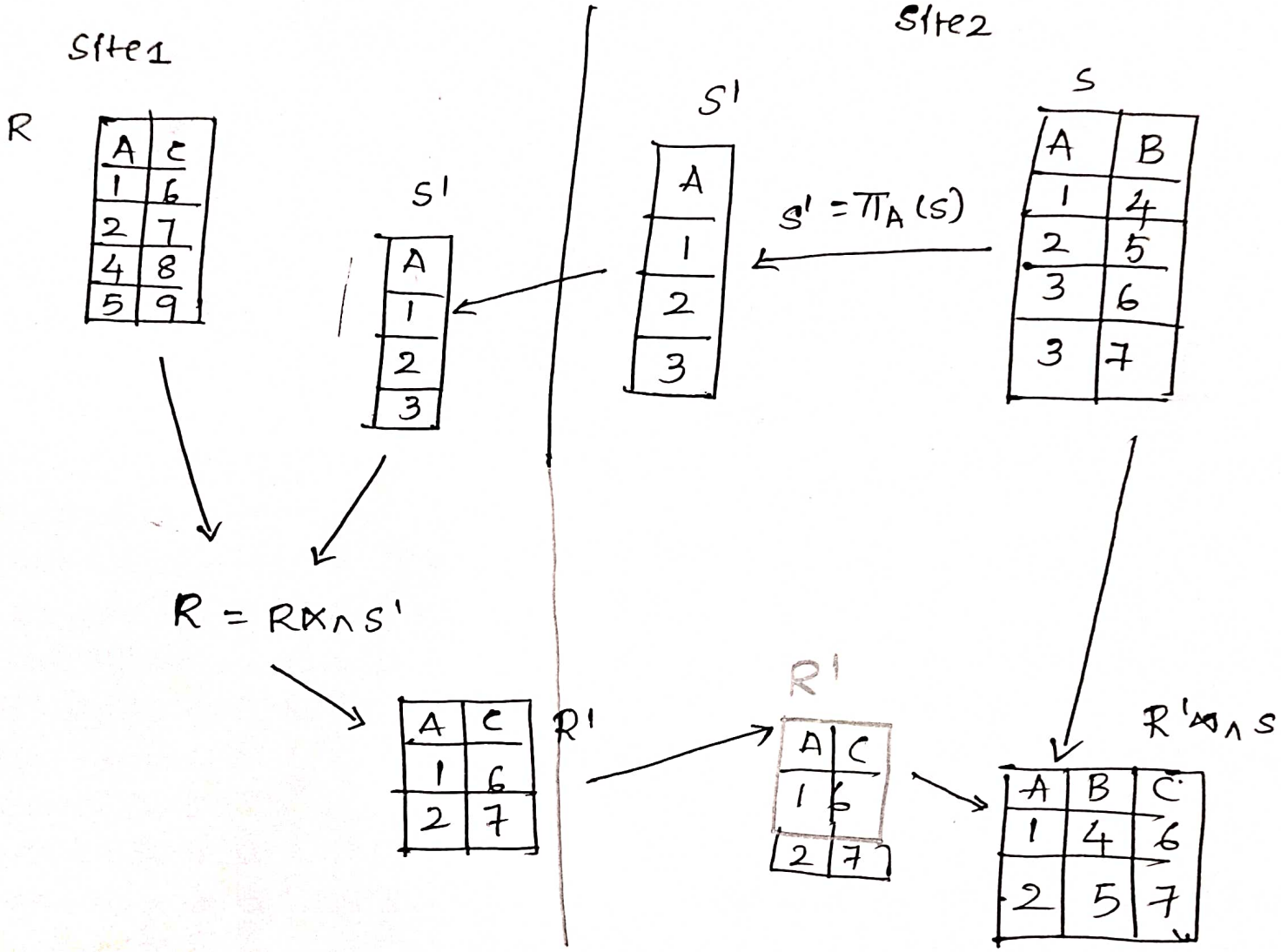
R

A	C
1	6
2	7
4	8
5	9

A	B	C
1	4	6
2	5	7

- Perform the semijoins $(R \ltimes S) \bowtie S$.

1. $S' = \Pi_A(S)$
2. $S' = $ site1
3. site1 computes $R' = R \ltimes S'$
4. $R' \rightarrow$ site2.



- Semijoin is better if

  Size $(\Pi_A(S))$ + size $(R \ltimes S)$ < size (R)

- The semijoin approach is better if the semijoin acts as a sufficient reducer (i.e. a few tuples of R participate in the join).

HILL CLIMBING ALGORITHM

- Let us denote initial strategy as ES_0.
- Then the optimizer splits ES_0 into two strategies, ES_1 followed by ES_2
  - ES_1 :ships one relation of join to the site of the other relation.
  - ES_2: These two relations are joined locally and the resulting relation is transmitted to the chosen result site
- If the cost of executing strategies ES1 and ES2, plus the cost of local join processing, is less than that of ES_0, then ES_0 is replaced in the schedule by ES_1 and ES_2.
  - If cost(ES_1) + cost(ES_2) + LC > cost (ES_0) select ES_0, else select ES_1 and ES_2.
- The process is then applied recursively to ES_1 and ES_2 until no more benefit can be gained.

Example . Let us illustrate the hill-climbing algorithm using the following query involving relations EMP, PAY, PROJ, and ASG of the engineering database:
"Find the salaries of engineers who work on the CAD/CAM project"

Schemas:
EMP(ENO, ENBAME, TITLE),
ASG(ENO, PNO, RESP, DUR),
PROJ(PNO, PNAME, BUDGET, LOC),
PAY(TITLE, SAL)

Statistics:

Relation	Size	Site
EMP	8	1
PAY	4	2
PROJ	1	3
ASG	10	4

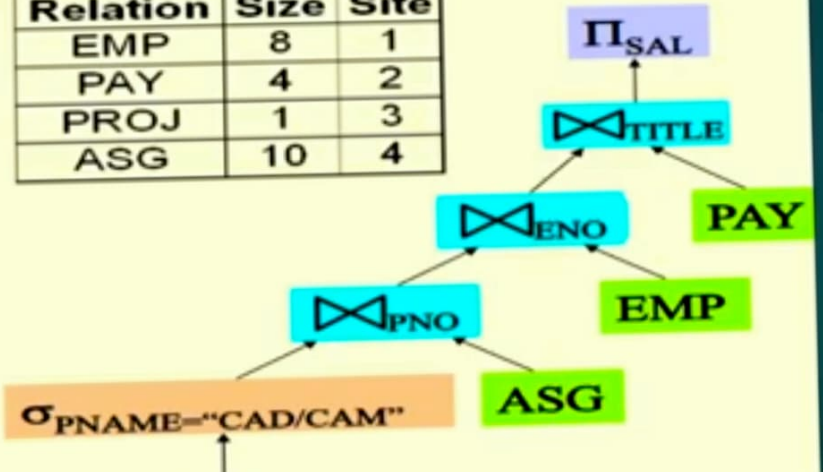
The query in relational algebra is

$$\Pi_{SAL}(PAY \bowtie_{TITLE} (EMP \bowtie_{ENO} (ASG \bowtie_{PNO} (\sigma_{PNAME="CAD/CAM"} (PROJ)))))$$

The query in relational algebra is

$\pi_{SAL}(PAY \bowtie_{TITLE} (EMP \bowtie_{ENO} (ASG \bowtie_{PNO} (\sigma_{PNAME="CAD/CAM"} (PROJ))))$

Relation	Size	Site
EMP	8	1
PAY	4	2
PROJ	1	3
ASG	10	4



Query tree

Statistics:

Relation	Size	Site
EMP	8	1
PAY	4	2
PROJ	1	3
ASG	10	4

- Alternative 1: Resulting site is site 1

Total cost = cost(PAY->Site1) + cost(ASG->Site1) + cost(PROJ->Site1)
= 4 + 10 + 1 = 15

- Alternative 2: Resulting site is site 4

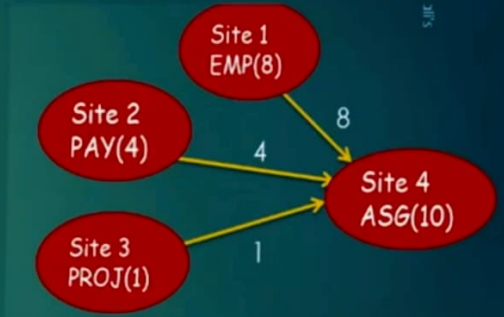
Total cost = 8 + 4 + 1 = 13

- Alternative 3: Resulting site is site 2

Total cost =8 + 10 + 1 = 19

- Alternative 4: Resulting site is site 3

Total cost = 8 + 4 + 10 = 22

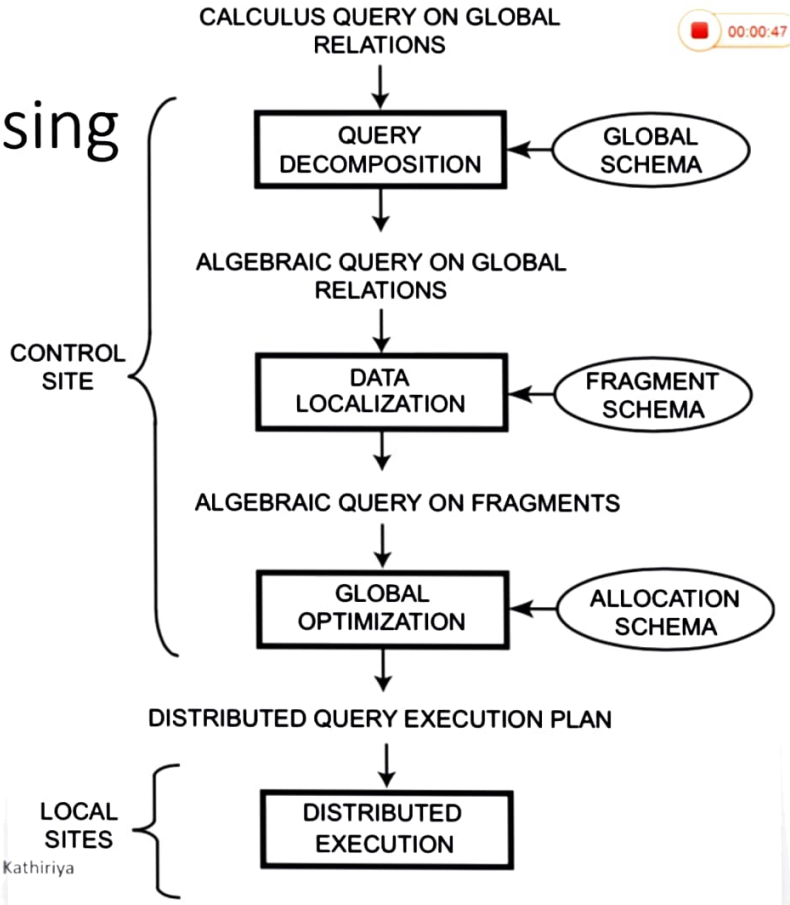


- Therefore ES_0 = EMP->Site4; PAY->Site4; PROJ->Site4

Ignore the local processing cost length of tuple is 1 for all relation

Layers of Query Processing

- Query processing has 4 layers:
  - Query Decomposition
  - Data Localization
  - Global Query Optimization
  - Distribution Query Execution



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.
  - Normalization
  - Analysis
  - Simplification
  - Restructure

Query Decomposition (Cont.)

- First, the calculus query is rewritten in a **normalized** form that is suitable for subsequent manipulation. Normalization of a query generally involves the manipulation of the query quantifiers and of the query qualification by applying logical operator priority.
- Second, the normalized query is **analyzed** semantically so that incorrect queries are detected and rejected as early as possible. Techniques to detect incorrect queries exist only for a subset of relational calculus. Typically, they use some sort of graph that captures the semantics of the query.

Query Decomposition (Cont.)

- Third, the correct query (still expressed in relational calculus) is **simplified**. One way to simplify a query is to eliminate redundant predicates. Note that redundant queries are likely to arise when a query is the result of system transformations applied to the user query. such transformations are used for performing semantic data control (views, protection, and semantic integrity control).
- Fourth, the calculus query is **restructured** as an algebraic query. The traditional way to do this transformation toward a "better" algebraic specification is to start with an initial algebraic query and transform it in order to find a "go
- The algebraic query generated by this layer is good in the sense that the worse executions are typically avoided.

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.
- Generating a query on fragments is done in two steps
  - First, the query is mapped into a fragment query by substituting each relation by its reconstruction program (also called materialization program).
  - Second, the fragment query is simplified and restructured to produce another "good" query.

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 .

Distribution Query Execution

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

Objectives of Distributed Query Processing

1. The main objectives of query processing in a distributed environment is to form 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 in local databases.

2. An important point 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.

3. The good measure of resource consumption are:

   i. The total cost that will be incurred in processing the query. It is the dome of all times incurred in processing the operations of the query at various sites and intrinsic communication.

ii. The resource time of the query. This is the time elapsed for executing the query. Since operations can be executed in parallel at different sited, the response time of a query may be significantly less than its cost.

4. Obviously the total cost should be minimized.

   i. In a distributed system, the total cost to be minimized includes CPU, I\O, and communication costs. This cost can be minimized by reducing the number of I\O operation through fast access methods to the data and efficient use of main memory. The communication cost is the time needed for exchanging the data between sited participating in the execution of the query.

   ii. In centralized systems, only CPU and I\O cost have to be considered.

Query Decomposition

- Process of minimizing the resource usage.

Aims of Query Decomposition

- To transform a high-level query into a Relational Algebra query

&

- To check the query is syntactically and semantically correct

- It is efficient way to retrieve data from database.

For Example

```
select * from staff s, branch b
where s.branchNo=b.branchNo And
(s.position='Manager' And b.city='London');
```

- ✓ Three equivalent relational algebra queries

1. σ(position='Manager')^(staff.branchNo=branch.branchNo)(staff x branch)
   - Suppose staff have 1000 records and branch have 50 records

For Example(cont:)

- This query calculates the Cartesian product of staff and branch, (1000 + 50) disk accesses to read and creates relation with (1000*50) tuples. We then have to read each of these tuples again to test them against the selection predicate so it gives total cost of :

  (1000+50)+2*(1000*50)=101050 disk access

2. σ(position='Manager')^(city='London')(staff∞staff.branchNo=branch.branchNo branch)

   Gives total cost of:

   - 2*1000+(1000+50)=3050 disk accesses

3. (σposition='manager'(staff))∞staff.branchNo=branch.branchNo(σcity='London'(branch))

   Gives total cost of:

   - 1000+2*50+5+(50+5)=1160 disk accesses

Stages of Query Decomposition

1. Analysis
2. Normalization
3. Semantic Analysis
4. Simplification Or Redundancy eliminator
5. Query Restructuring

1. Analysis

In this stage query is syntactically analyzed

- ✓ Verifies that the relations and attributes specified in the query are defined in the system catalog
- ✓ Verifies that any operations applied to database objects are appropriate for the object type

For Example

- SELECT staffNumber
  FROM staff
  where position >10;

This query would be rejected on two grounds

1. in SELECT list the attribute staffNumber is not defended for staff relation(should be staffNo)

2. in WHERE clause comparison '>10' is incompatible with the data type position, which is a variable character string

After completion this stage ,high level query has been transformed internal representation that is some kind of query tree.

Query Tree

- A leaf node is created for each base relation in query
- Each intermediate relation produced by a relational algebra operation
- The Root of the tree represents the result of the query
- The sequence of operation is directed from the leaves to the root

2. Normalization

- It normalizes query for easy manipulation

- Arbitrarily complex predicate (in SQL, the WHERE condition) can be converted into one of two forms by applying few transformation rules:

1. Conjunctive Normal form:- A sequence of conjuncts that are connected with the ^ (AND) operator. Each conjunct contains one or more terms connected by the v (OR) operator

   For Example :-

   (position='manager' v salary > 20000) ^ branchNo='B003'

   A conjunctive selection contains only those tuples that satisfy all conjuncts.

2. Disjunctive Normal form:- A sequence of disjuncts that are connected with the v (OR) operator. Each disjunct contains one or more terms connected by the ^ (AND) operator

For Example : we can rewrite the above conjunctive normal form into disjunctive

(position='manager'^ branchNo='B003') v (salary >20000 ^ branchNo='B003')

A disjunctive selection contains those tuples formed by the union of all tuples that satisfy the disjuncts

3. Semantic Analysis

- It reject normalized queries that are incorrectly formulated or contradictory.
- A query is incorrectly formulated if components do not contribute to the generation of the result, a query is contradictory if its predicate cannot be satisfied by any tuple.
- For Example: the predicate (position='manager' ^ position='assistant')on staff relation is contradictory b/c a person can not the both manager and assistant simultaneously
- However, the predicate((position='manager' ^position='assistant') v salary >2000) could be simplified

Elimination Of Redundancy

- ✓ Simplifying the qualification of user query to eliminate redundancy
- ✓ Transform the query to a semantically equivalent but more easily and efficiently computed form
- ✓ Queries on relation that satisfy certain integrity and security constraints (access restriction)

Example

- **Before elimination of Redundancy**

```
SELECT  Position
FROM staff s,  branch b
WHERE (NOT(b.Position = 'manager')
AND (b.Position = 'manager' OR b.Position='assistant')
AND NOT(b.Position = 'assistant'))
OR (b.BID = s.SID
AND s.Name = 'Ali')
```

- **After Elimination of Redundancy**

```
SELECT  b.Position
                FROM staff s, branch b
WHERE b.BID = s.SID    AND s.Name = 'ali
```

Query Restructuring

- The final stage of query decomposition.
- The query is restructured to provide a more efficient implementation.

Advantages & Disadvantages

Advantages:-

- all information required to select an optimum strategy is up to date

Disadvantage:

- It takes time to decompose the query has to be parsed, validated, and optimized before it can be executed

Conclusion

- Query is optimized ,data can be retrieve easily and more efficiently

Distributed System R* Algorithm

The System R* query optimization algorithm is an extension of the System R query

Optimization algorithm with the following main characteristics:

- Only the whole relations can be distributed, i.e., fragmentation and replication is not considered
- Query compilation is a distributed task, coordinated by a master site, where the query is initiated
- Master site makes all inter-site decisions, e.g., selection of the execution sites, join ordering, method of data transfer, ...
- The local sites do the intra-site (local) optimizations, e.g., local joins, access paths
- Join ordering and data transfer between different sites are the most critical issues to be considered by the master site

- Four main join strategies for R ⋈ S:

- R is outer relation(EXTERNAL)
- S is inner relation(INTERNAL)

- Notation:

- LT denotes local processing time
- CT denotes communication time
- s denotes the average number of S-tuples that match one tuple of R-tuple

Strategy 1: Ship the entire outer relation to the site of the inner relation, i.e.,

- Retrieve outer tuples
- Send them to the inner relation site
- Join them as they arrive

Total cost = LT(retrieve card(R) tuples from R) + CT(size(R)) + LT(retrieve s tuples from S)* card(R)

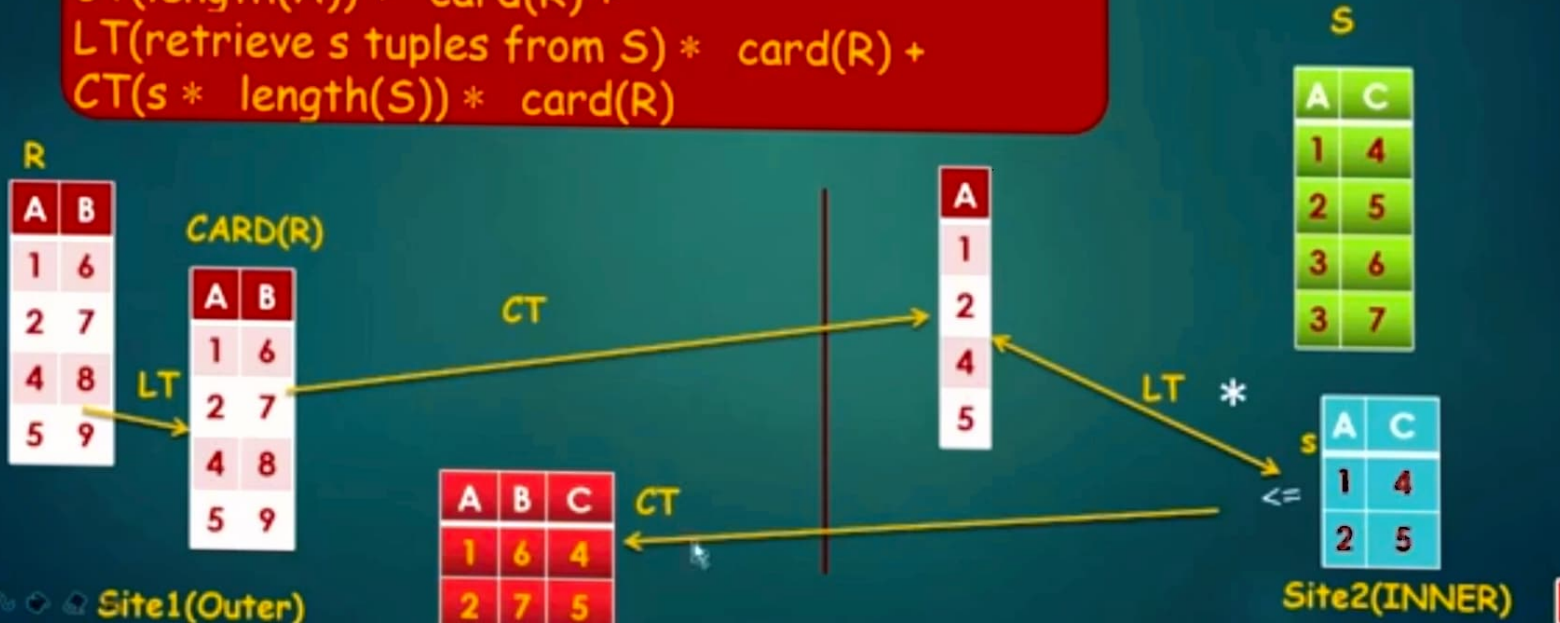
- Strategy 2: Ship the entire inner relation to the site of the outer relation. We cannot join as they arrive; they need to be stored.
- The inner relation S need to be stored in a temporary relation

Total cost = LT(retrieve card(S) tuples from S) +
CT(size(S)) +
LT(store card(S) tuples in T) +
LT(retrieve card(R) tuples from R) +
LT(retrieve s tuples from T) * card(R)

- Strategy 3:Fetch tuples of the internal relation as needed for each tuple of the external relation

- For each R-tuple, the join attribute A is sent to the site of S
- Then the s tuples of S which match that value are retrieved and sent to the site of R to be joined as they arrive.

Total cost = LT(retrieve card(R) tuples from R) +
CT(length(A)) * card(R) +
LT(retrieve s tuples from S) * card(R) +
CT(s * length(S)) * card(R)



• Strategy 4: Move both relations to a third site and compute the join there.

- the internal relation is first moved to a third site i.e the inner relation S is first moved to a third site and stored in a temporary relationT
- Then the outer relation is moved to the third site and its tuples are joined as they arrive

Total cost = LT(retrieve card(S) tuples from S) +
CT(size(S)) +
LT(store card(S) tuples in T) +
LT(retrieve card(R) tuples from R) +
CT(size(R)) +
LT(retrieve s tuples from T) * card(R)

Overview of SDD1 Algorithm

- System for Distributed Databases
- It is a distributed query optimization algorithm
- It is based on "Hill Climbing" algorithm which does not use semi join nor does it assume data replication and fragmentation
- Its a greedy approach it finds the local minimum and iteratively tries to improve it. It may not reach global solution all the time

SDD1 Algorithm(2)

- The main step of this algorithm consists of determining and ordering beneficial semijoins, that is semijoins whose cost is less than their benefit
- The cost of a semijoin is that of transferring the semijoin attributes A
- $\text{Cost}(R\ SJ_A\ S) = T_{MSG} + T_{TR} * \text{size}(\text{Project}_A(S))$
- $\text{Benefit}(R\ SJ_A\ S) = (1 - SF_{SJ}(S.A)) * \text{size}(R) * T_{TR}$

SDD Algorithm(3)

- Select the most beneficial semijoin and ignore the rest i.e., MAX(Benefit-Cost)
- Perform table statistics
- Exclude the selected semijoin and do with the exhaustive check on all the other semijoins where Benefit > Cost

SDD1

- SDD1 takes

Query graph

Location of relations

Relation statistics as inputs

- Produces: Global strategy for executing the query

Implementation Details

- Programming language : Java
- Used Swings Components in Java

- Input : Relations and its profile
- Output : Gives the best strategy and the assembly site

----------Iteration 3----------

Join	Benefit	Cost	Benefit-Cost
R2 SJ R3 ON B	540.0	80.0	460.0
R3 SJ R2 ON B	0.0	400.0	-400.0

Best Semi Join Removed is : R2 SJ R3 ON B

System R (Centralized) Algorithm

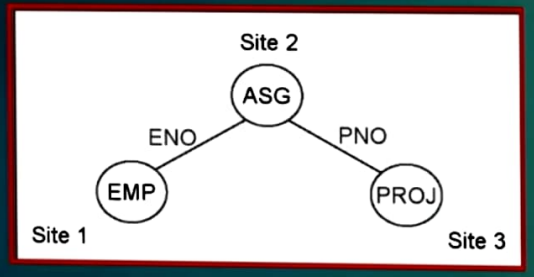
- Simple (one relation) queries are executed according to the best access path.
- Execute joins
  - Determine the possible ordering of joins
  - Determine the cost of each ordering
  - Choose the join ordering with the minimal cost
- For joins, two join methods are considered:
  - Nested loops
  - Merge join

- Example: "Names of employees working on the CAD/CAM project"

$$P\ ROJ \bowtie_{P\ NO} ASG \bowtie_{ENO} EMP$$

Indexes

* EMP has an index on ENO
* ASG has an index on PNO
* PROJ has an index on PNO and an index on PNAME



Join graph

EMP

ENO	ENAME	TITLE
E1	J. Doe	Elect. Eng
E2	M. Smith	Syst. Anal.
E3	A. Lee	Mech. Eng.
E4	J. Miller	Programmer
E5	B. Casey	Syst. Anal.
E6	L. Chu	Elect. Eng.
E7	R. Davis	Mech. Eng.
E8	J. Jones	Syst. Anal.

ASG

ENO	PNO	RESP	DUR
E1	P1	Manager	12
E2	P1	Analyst	24
E2	P2	Analyst	6
E3	P3	Consultant	10
E3	P4	Engineer	48
E4	P2	Programmer	18
E5	P2	Manager	24
E6	P4	Manager	48
E7	P3	Engineer	36
E8	P3	Manager	40

PROJ

PNO	PNAME	BUDGET
P1	Instrumentation	150000
P2	Database Develop.	135000
P3	CAD/CAM	250000
P4	Maintenance	310000

PAY

TITLE	SAL
Elect. Eng.	40000
Syst. Anal.	34000
Mech. Eng.	27000
Programmer	24000

- The label ENO on edge EMP-ASG stands for the predicate EMP.ENO=ASG.ENO
- The label PNO on edge ASG-PROJ stands for the predicate ASG.PNO=PROJ.PNO.

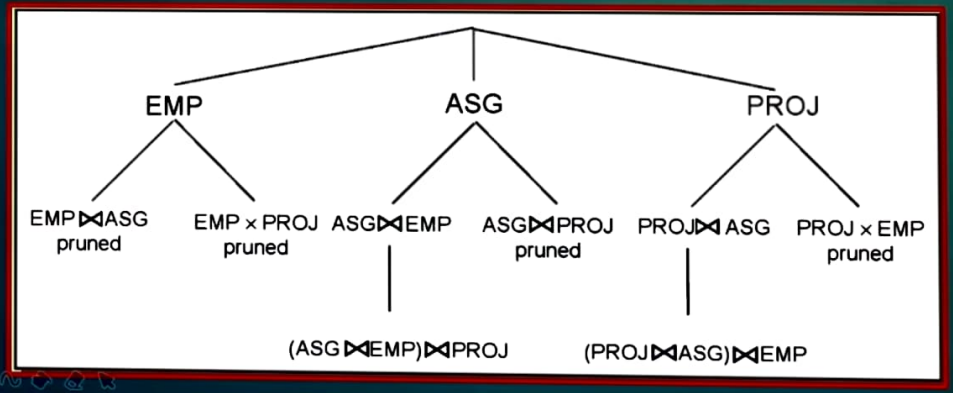
- Example: "Names of employees working on the CAD/CAM project"

$PROJ \bowtie_{PNO} ASG \bowtie_{ENO} EMP$

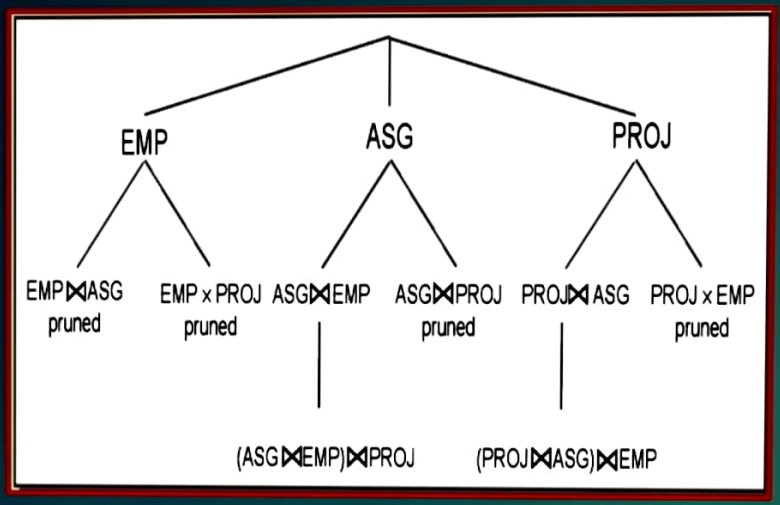
Step 1 - Select the best single-relation access paths

- EMP: sequential scan (because there is no selection on EMP)
- ASG: sequential scan (because there is no selection on ASG)
- PROJ: index on PNAME (because there is a selection on PROJ based on PNAME)

Step 2 - Select the best join ordering for each relation



Note that the maximum number of join orders is 3!



EMP

ENO	ENAME	TITLE
E1	J. Doe	Elect. Eng
E2	M. Smith	Syst. Anal.
E3	A. Lee	Mech. Eng.
E4	J. Miller	Programmer
E5	B. Casey	Syst. Anal.
E6	L. Chu	Elect. Eng.
E7	R. Davis	Mech. Eng.
E8	J. Jones	Syst. Anal.

ASG

ENO	PNO	RESP	DUR
E1	P1	Manager	12
E2	P1	Analyst	24
E2	P2	Analyst	6
E3	P3	Consultant	10
E3	P4	Engineer	48
E4	P2	Programmer	18
E5	P2	Manager	24
E6	P4	Manager	48
E7	P3	Engineer	36
E8	P3	Manager	40

PROJ

PNO	PNAME	BUDGET
P1	Instrumentation	150000
P2	Database Develop.	135000
P3	CAD/CAM	250000
P4	Maintenance	310000

PAY

TITLE	SAL
Elect. Eng.	40000
Syst. Anal.	34000
Mech. Eng.	27000
Programmer	24000

- The operations marked "pruned" are dynamically eliminated.
- (EMP × PROJ) and (PROJ × EMP) are pruned because they are Cartesian products
- We assume that (EMP ⋈ ASG) and (ASG ⋈ PROJ) have a cost higher than (ASG ⋈ EMP) and (PROJ ⋈ ASG), respectively.