DISTRIBUTED QUERY PROCESSING

UNIT-2

Query Processing

Query processing is a method or technique where high level user query is processed by the query processor to execute low level data commands.

Query Processing Components

1) Query language
2) Query execution methodology
3) Query optimization

Query Decomposition(Step-1)

The query on distributed relations of global schema is broken into algrbraic query. It consists of

1) <u>Normalization</u>

- manipulate query quantifiers and qualification

2) <u>Analysis</u>

- detect and reject “incorrect” queries
- possible for only a subset of relational calculus

3) <u>Simplification</u>

- eliminate redundant predicates

4) <u>Restructuring</u>

- calculus query / algebraic query
- more than one translation is possible
- use transformation rules

Normalization

a) Lexical and syntactic analysis

- check validity (similar to compilers)
- check for attributes and relations

b) Put into normal form

- OR's mapped into union
- AND's mapped into join or selection

Analysis

Analysis is done to prove false for incorrect queries.

1) <u>Type incorrect</u>

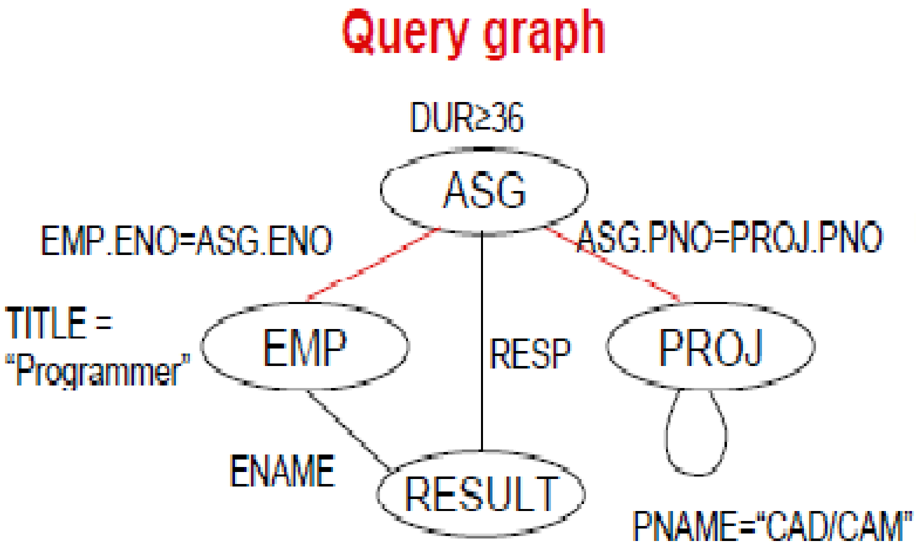
- If any of its attribute or relation names are not defined in the global schema.
- If operations are applied to attributes of the wrong type.

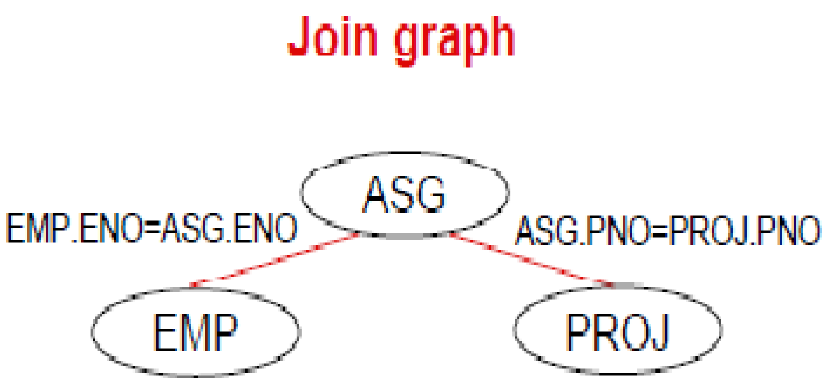
2.) <u>Semantically incorrect</u>

- Components do not contribute in any way to the generation of the result

Analysis(example)

```
SELECT   ENAME,RESP
FROM     EMP, ASG, PROJ
WHERE    EMP.ENO = ASG.ENO
AND      ASG.PNO = PROJ.PNO
AND      PNAME = "CAD/CAM"
AND      DUR ≥ 36
AND      TITLE = "Programmer"
```





Analysis(example)

- If the query graph is not connected, the query is wrong.

SELECT ENAME,RESP

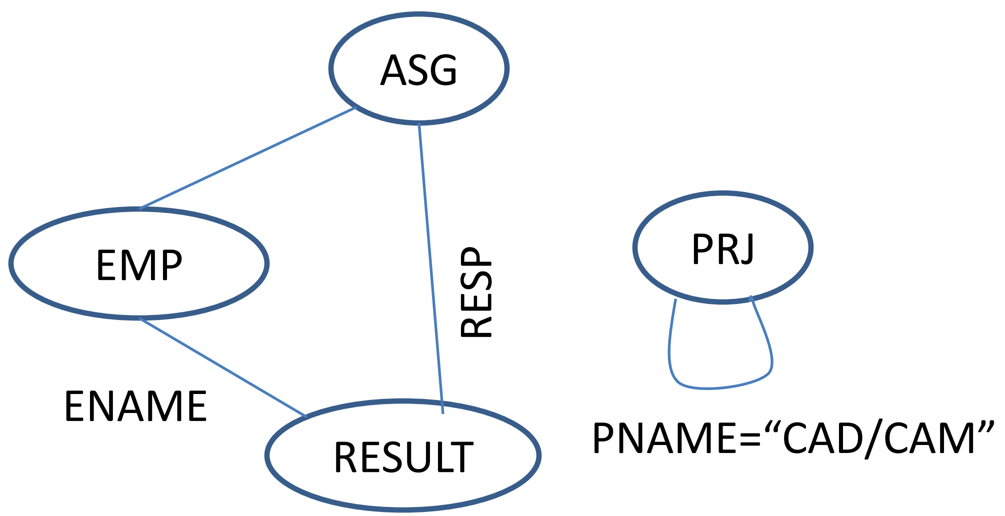
FROM EMP, ASG, PROJ

WHERE EMP.ENO = ASG.ENO

AND PNAME = "CAD/CAM"

AND DUR ≥ 36

AND TITLE = "Programmer“



Elimination of redundancy(Simplification)

To eliminate redundancy we must use transformation rules.

1)idempotency rules

- $p1 \wedge \neg(p1) \Leftrightarrow false$
- $p1 \wedge (p1 \vee p2) \Leftrightarrow p1$
- $p1 \vee false \Leftrightarrow p1$

2)Application of transitivity

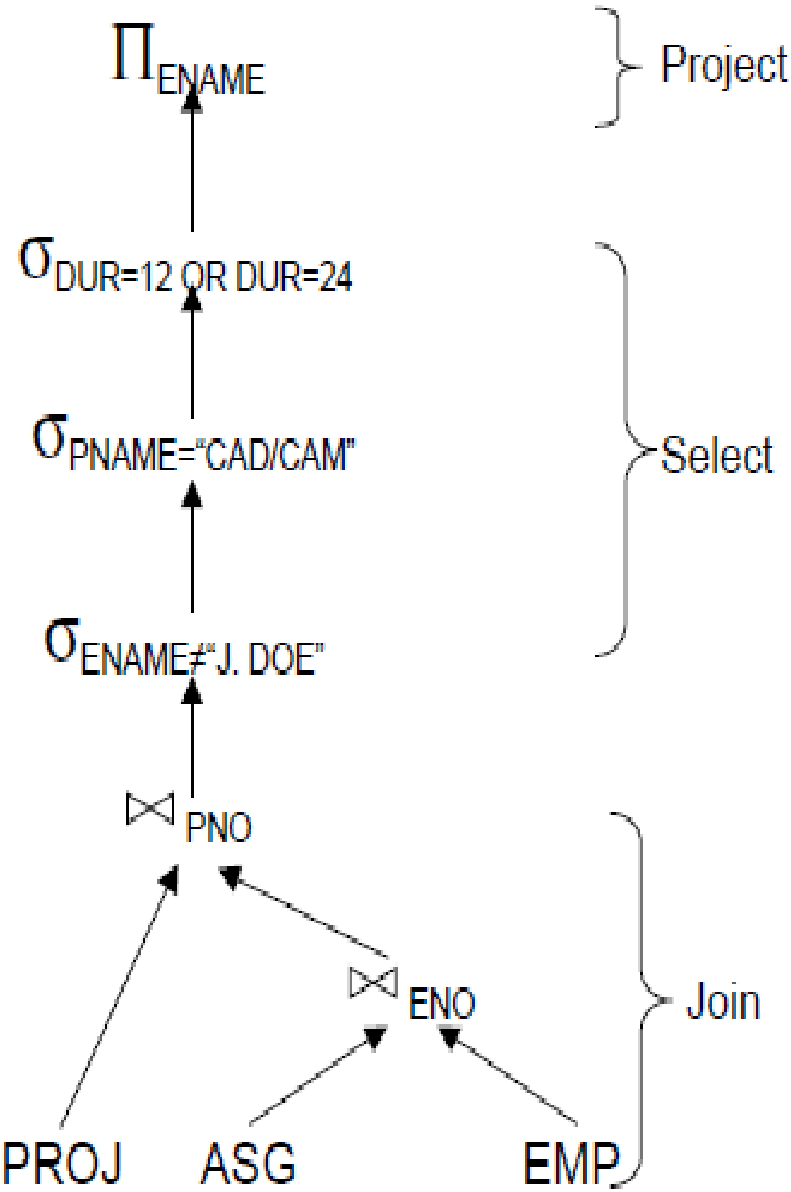
3)Use of integrity rules

Rewriting(Restructuring)

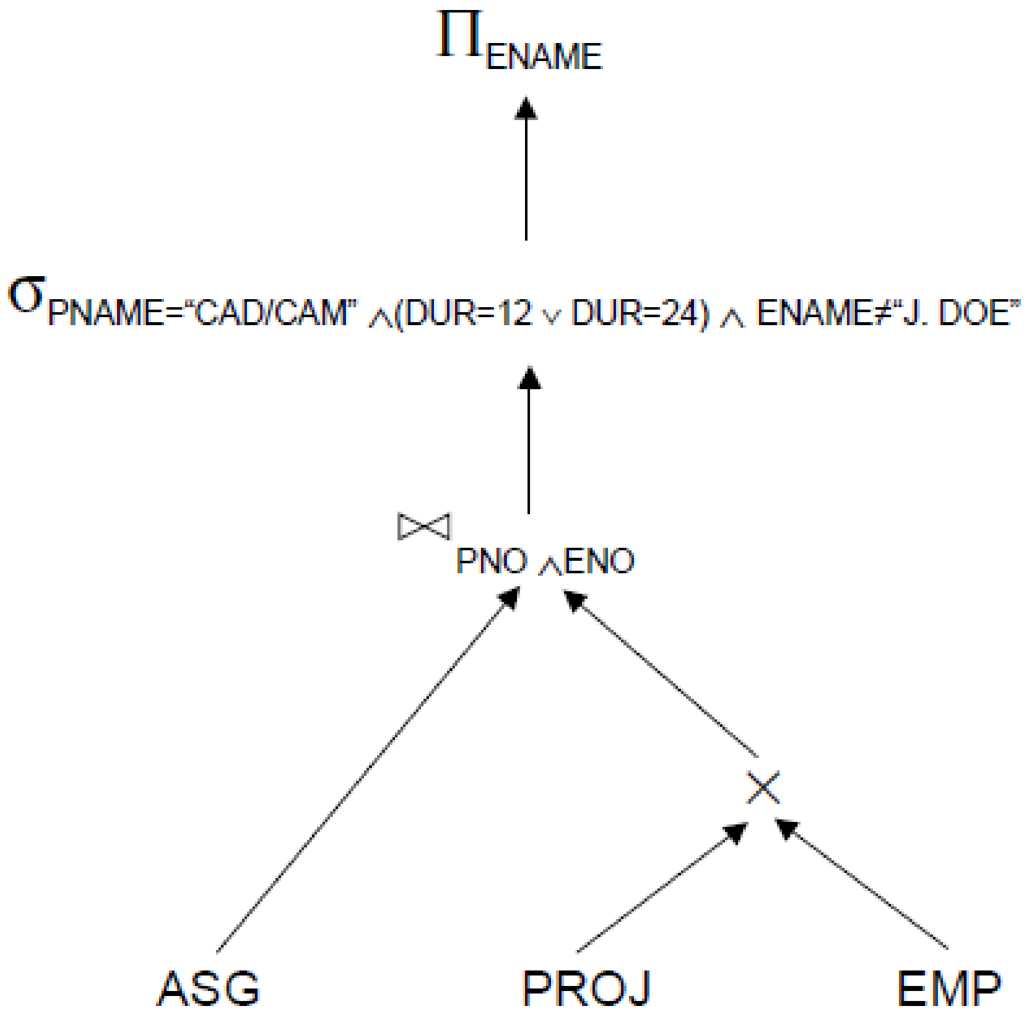
- Convert relational calculus to relational algebra
- Make use of query trees
- Example

  Find the names of employees other than J. Doe who worked on the CAD/CAM project for either 1 or 2 years.

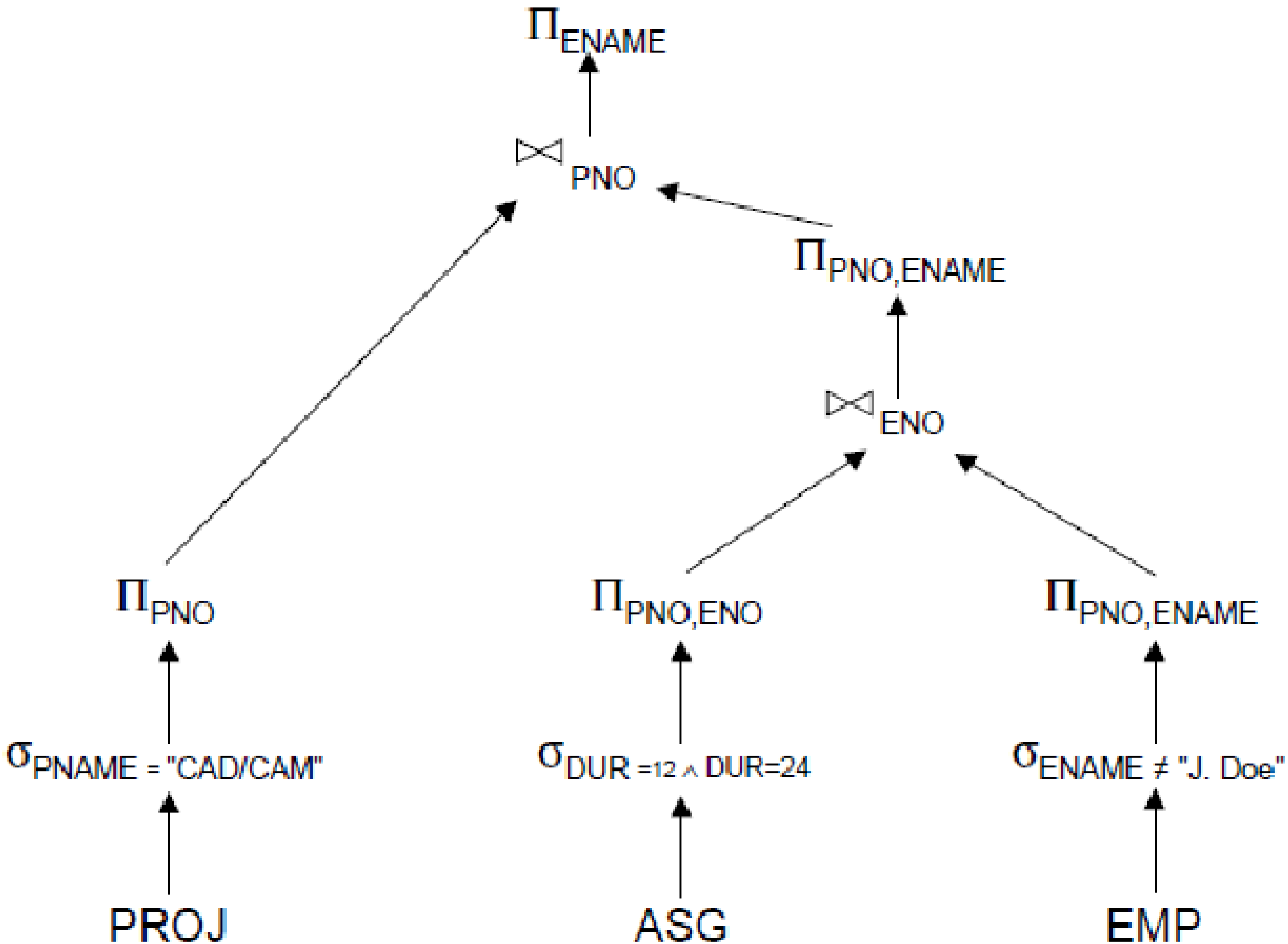
```
SELECT  ENAME
FROM    EMP, ASG, PROJ
WHERE   EMP.ENO = ASG.ENO
AND     ASG.PNO = PROJ.PNO
AND     ENAME ≠ "J. Doe"
AND     PNAME = "CAD/CAM"
AND     (DUR = 12 OR DUR = 24)
```



Equivalent query



Rewriting the query



Location of Distributed Data (Step-2)

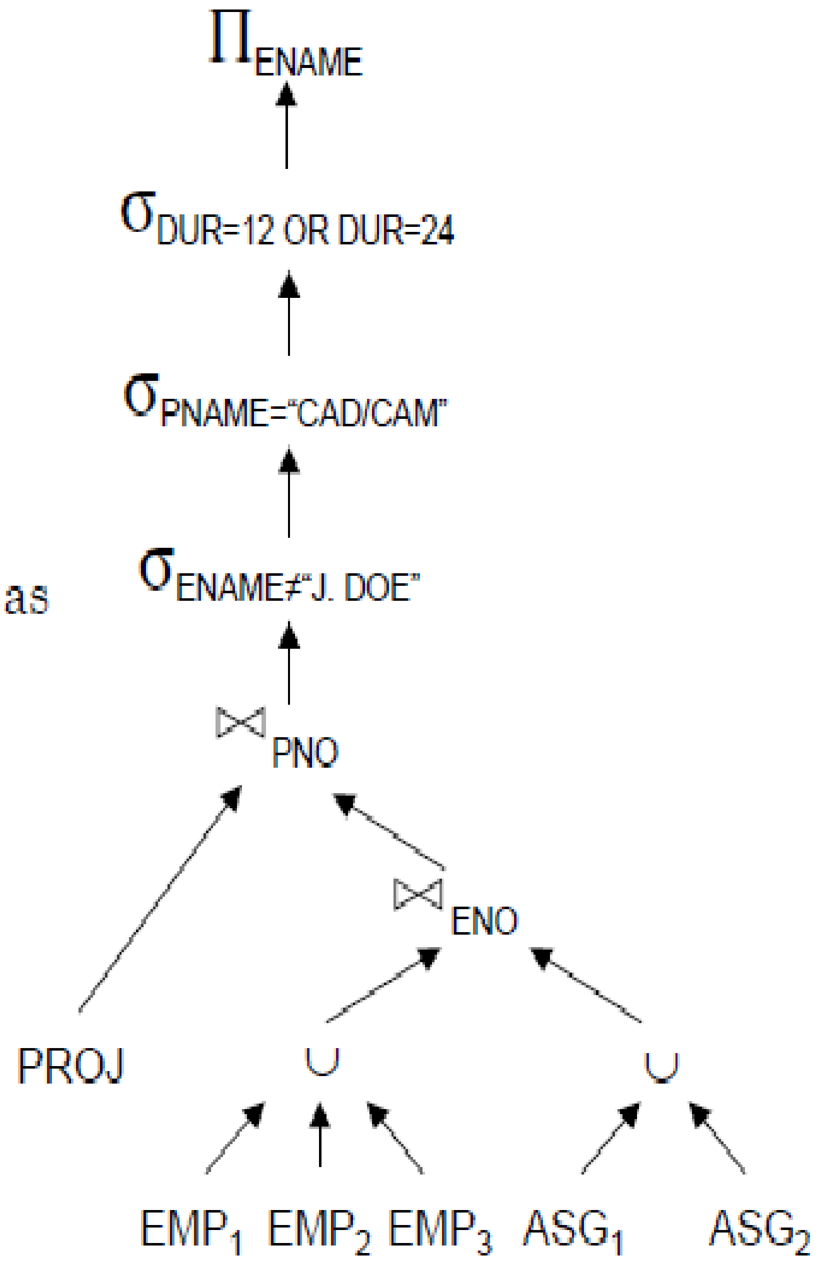
- It is also known as data localization.
- It is the process of determining which fragments of the database objects are involved.
- It is done by substituting the materialization program for the global query as shown in the example.

Example

Assume

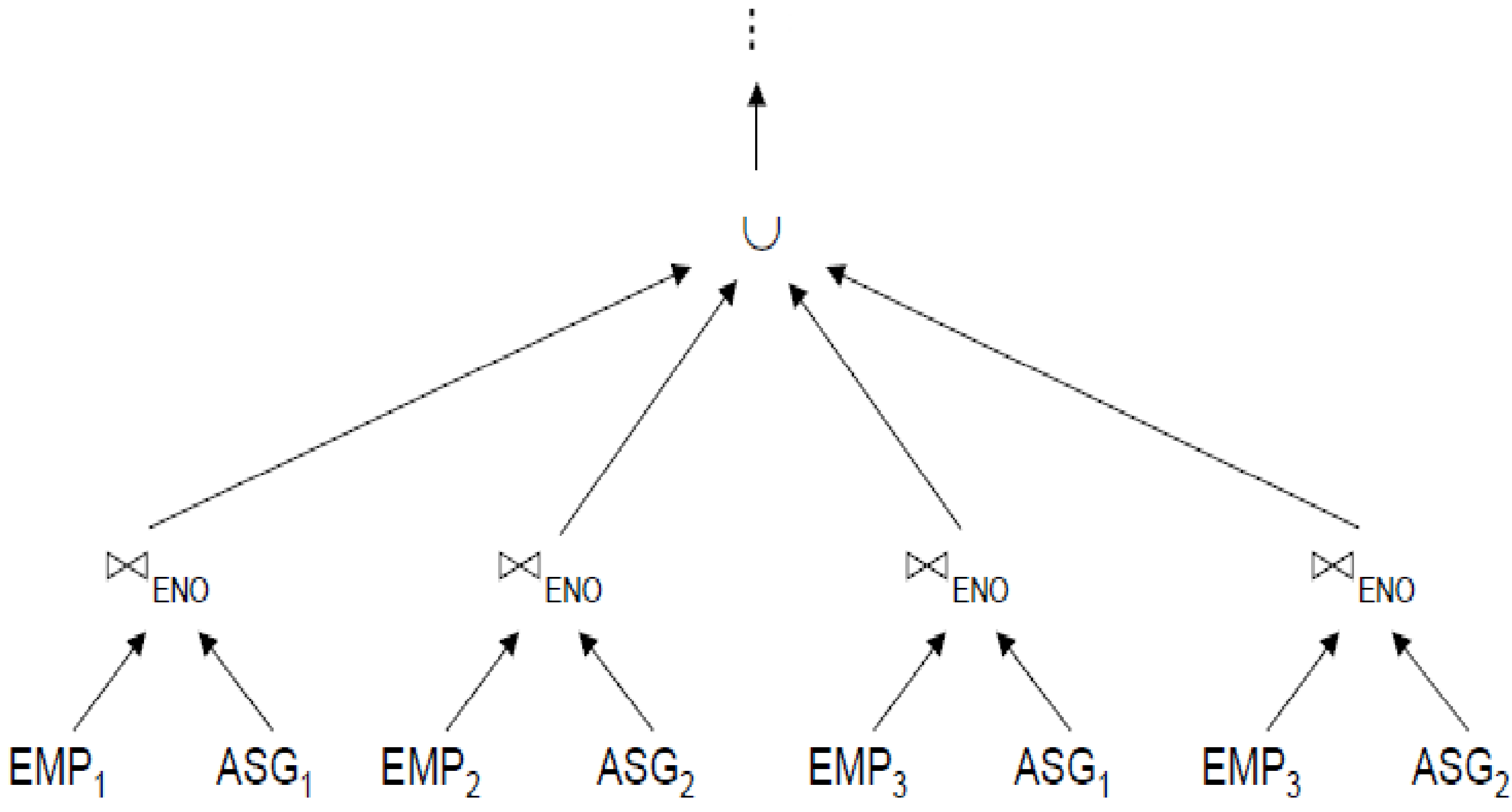
- EMP is fragmented into EMP_1, EMP_2, EMP_3 as follows:
  - $EMP_1=\sigma_{ENO\leq\text{"E3"}}(EMP)$
  - $EMP_2= \sigma_{\text{"E3"}<ENO\leq\text{"E6"}}(EMP)$
  - $EMP_3=\sigma_{ENO\geq\text{"E6"}}(EMP)$
- ASG fragmented into ASG_1 and ASG_2 as follows:
  - $ASG_1=\sigma_{ENO\leq\text{"E3"}}(ASG)$
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

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



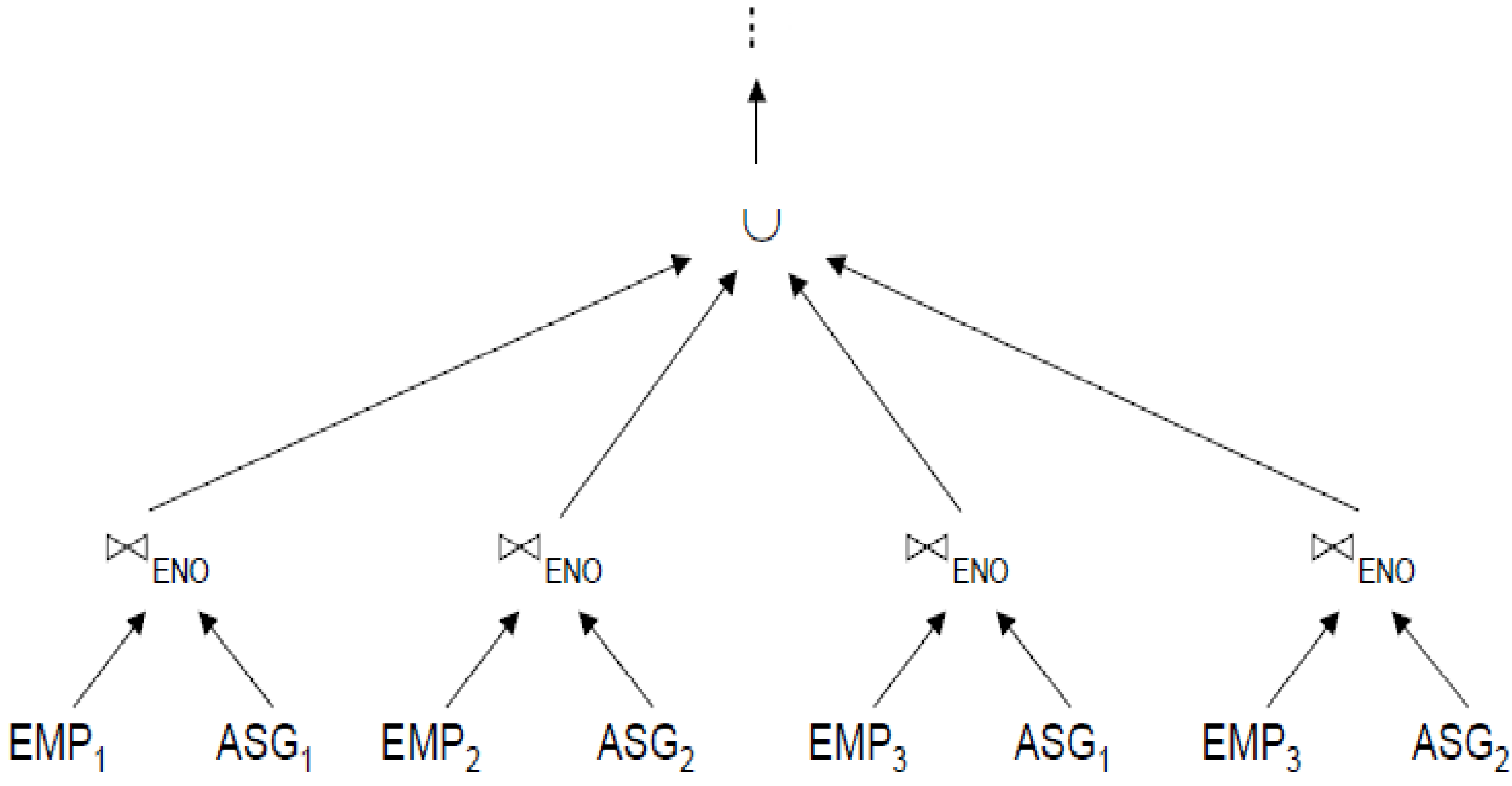
Data localization

- Thus data localization provides parallelism.



Data localization

- Thus data localization eliminates unnecessary works.

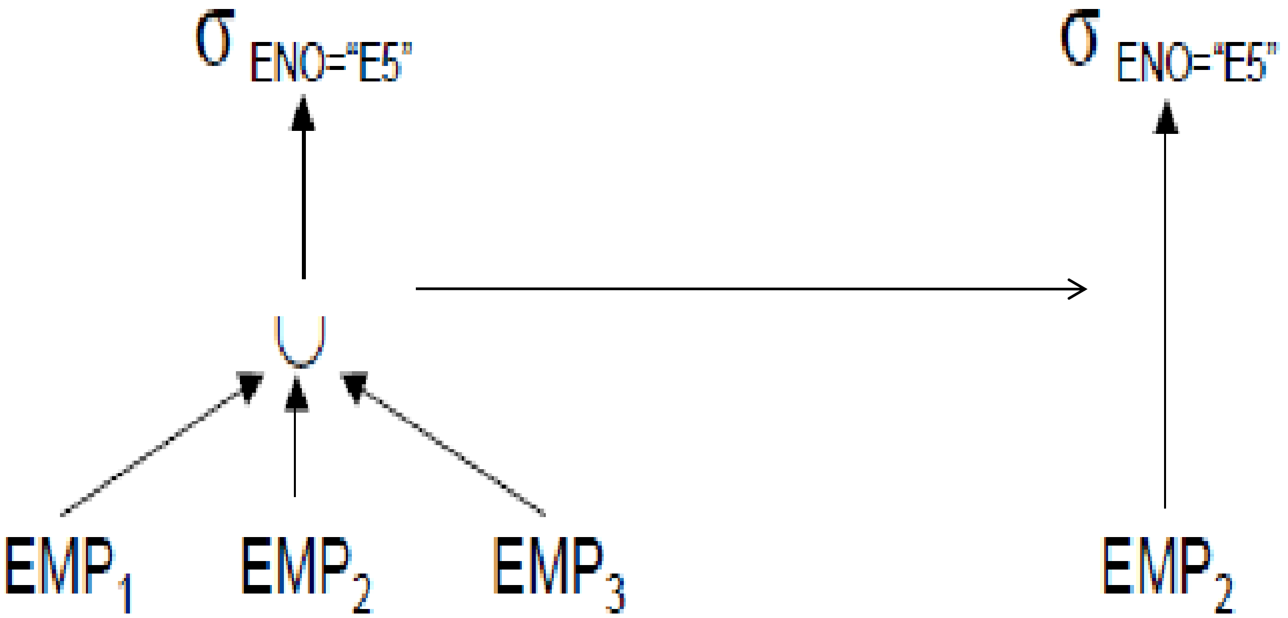


Reduction for PHF

Consider the relation

SELECT * FROM EMP

WHERE ENO="E5"



Reduction for VF

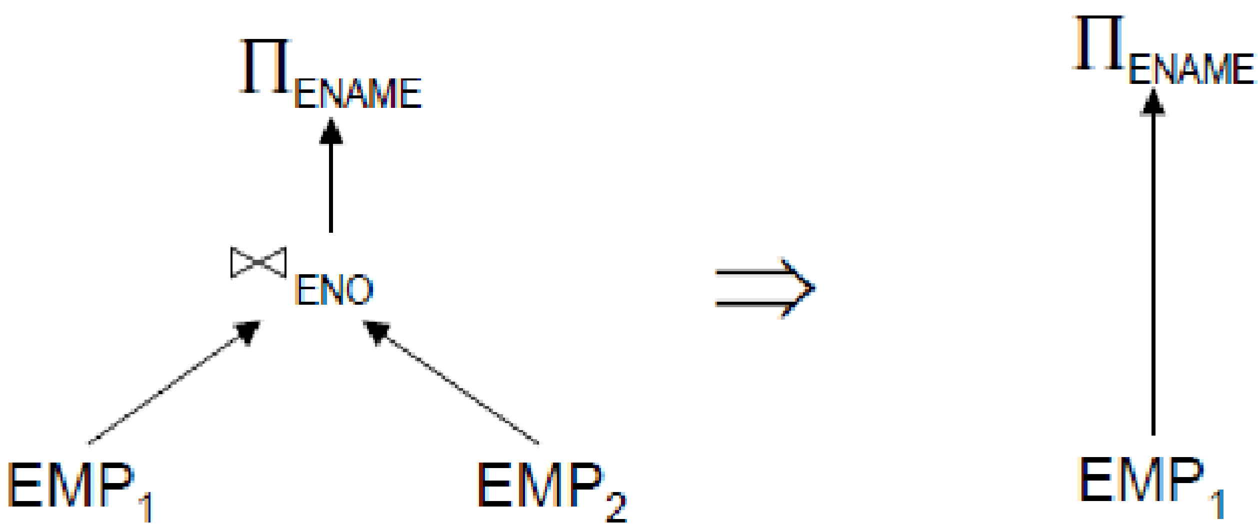
- Example:

  EMP1= $\Pi_{ENO,ENAME}$ (EMP);

  EMP2= $\Pi_{ENO,TITLE}$ (EMP)

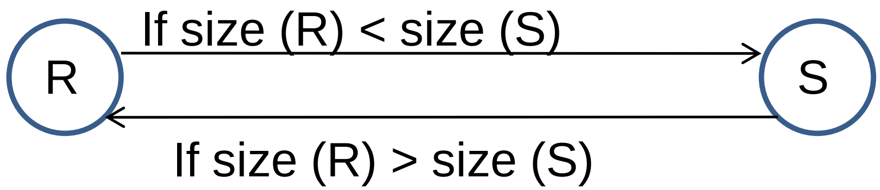
  SELECT ENAME

  FROM EMP



Join Ordering

- Consider two relations only

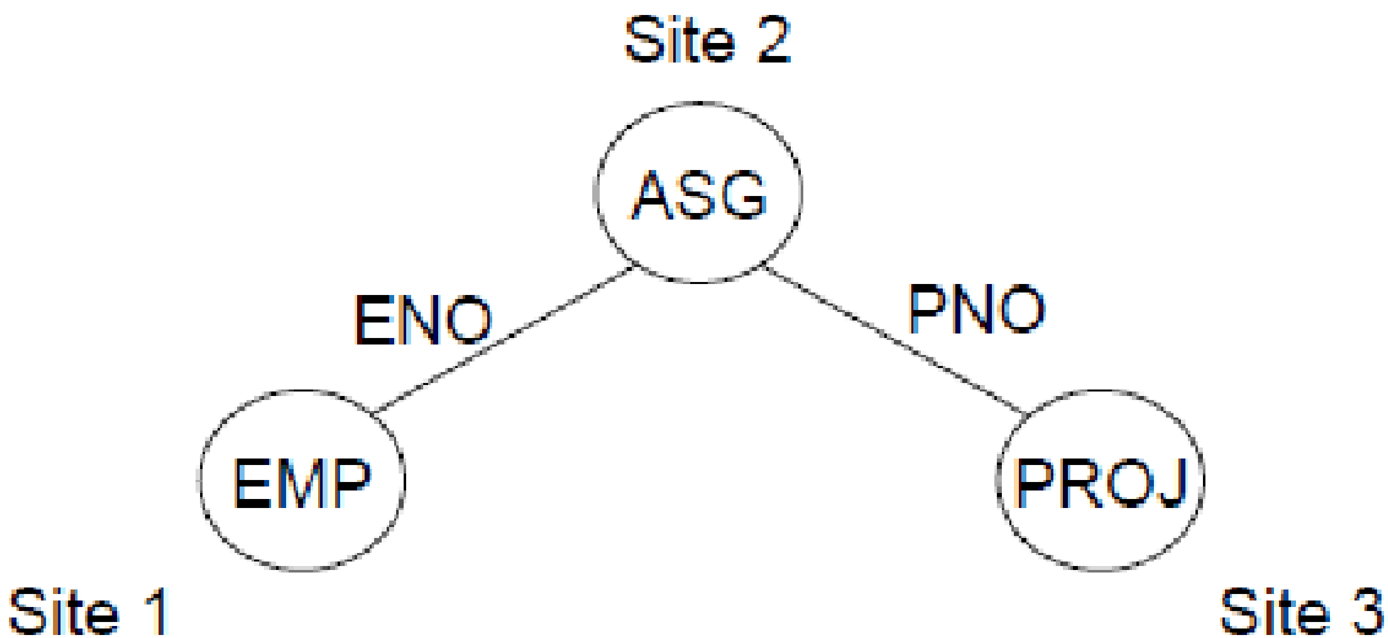


- Multiple relations more difficult because too many alternatives.
- Compute the cost of all alternatives and select the best one.
- Necessary to compute the size of intermediate relations which is difficult.
- Use heuristics

Join Ordering

Consider

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



Join Ordering Example

Execution alternatives:

1.)EMP → Site 2

Site 2 computes EMP'=EMP$\bowtie$ASG

EMP' → Site 3

Site 3 computes EMP'$\bowtie$PROJ

2.) ASG → Site 1

Site 1 computes EMP'=EMP$\bowtie$ASG

EMP' → Site 3

Site 3 computes EMP'$\bowtie$PROJ

3.) ASG → Site 3

Site 3 computes ASG'=ASG $\bowtie$PROJ

ASG' → Site 1

Site 1 computes ASG'$\bowtie$ EMP

Join Ordering Example

4.) PROJ → Site 2

Site 2 computes PROJ'=PROJ $\bowtie$ ASG

PROJ' → Site 1

Site 1 computes PROJ' $\bowtie$ EMP

5.) EMP → Site 2

PROJ → Site 2

Site 2 computes EMP $\bowtie$ PROJ $\bowtie$ ASG

Semi-Join Strategy/Algorithm

- Perform the join
  - send *R to Site 2*
  - Site 2 computes $R \bowtie_A S$
- Consider semijoin $(R \ltimes_A S) \bowtie_A S$
  - $S' \leftarrow \Pi_A(S)$
  - $S' \rightarrow$ *Site 1*
  - Site 1 computes $R' = R \ltimes_A S'$
  - $R' \rightarrow$ *Site 2*
  - Site 2 computes $R' \bowtie_A S$
- Semi-join is better if

$$\boldsymbol{size(\Pi_A(S)) + size(R \ltimes_A S)) < size(R)}$$

Distributed Query Optimization Methods

1) R * Algorithm
2) SDD-1 Algorithm
3) Hill Climbing Algorithm

R * Algorithm

- Cost function includes local processing as well as transmission.
- Considers only joins.
- Exhaustive(Full) search.
- Published papers provide solutions to handle horizontal and vertical fragmentations but the implemented prototype does not .

R * Algorithm

While performing joins it uses either

a)Whole Relationship

- larger data transfer
- smaller number of messages
- better if relations are small.

b)Fetch relations as needed

- data transfer per message is minimal
- better if relations are large and the selectivity is good.

R * Algorithm

1. <u>Move outer relation tuples to the site of the inner relation</u>

(a)Retrieve outer tuples

(b)Send them to the inner relation site

(c) Join them as they arrive

2. <u>Move inner relation to the site of outer relation</u>

- cannot join as they arrive; they need to be stored

3. <u>Move both inner and outer relations to another site</u>

4. <u>Fetch inner tuples as needed</u>

SDD-1 Algorithm

- Based on the Hill Climbing Algorithm
  - Semi-joins
  - No replication
  - No fragmentation
  - Cost of transferring the result to the user site from the final result site is not considered
  - Can minimize either total time or response time.

Hill Climbing Algorithm

Assume join is between three relations.

Step 1: Do initial processing

Step 2: Select initial feasible solution (*ES_0*)

(*ES_0 = candidate site with minimum cost)*

Step 3: Determine candidate splits of *ES_0 into*

{*ES_1, ES_2*}

Step 4: Replace *ES_0 with the split schedule*

Step 5: Recursively apply steps 3–4 on *ES_1 and*

ES_2 until no such plans can be found

Step 6: Check for redundant transmissions in the

final plan and eliminate them.