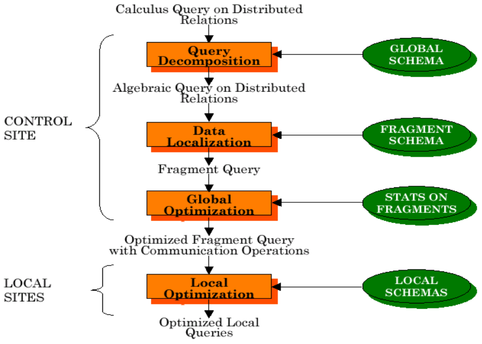
Distributed Data Processing

- Introduction
- Distributed DBMS Architecture
- Distributed DB Design
- Semantic Data Control
- **Query Processing**
- Transaction Management

6. Query Decomposition and Data Localization

- Query Decomposition
  - Transform a relational *calculus query* into an *algebra query* on global relation
- Localization of Distributed Data
  - *Localize* the query's data using data distributed information



6.1 Query Decomposition

Query Decomposition

Input : relational **calculus query** on global relations

- Normalization
  - manipulate query quantifiers and qualification
- Analysis
  - detect and reject "**incorrect**" queries
  - possible for only a subset of relational calculus
- Simplification
  - eliminate **redundant** predicates
- Restructuring
  - calculus query => algebraic query
  - more than one translation is possible
  - use transformation rules

Output: relational **algebra query** on global relations

6.1.1 Normalization

- Transform the query to a **normalized form** to facilitate further processing

- Put into normal form
  - Conjunctive normal form

    $(p_{11} \vee p_{12} \vee \ldots \vee p_{1n}) \wedge \ \ldots \ \wedge \ (p_{m1} \vee p_{m2} \vee \ldots \vee p_{mn})$
  - Disjunctive normal form

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

OR's mapped into union

AND's mapped into join or selection

Equivalent rules for logical operations

1. $p1 \wedge p2 \Leftrightarrow p2 \wedge p1$
2. $p1 \vee p2 \Leftrightarrow p2 \vee p1$
3. $p1 \wedge (p2 \wedge p3) \Leftrightarrow (p1 \wedge p2) \wedge p3$
4. $p1 \vee (p2 \vee p3) \Leftrightarrow (p1 \vee p2) \vee p3$
5. $p1 \wedge (p2 \vee p3) \Leftrightarrow (p1 \wedge p2) \vee (p1 \wedge p3)$
6. $p1 \vee (p2 \wedge p3) \Leftrightarrow (p1 \vee p2) \wedge (p1 \vee p3)$
7. $\mathrm{NOT}(p1 \wedge p2) \Leftrightarrow \mathrm{NOT}(p1) \vee \mathrm{NOT}(p2)$
8. $\mathrm{NOT}(p1 \vee p2) \Leftrightarrow \mathrm{NOT}(p1) \wedge \mathrm{NOT}(p2)$
9. $\mathrm{NOT}(\mathrm{NOT}(p)) \Leftrightarrow p$

Example of normalization

- The **conjunctive normal form** is more practical since query qualifications typically include more AND than OR predicates
- **Example:** Find the names of employees who have been working on project P1 for 12 or 24 months

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

In conjunctive normal form:

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

In disjunctive normal form:

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

6.1.2 Analysis

- Example

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

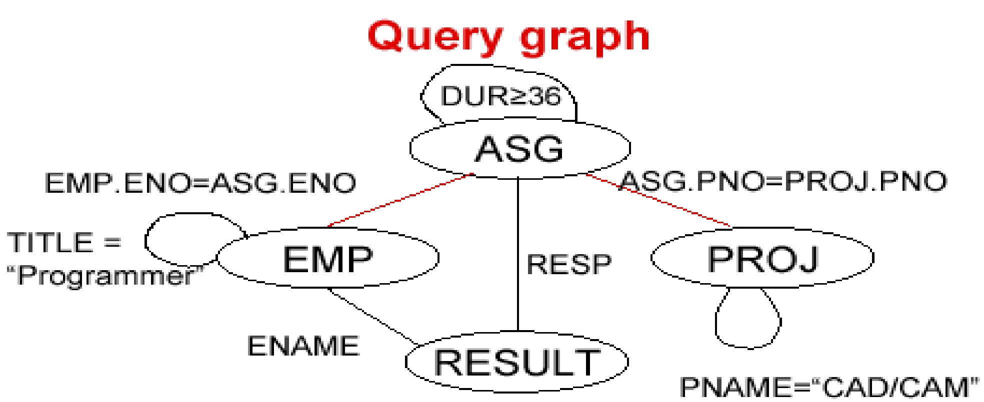
! Undefined attribute

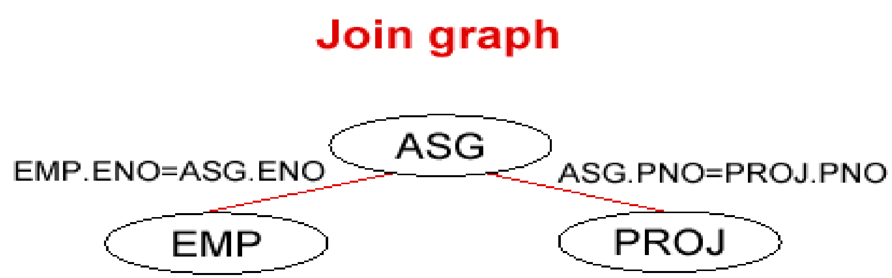
! Type mismatch

- Rejection of incorrect queries
- Type incorrect
  - 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
  - Similar to type checking
- Semantically incorrect
  - Only a subset of relational calculus queries can be tested for correctness
    - Those that do not contain disjunction and negation
  - Components do not contribute in any way to the generation of the result
  - To detect
    - query graph (connection graph)
    - join graph

Query graph

```
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"
```





Example of Analysis

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

SELECT *ENAME,RESP*

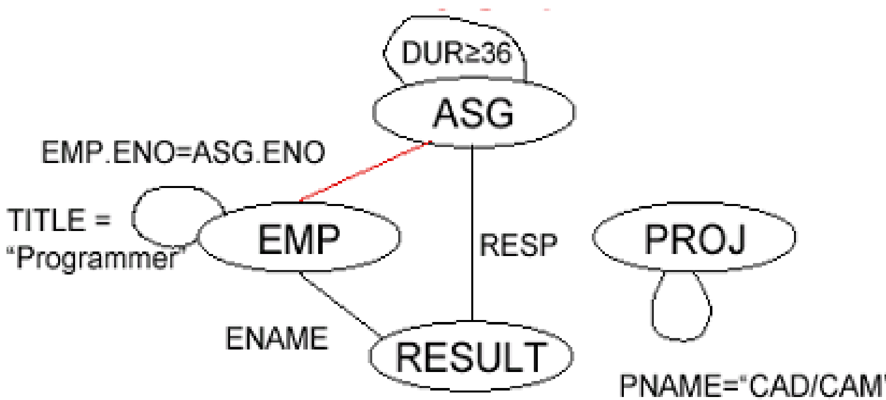
FROM *EMP, ASG, PROJ*

WHERE *EMP.ENO = ASG.ENO*

AND *PNAME = "CAD/CAM"*

AND *DUR ≥36*

AND *TITLE = "Programmer"*



Solution to the incorrect problem

- **Reject** the query
- **Assume** that there is an implicit Cartesian product between the two relations
- **Infer** (using the schema) the missing join predicate

6.1.3 Simplification

- Elimination of Redundancy
  - A user query typically expressed on a view may be enriched with several predicates to achieve view-relation correspondence, and ensure semantic integrity and security – that may then contain redundant predicates
- Idempotency rules

1. $p \wedge p \Leftrightarrow p$
2. $p \vee p \Leftrightarrow p$
3. $p \wedge true \Leftrightarrow p$
4. $P \vee false \Leftrightarrow p$
5. $p \wedge false \Leftrightarrow false$
6. $P \vee true \Leftrightarrow true$
7. $p \wedge NOT(p) \Leftrightarrow false$
8. $p \vee NOT(p) \Leftrightarrow true$
9. $p1 \wedge (p1 \vee p2) \Leftrightarrow p1$
10. $p1 \vee (p1 \wedge p2) \Leftrightarrow p1$

Example1: Semantic Data Control – View Management

Query expressed on view

SELECT **ENAME, PNO, RESP**
FROM ***SYSAN*, ASG**
WHERE ***SYSN*.ENO = ASG.ENO**

Query expressed on base relation

SELECT **ENAME,PNO,RESP**
FROM ***EMP*, ASG**
WHERE ***EMP*.ENO = ASG.ENO**
AND TITLE = "Syst. Anal."

Definition of view

CREATE VIEW ***SYSAN*(ENO,ENAME)**
AS **SELECT ENO,ENAME**
FROM *EMP*
WHERE *TITLE="Syst. Anal."*

Example2: Semantic Data Control – Semantic Integrity Control

- **Increasing the budget of CAD/CAM project by 10%**

```
UPDATE  PROJ
SET  BUDGET = BUDGET*1.1
WHERE  PNAME =“CAD/CAM”

UPDATE  PROJ
SET B UDGET = BUDGET*1.1
WHERE  PNAME =“CAD/CAM”
     AND NEW.BUDGET ≥500000
     AND NEW.BUDGET ≤1000000
```

Example of simplification

Idempotency rules

$p \wedge p \Leftrightarrow p$

$p \vee p \Leftrightarrow p$

$p \wedge true \Leftrightarrow p$

$P \vee false \Leftrightarrow p$

$p \wedge false \Leftrightarrow false$

$P \vee true \Leftrightarrow true$

$p \wedge NOT(p) \Leftrightarrow false$

$p \vee NOT(p) \Leftrightarrow true$

$p1 \wedge (p1 \vee p2) \Leftrightarrow p1$

$p1 \vee (p1 \wedge p2) \Leftrightarrow p1$

SELECT *TITLE*
FROM *EMP*
WHERE *EMP.ENAME = "J. Doe"* P1
OR *(NOT(EMP.TITLE = "Programmer")*
AND *(EMP.TITLE = "Programmer"* P2
OR *EMP.TITLE = "Elect. Eng.")* P3
AND *NOT(EMP.TITLE = "Elect. Eng."))*

➔

SELECT *TITLE*
FROM *EMP*
WHERE *EMP.ENAME = "J. Doe"*

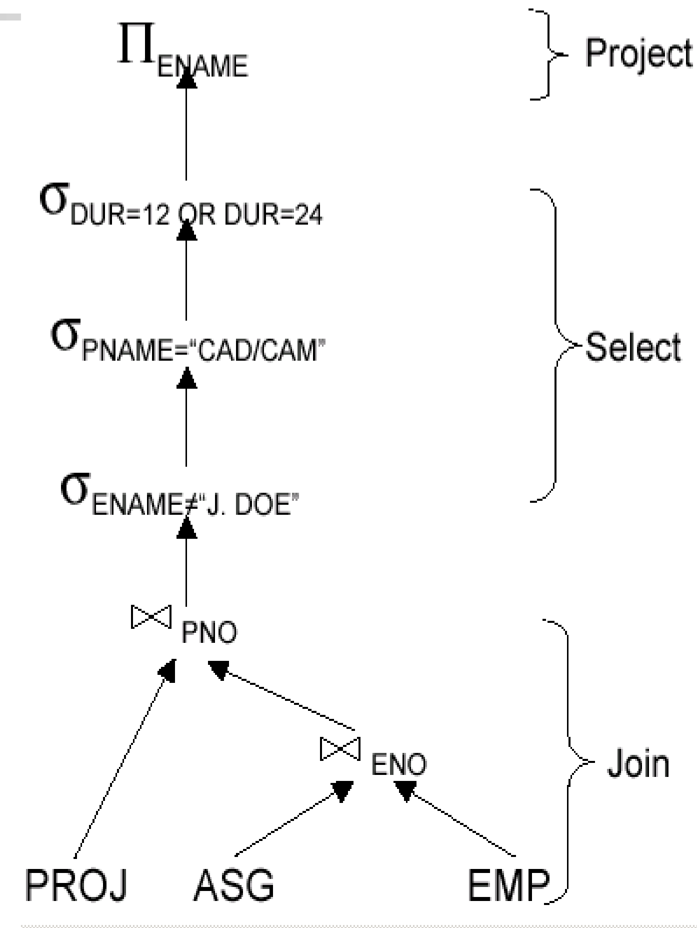
6.1.4 Restructuring

- Rewrite the query based on relational calculus to that based on relational algebra
  - Straightforward transformation of the query from relational calculus into relational algebra
  - Restructuring of the relational algebra query to improve performance by transformation rules
- **operator tree**

Example of operation tree

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)



Transformation rules

- Commutativity of binary operations
  - $R \times S \Leftrightarrow S \times R$
  - $R \infty S \Leftrightarrow S \infty R$
  - $R \cup S \Leftrightarrow S \cup R$
- Associativity of binary operations
  - $(R \times S) \times T \Leftrightarrow R \times (S \times T)$
  - $(R \infty S) \infty T \Leftrightarrow R \infty (S \infty T)$
- Idempotence of unary operations
  - $\prod_{A'}(\prod_{A''}(R)) \Leftrightarrow \prod_{A'}(R)$
  - $\delta_{p1(A1)}(\delta_{p2(A2)}(R)) = \delta_{p1(A1) \wedge p2(A2)}(R)$

  where R[A] and A' belongs to A, A" belongs to A and A' belongs to A"
- Commuting selection with projection
  - $\prod_{A1\ldots An}(\delta_{p(Ap)}(R)) \Leftrightarrow \prod_{A1\ldots An}(\delta_{p(Ap)}(\prod_{A1\ldots An\ Ap}(R)))$

Transformation rules

- Commuting selection with binary operations
  - $\delta_{p(A)}(R \times S) \Leftrightarrow (\delta_{p(A)}(R)) \times S$
  - $\delta_{p(Ai)}(R \infty_{(A\,j\,,B\,k\,)} S) \Leftrightarrow (\delta_{p(Ai)}(R)) \infty_{(A\,j\,,B\,k\,)} S$
  - $\delta_{p(Ai)}(R \cup T) \Leftrightarrow (\delta_{p(Ai)}(R)) \cup (\delta_{p(Ai)}(T))$

  where *Ai* belongs to *R* and *T*
- Commuting projection with binary operations
  - $\Pi_C(R \times S) \Leftrightarrow \Pi_{A'}(R) \times \Pi_{B'}(S)$
  - $\Pi_C(R \infty_{(A\,j\,,B\,k\,)} S) \Leftrightarrow \Pi_{A'}(R) \infty_{(A\,j\,,B\,k\,)} \Pi_{B'}(S)$
  - $\Pi_C(R \cup S) \Leftrightarrow \Pi_C(R) \cup \Pi_C(S)$

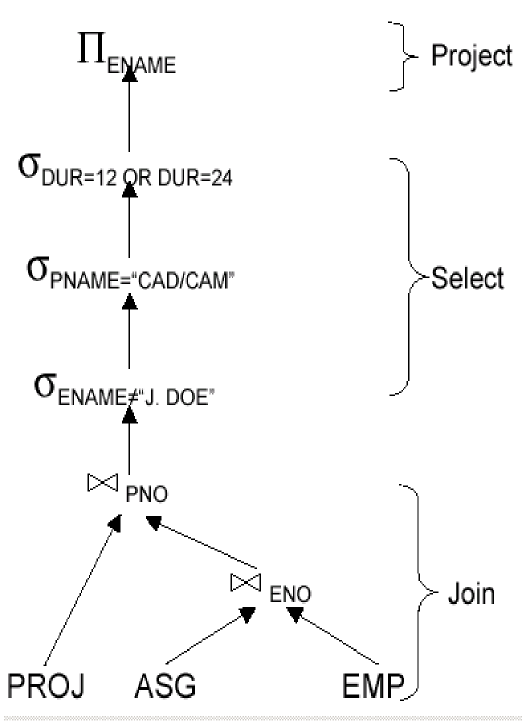
  where R[A] and S[B]; *C* = A' ∪ B' where *A' belongs to* A, *B' belongs to B*

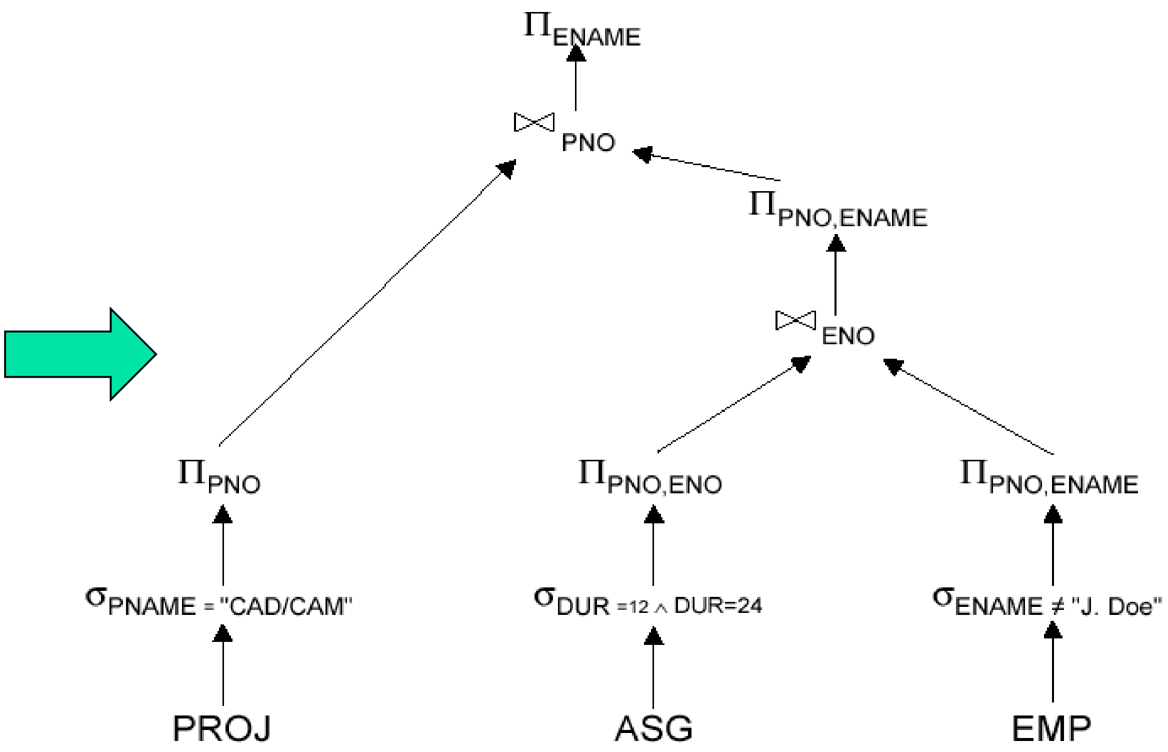
Restructure

- To restructure the tree should be in a systematic way so that the “bad” operator trees are eliminated.
  - Allow the separation of the unary operations, simplifying the query expression
  - Unary operations on the same relations may be grouped so that access to a relation for performing unary operations can be done only once
  - Unary operations can be commuted with binary operations so that some operations may be done first
  - The binary operations can be ordered

Note: Generally, after the layer, the query is still far from providing an optimal execution, since information about data distribution and local fragments is not used at this layer

Example of restructuring





6.2 Localization of Distributed Data

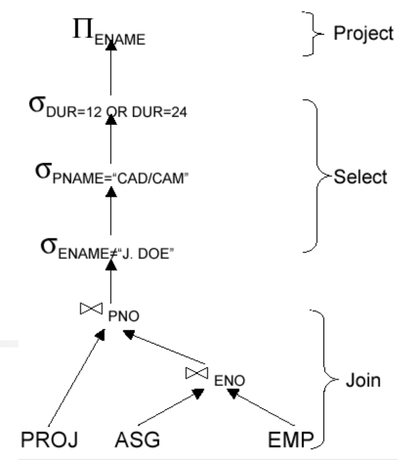
Localization of Distributed Data

Input: Algebraic query on **global relations**

- Localize the query's data using data distributed information
  - Determine which fragments are involved
  - Transform the distributed (globe) query into a fragment query
- Step1: Localization program
  - substitute for each global query its materialization (reconstruction) program
- Step2: Optimize
  - simplification and restructuring

Output: Algebra query on **physical fragments**

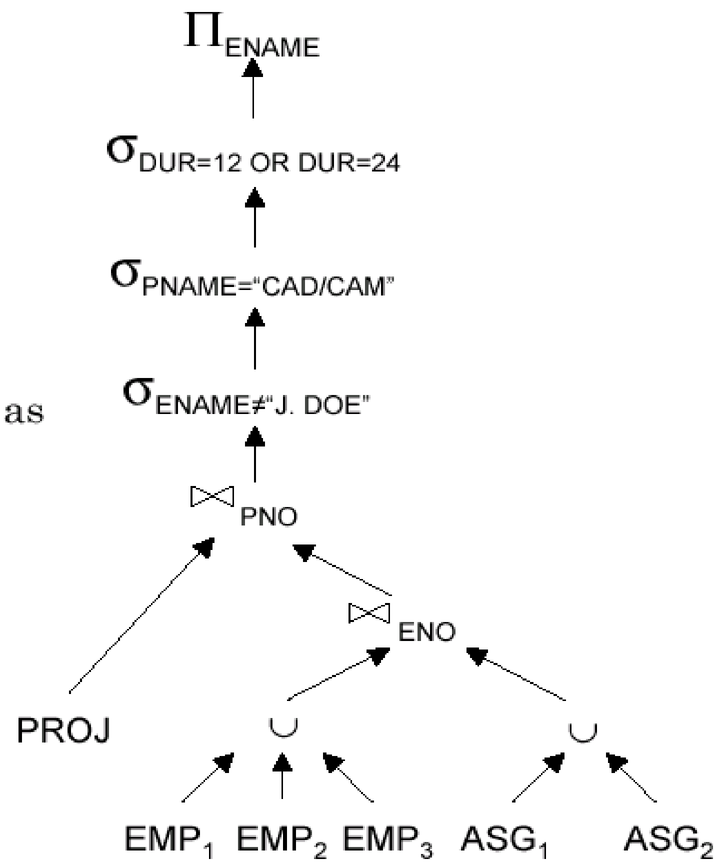
Example



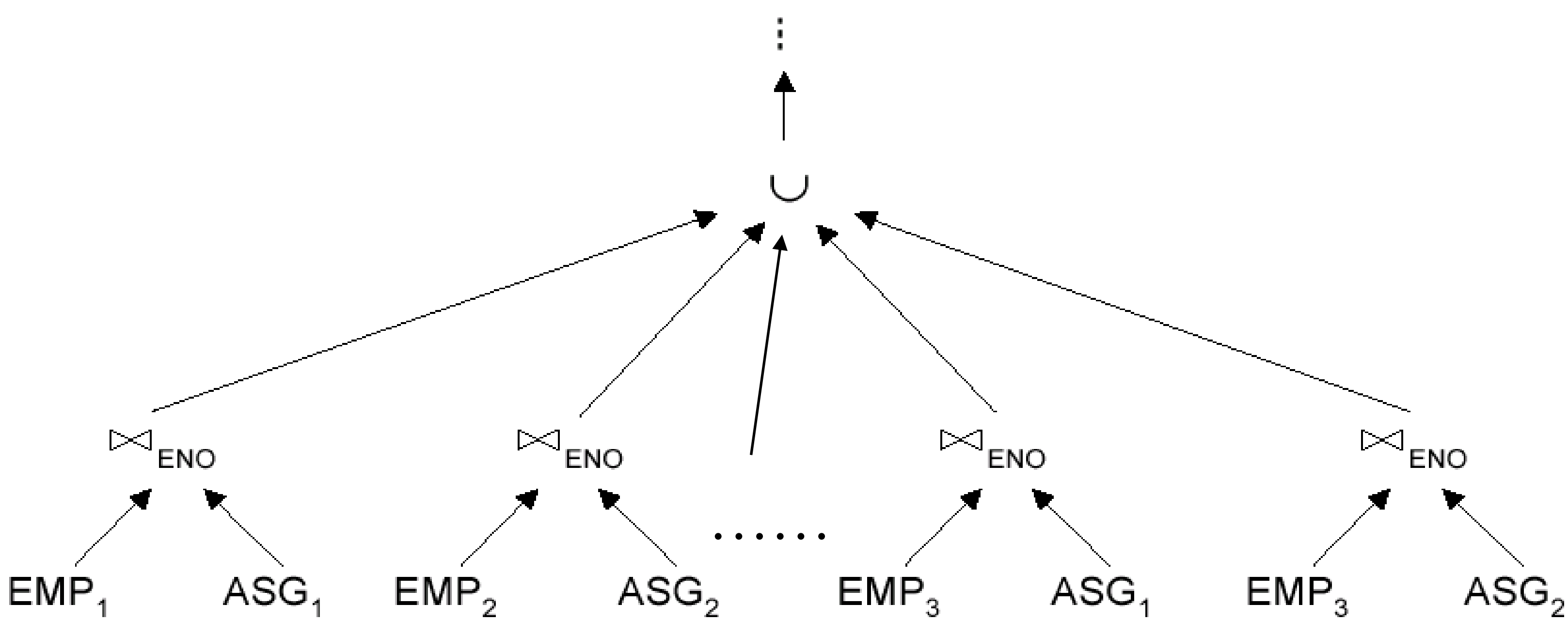
Assume

- 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:
  - $ASG_1 = \sigma_{ENO \leq \text{"E3"}}(ASG)$
  - $ASG_2 = \sigma_{ENO > \text{"E3"}}(ASG)$

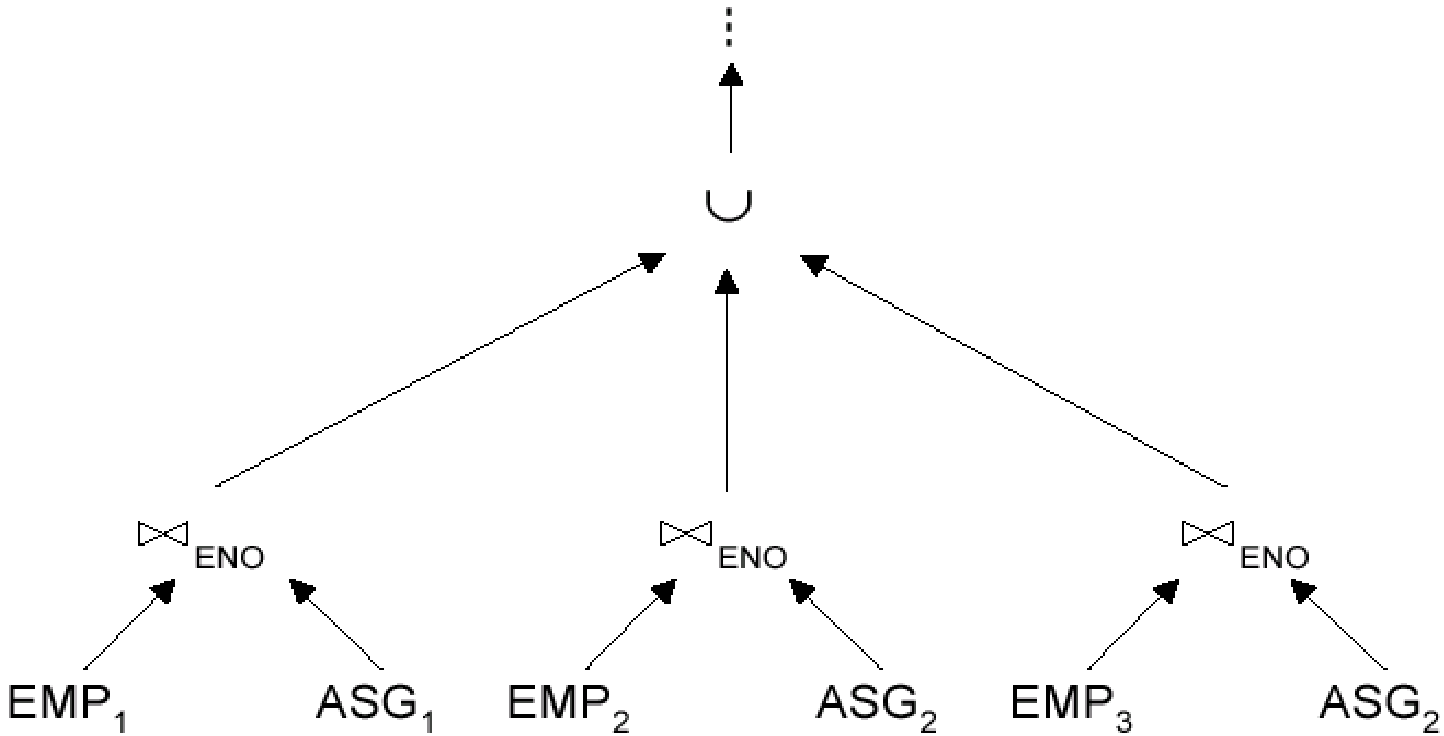
Replace EMP by ($EMP_1 \cup EMP_2 \cup EMP_3$) and ASG by ($ASG_1 \cup ASG_2$) in any query



Provides Parallelism



Eliminates Unnecessary Work



6.2.1 Reduction for PHF

- The **localization program** for an PHF relation is the **union** of the fragments
- Reduction for PHF
  - After restructuring the subtrees, Determining those that will produce **empty relations**, and removing them

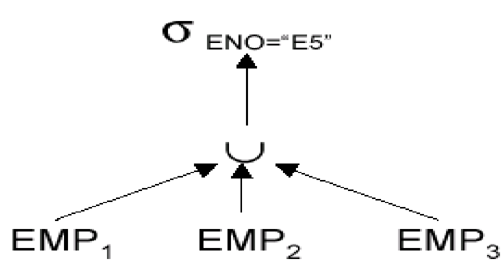
Reduction with selection

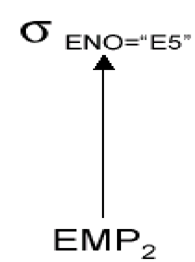
- Selections on fragments that have a qualification contradicting the qualification of the fragmentation rule generate empty relations
  - Relation R and $F_R=\{R_1, R_2, \ldots, R_w\}$ where $R_j=\sigma_{p_j}(R)$

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

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

- $EMP_1=\sigma_{ENO\leq\text{“E3”}}(EMP)$
- $EMP_2=\sigma_{\text{“E3”}<ENO\leq\text{“E6”}}(EMP)$
- $EMP_3=\sigma_{ENO\geq\text{“E6”}}(EMP)$





Reduction with join

- Possible if fragmentation is done on join attribute
  - The simplification consists of distributing join over unions and eliminating useless joins

➠ 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 $S_j = \sigma_{p_j}(S)$

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

Example of Reduction with join

- Assume EMP is fragmented as before and

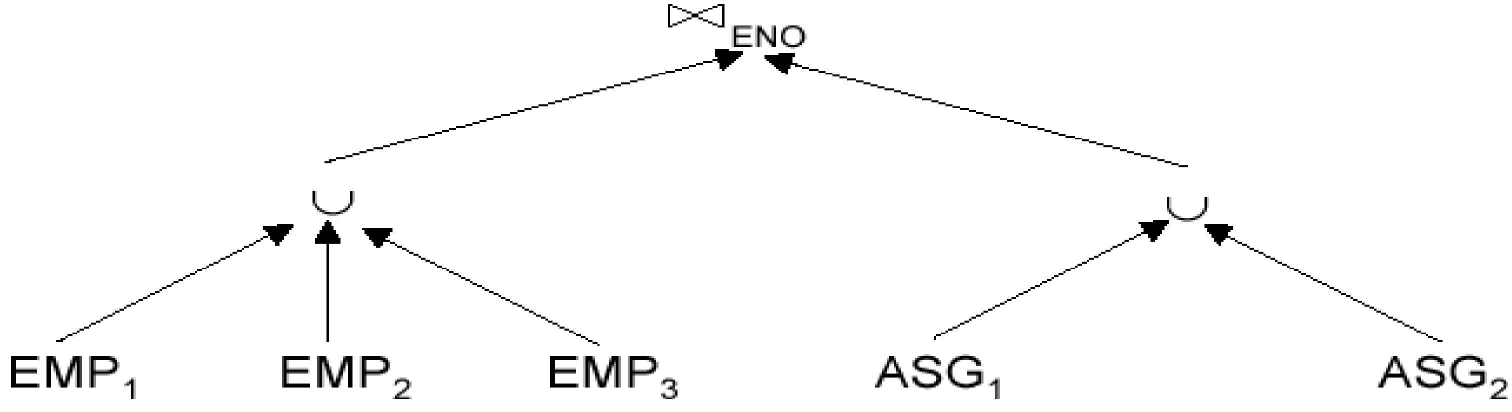
  ASG_1: $\sigma_{ENO \leq "E3"}(ASG)$

  ASG_2: $\sigma_{ENO > "E3"}(ASG)$

- Consider the query

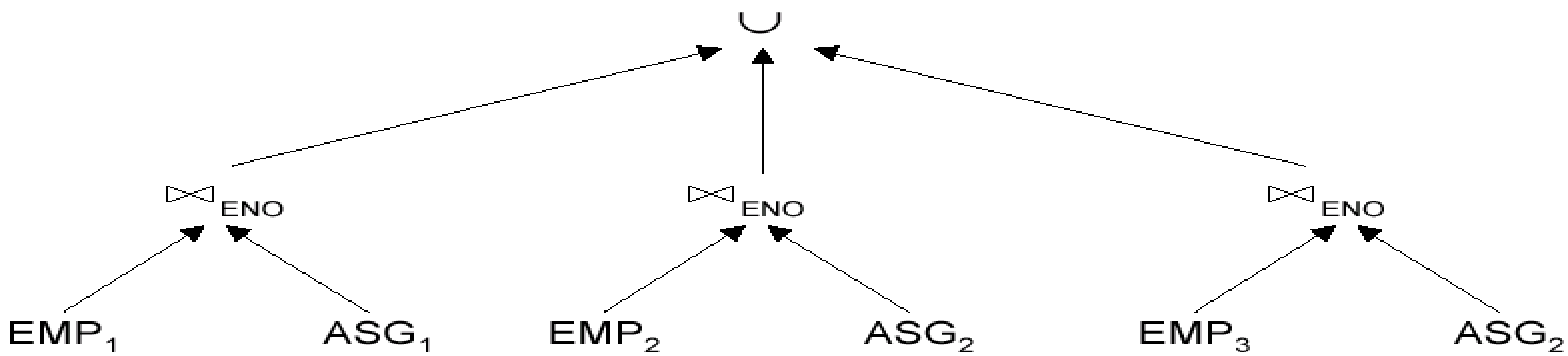
```
SELECT *
FROM   EMP, ASG
WHERE  EMP.ENO=ASG.ENO
```

- $EMP_1 = \sigma_{ENO \leq "E3"}(EMP)$
- $EMP_2 = \sigma_{"E3" < ENO \leq "E6"}(EMP)$
- $EMP_3 = \sigma_{ENO \geq "E6"}(EMP)$



Example of Reduction with join

- Distribute join over unions
- Apply the reduction rule



- One advantage of the reduced query is that the partial joins can be done in parallel, and thus improve response time
- Reduced query is better when the number of partial joins is small

6.2.2 Reduction for VF

- The **localized program** for a VF relation consists of the **join** of the fragments on the common attribute
- Reduction for VF
  - Determining the **useless** intermediate relations and removing the subtrees that produce them
  - Projects on a vertical fragment that has no attributes in common with the projection attributes (except the key of the relation) produce useless, though not empty relations

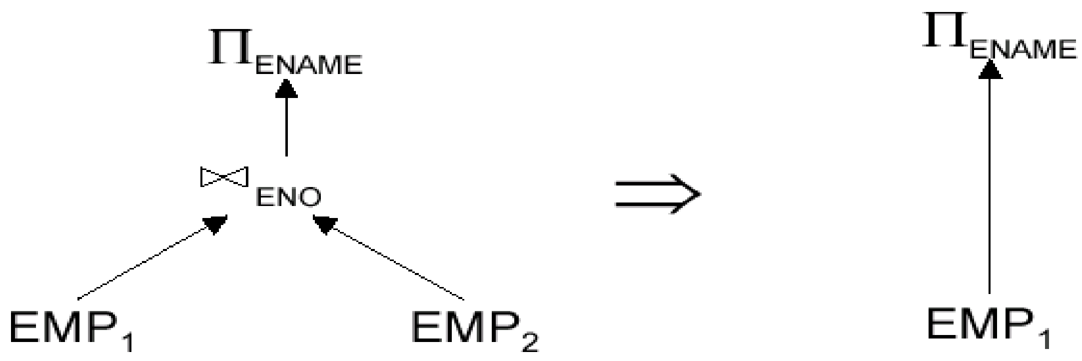
Example of reduction for VF

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
```



6.2.3 Reduction for DHF

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

  ASG1 : ASG $\propto_{ENO}$ EMP1

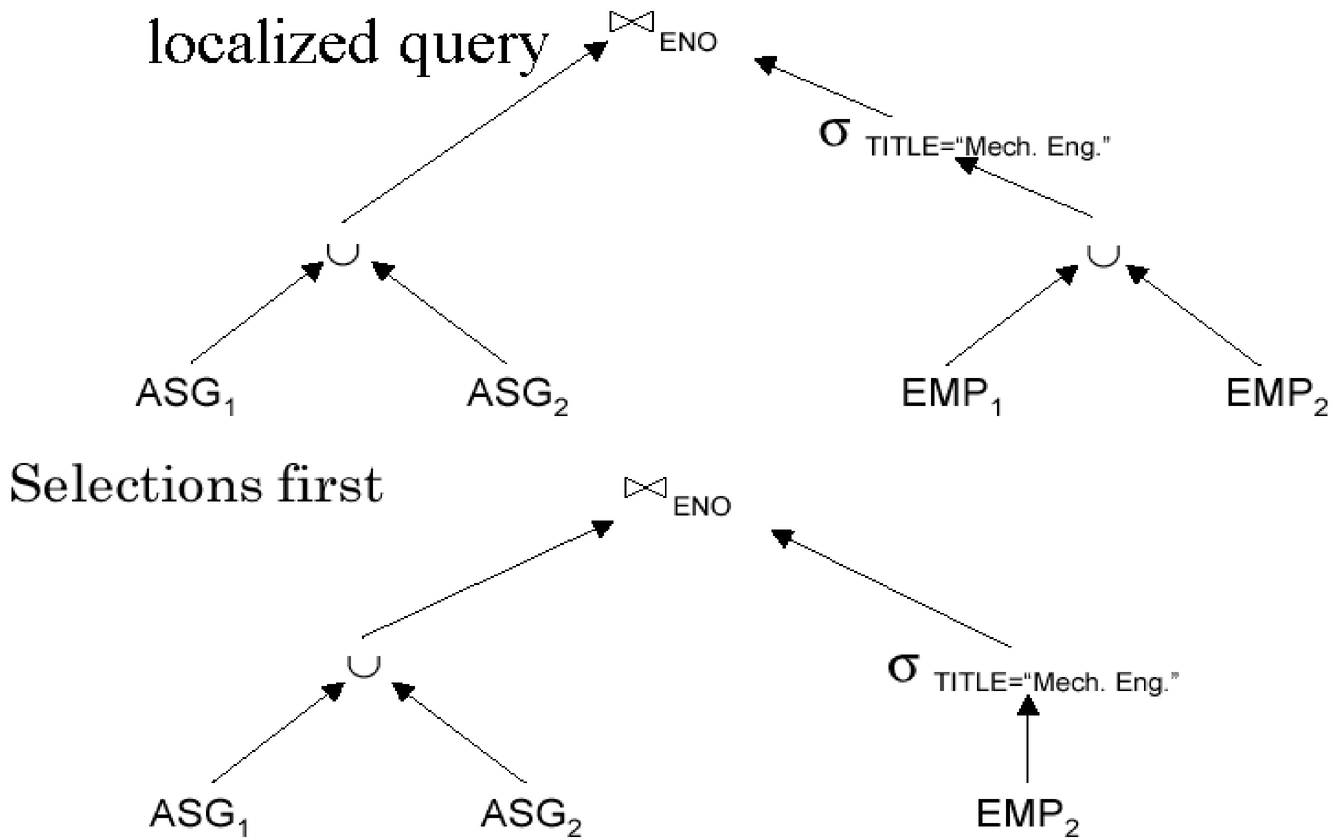
  ASG2 : ASG $\propto_{ENO}$ EMP2

  EMP1 : $\delta_{TITLE=\text{“Programmer”}}$ (EMP)

  EMP2 : $\delta_{TITLE\neq \text{“Programmer”}}$ (EMP)
- Query

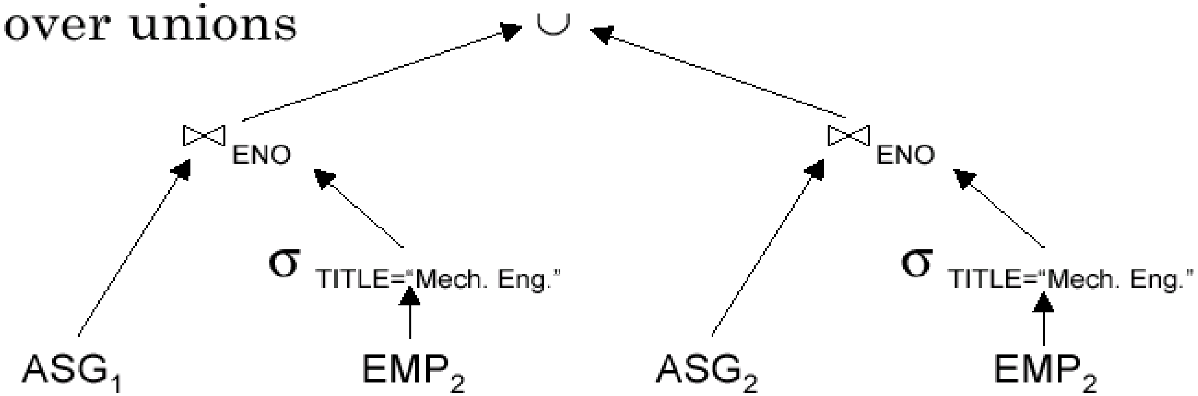
  SELECT *
  FROM EMP, ASG
  WHERE ASG.ENO = EMP.ENO
  AND EMP.TITLE = “Mech. Eng.”

Example of reduction for DHF

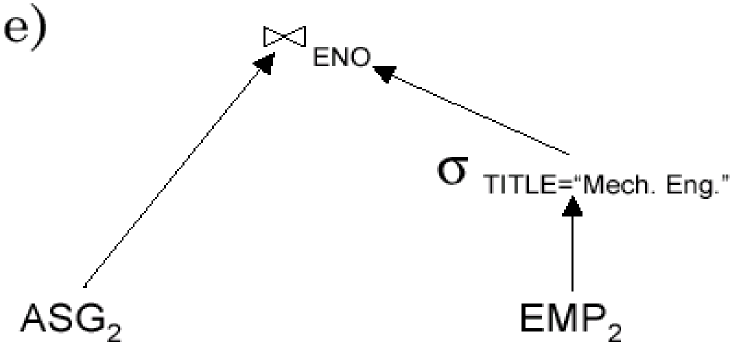


Example of reduction for DHF

Joins over unions



Elimination of the empty intermediate relations (left sub-tree)



The reduced query is always preferable to the localized query because the number of partial joins usually equals the number of fragments

6.2.4 Reduction for HF

- 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***.

Example of reduction for HF

Consider the following hybrid fragmentation:

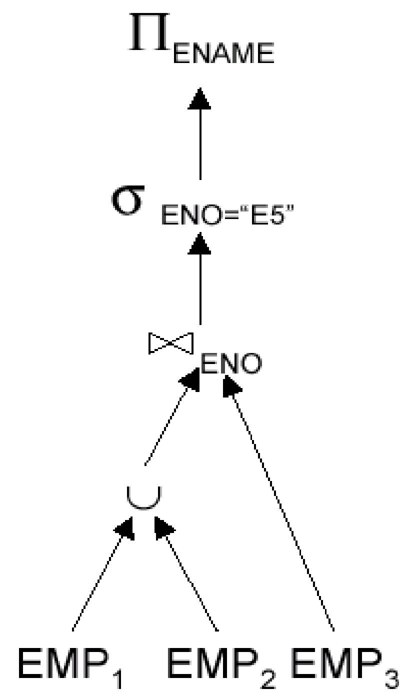
$EMP_1 = \sigma_{ENO \leq "E4"} (\Pi_{ENO, ENAME} (EMP))$

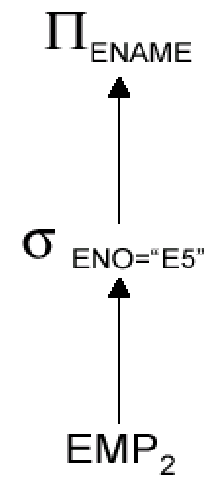
$EMP_2 = \sigma_{ENO > "E4"} (\Pi_{ENO, ENAME} (EMP))$

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

and the query

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





Note: Generally, after the layer, the query is still far from providing an optimal execution

Conclusion

Conclusion

- Naive way
  - Query decomposition
  - Data localization
- Optimization in the above layers
  - Avoid the worse executions
- Global and local Optimization layers

References

- M. Jarke and J. Koch. “Query Optimization in Database Systems,” *Computing Surveys*, June 1984, 16(2): 227-269.
- J. D. Ullman, Principles of Database Systems (2nd edition), Rockville, Md.: Computer Science Press, 1982.
- S. Ceri and Pelagatti, Correctness of Query Execution Strategies in Distributed Databases, ACM Trans. Database System (Dec. 1983), 8(4): 577-607