Lecture 3: Query processing and optimization

Objectives:

- Reduce processing time
- Reduce buffer memory
- Reduce communication cost between sites
- Low resource usage

Hoa Dinh Nguyen, PhD.
Department of Information Technology

Introduction

- ***Functions of query processing***:
  - Transform a complicated query into a much simpler query.
  - This transformation must ensure correctness and efficiency
  - Each transformation method leads to different resource usages → lowest resource usage.

Hoa Dinh Nguyen, PhD.
Department of Information Technology

Introduction

Simple transformation methods

Relational algebra:

- Simplify queries based on equivalent relational algebra expressions such that the querying time is minimized.
- This method disregards database structure and size.

Cost estimation:

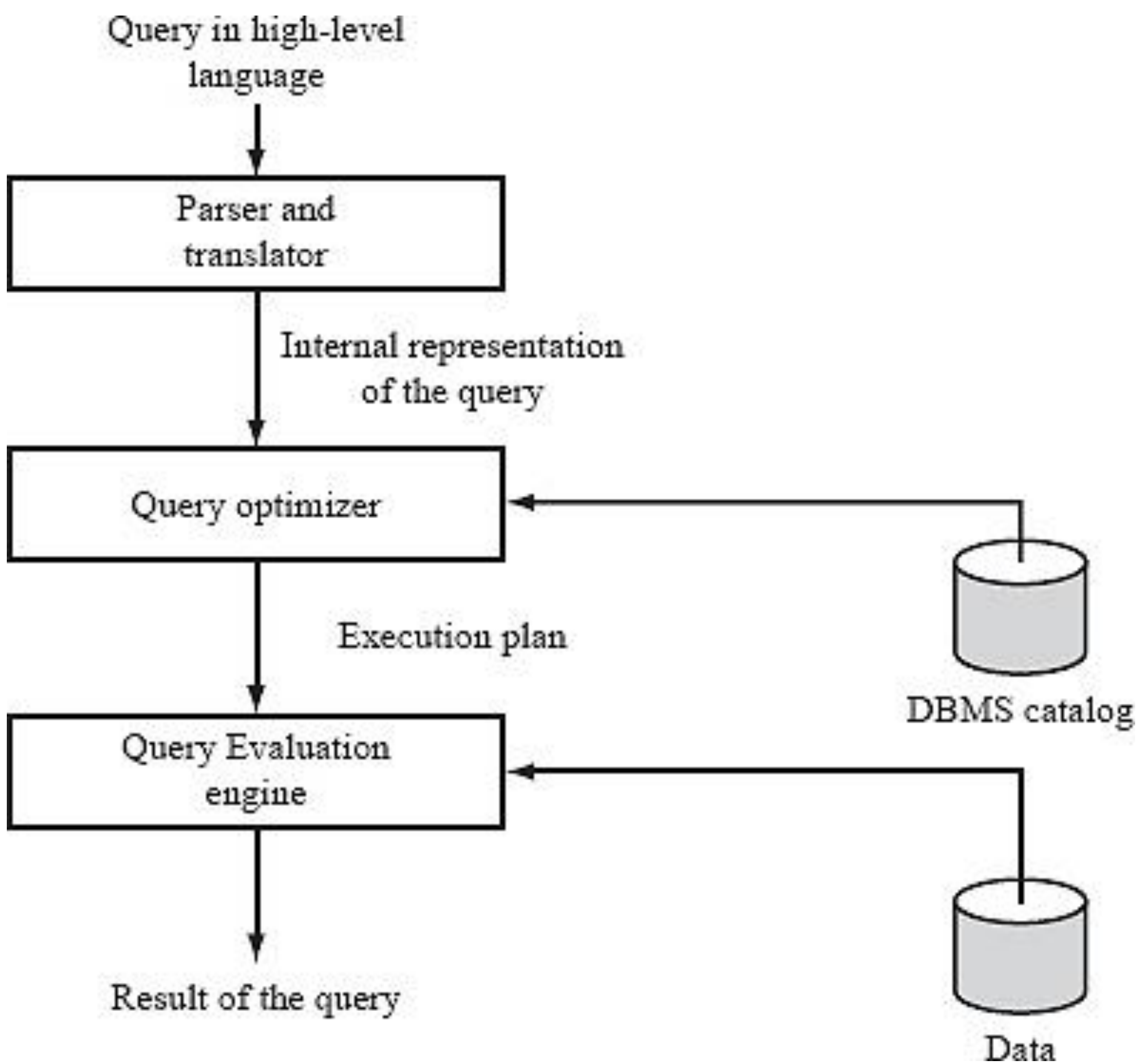
- Specify data size and processing time of each element in the query.
- This method takes into account the data size and real processing time of the query.

Hoa Dinh Nguyen, PhD.
Department of Information Technology

Querying Process

- Procedure of query execution
- Query pre-processing
- Query transformation
- Query optimization

Querying Process



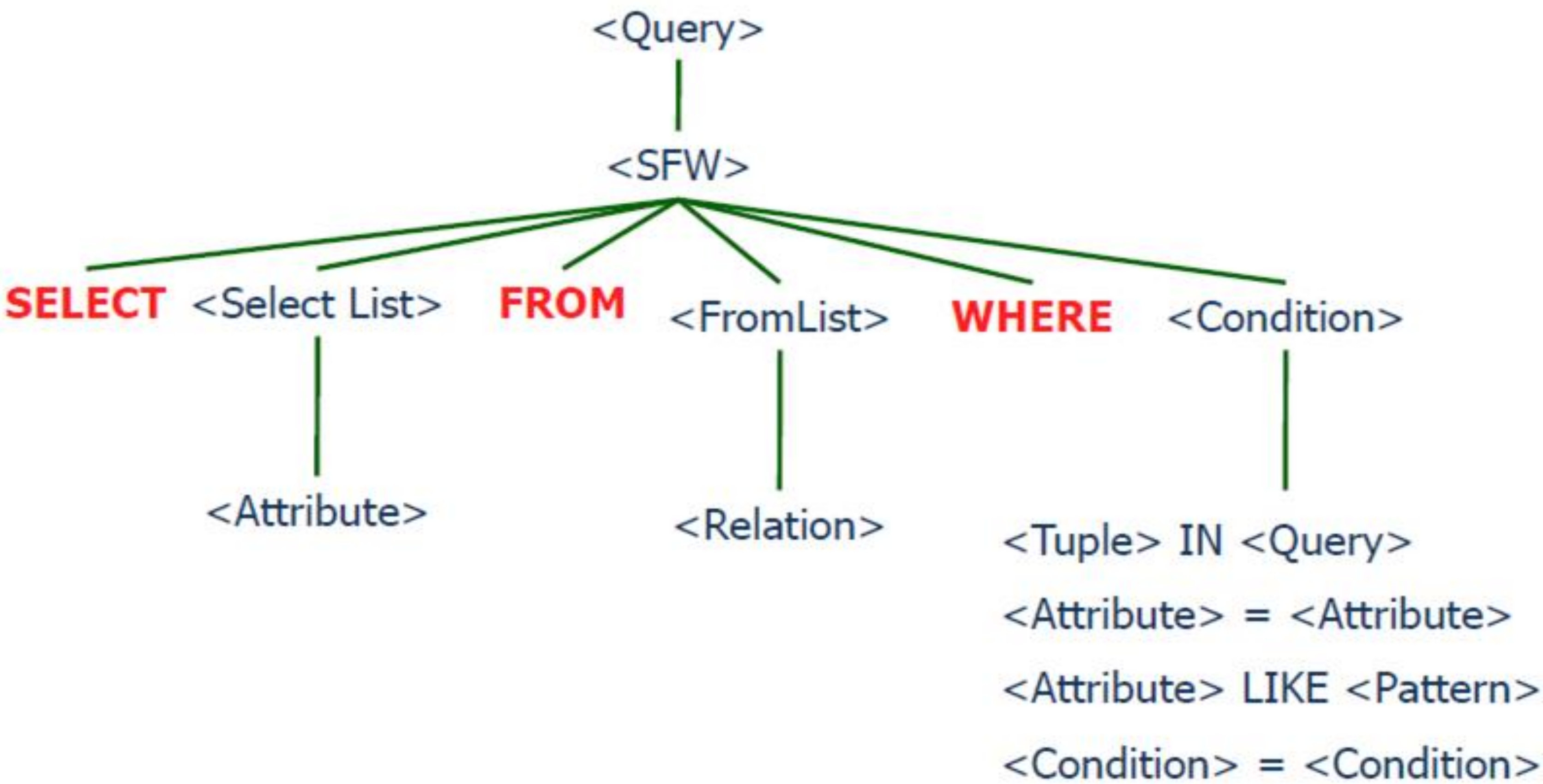
Querying Process

- **Query preprocessor:**
  - Scanning: key words, attribute names, relations,…
  - Parsing: syntax checking, parse tree representation
  - Validating: semantic checking (relations, attributes, data types)

Querying Process

- **Query optimizer:**
  - Select suitable strategy for query processing
- **Query code generator:**
  - Generate codes to implement the plan
- **Runtime database processor:**
  - Compile codes to provide query results

Querying Process



Querying Process

Transform in to relational algebra expressions

- Query blocks: SELECT-FROM-WHERE-GROUP BY-HAVING
- Integrated queries: separate into query blocks

Querying Process

Transform in to relational algebra expressions

```
SELECT Name, Salary
FROM Staff
WHERE Salary > (SELECT AVG(Salary)
                FROM Staff
                WHERE Gender = “Male”)
```

Query Processing in Centralized DBS

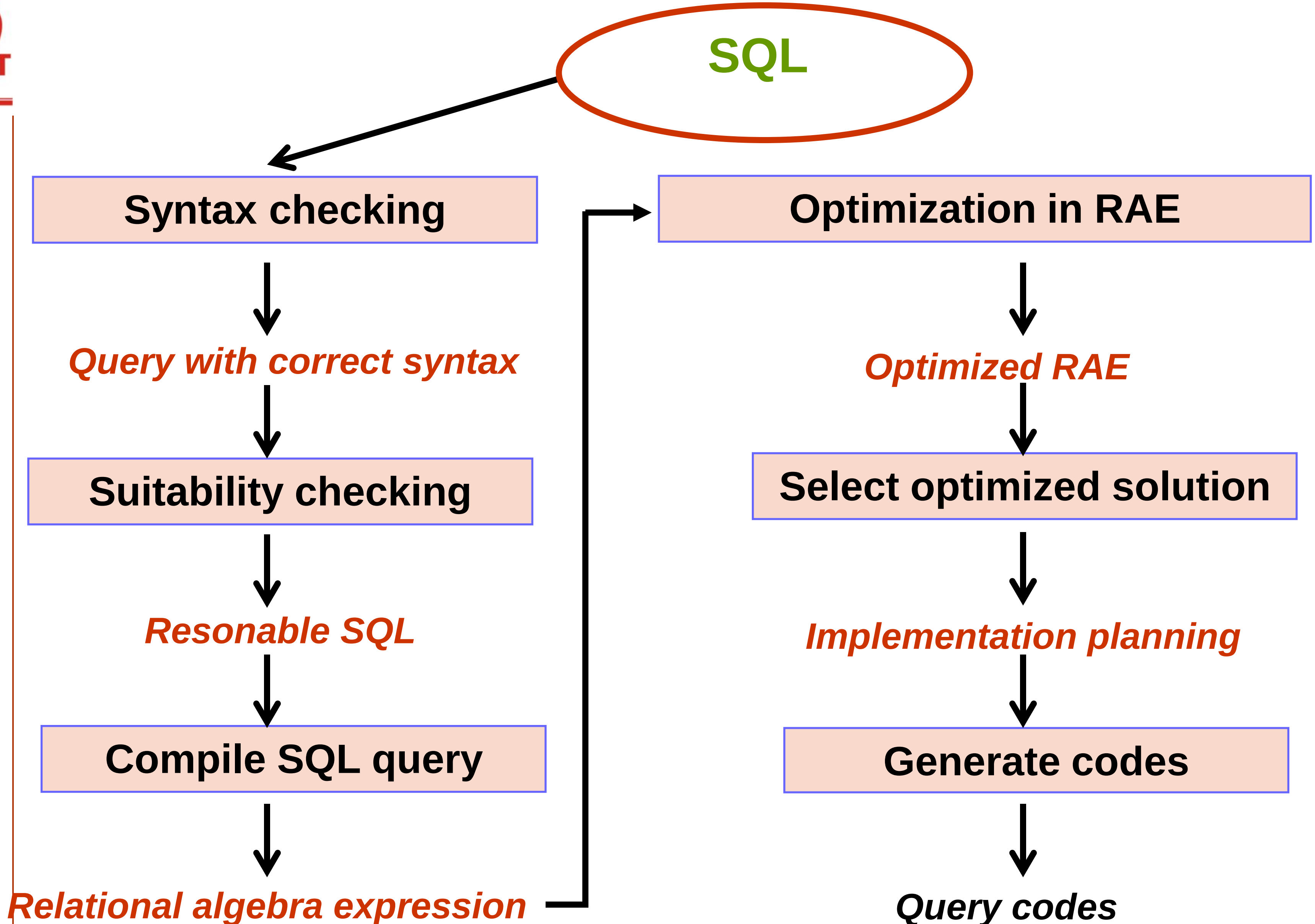
Comparison between centralized and distributed query processing:

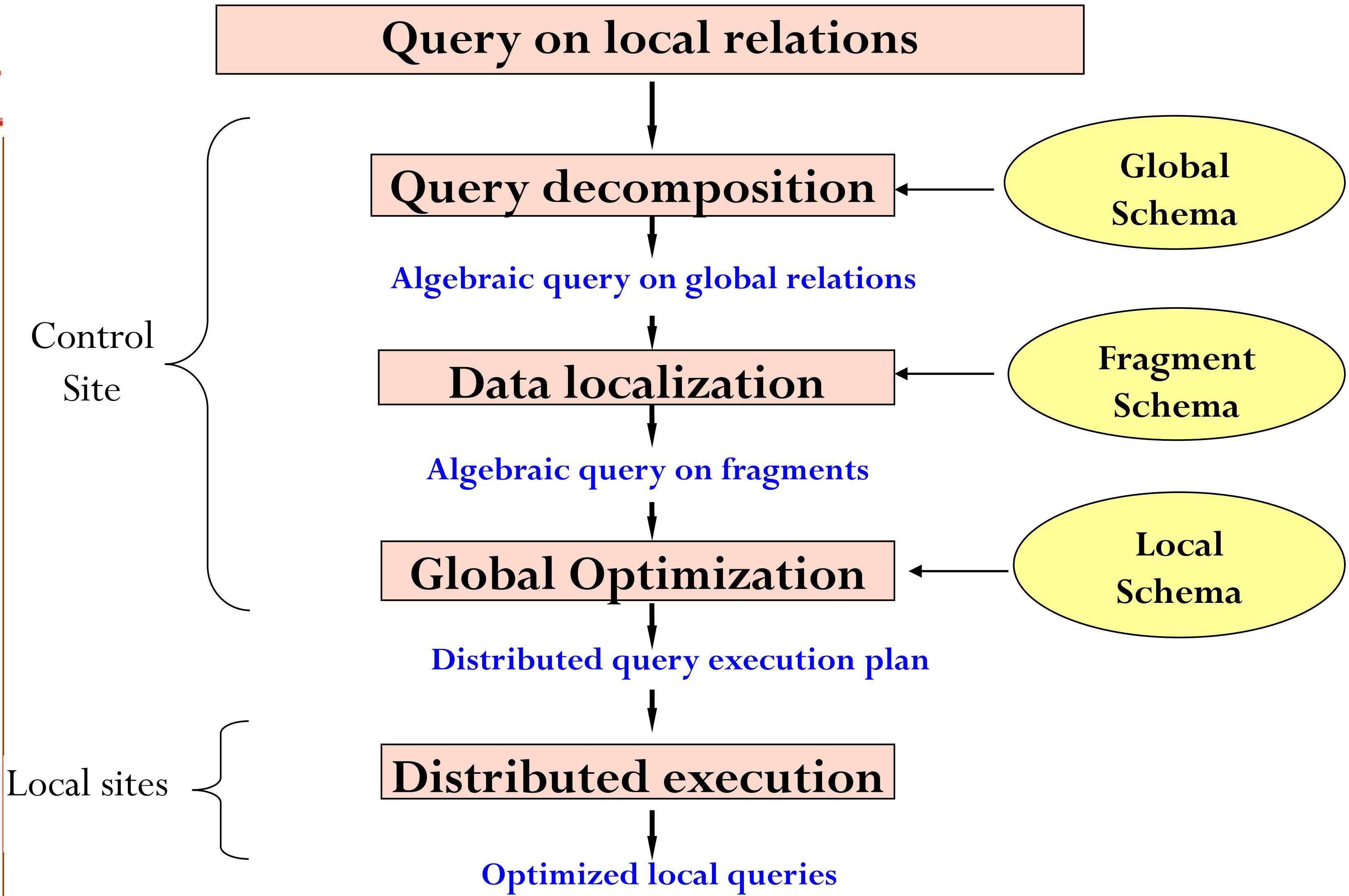
Centralized:

- Select the best relational algebra expression equivalent to the query
- Query processing strategies are described using relational algebra extensions

Distributed:

- Inherited from centralized environment
- More issues to concern:
  - Math expressions of data transmission between sites
  - Choose the best site to process data
  - Data transmission methods


SQL
Syntax checking
Query with correct syntax
Suitability checking
Resonable SQL
Compile SQL query
Relational algebra expression
Optimization in RAE
Optimized RAE
Select optimized solution
Implementation planning
Generate codes
Query codes



Generic layering scheme for distributed query processing

4

Query Processing in Centralized DBS

Optimization strategies in centralized databases:

- Distributed queries must be composed and processed in a centralized manner
- All distributed query optimization techniques are extensions from centralized query processing approaches
- Centralized query optimization is often simple

Query Processing in Centralized DBS

INGRES algorithm: recursively breaks up a query expressed in SQL into smaller pieces which are executed along the way

- The query is first decomposed into a sequence of queries having a unique relation in common
- Each mono-relation query is processed by selecting, based on the predicate, the best access method to that relation

E.g: For example, if the predicate is of the form $A = value$, an index available on attribute A would be used if it exists. However, if the predicate is of the form $A \neq value$, an index on A would not help, and sequential scan should be used.

Query Processing in Centralized DBS

INGRES algorithm:

- Executes first the unary (monorelation) operations and tries to minimize the sizes of intermediate results in ordering binary (multirelation) operations
- Denote by $q_{i-1} \rightarrow q_i$ a query q decomposed into two subqueries, q_{i-1} and q_i, where q_{i-1} is executed first and its result is consumed by q_i
- Decomposes q into n subqueries $q_1 \rightarrow q_2 \ldots \rightarrow q_n$ using *detachment* and *substitution*.

Query Processing in Centralized DBS

Detachment: breaks a query q into q'→q'', based on a common relation that is the result of q'

q: **SELECT** $R_2.A_2, R_3.A_3, \ldots, R_n.A_n$
FROM $R_1, R_2, \ldots, R_n$
WHERE $P_1(R_1.A'_1)$ **AND** $P_2(R_1.A_1, R_2.A_2, \ldots, R_n.A_n)$;

q': **SELECT** R_1A_1 **INTO** R'_1
FROM R_1
WHERE $P_1(R_1.A_1)$;

q'': **SELECT** $R_2A_2, \ldots, R_nA_n$
FROM $R'_1, R_2, \ldots, R_n$
WHERE $P_2(R_1.A_1, R_2.A_2, \ldots, R_n.A_n)$;

EMP (E)

ENO	ENAME	TITLE
A1	Nam	Phân tích HT
A2	Trung	Lập trình viên
A3	Đông	Phân tích HT
A4	Bắc	Phân tích HT
A5	Tây	Lập trình viên
A6	Hùng	Kỹ sư điện
A7	Dũng	Phân tích HT
A8	Chiến	Thiết kế DL

ASG (G)

ENO	PNO	RESPONSIBILITY	DUR
A1	D1	Quản lý	12
A2	D1	Phân tích	34
A2	D2	Phân tích	6
A3	D3	Kỹ thuật	12
A3	D4	Lập trình	10
A4	D2	Quản lý	6
A5	D2	Quản lý	20
A6	D4	Kỹ thuật	36
A7	D3	Quản lý	48
A8	D3	Lập trình	15

PRJ (J)

PNO	PNAME	BUDGET
D1	CSDL	20000
D2	CÀI ĐẶT	12000
D3	BẢO TRÌ	28000
D4	PHÁT TRIỂN	25000

PAY(S)

TITLE	SALARY
Kỹ sư điện	1000
Phân tích HT	2500
Lập trình viên	3000
Thiết kế DL	4000

Example

q_1="*Find names of staffs working on project CSDL*"

```
q1:    SELECT  E.ENAME
       FROM    E, G, J
       WHERE   E.ENO = G.ENO AND G.PNO = J.PNO
                     AND PNAME = "CSDL";
```

q_1 is detached into $q_{11} \rightarrow q'$, where TGIAN1 is an intermediate relation.

```
q11:   SELECT  J.PNO INTO TEMP1
       FROM    J
       WHERE   PNAME = "CSDL";
q':    SELECT  E.ENAME
       FROM    E, G, TEMP1
       WHERE   E.ENO = G.ENO
               AND G.PNO = TEMP1.PNO;
```

Example

The successive detachments of q' may generate

```
q12:  SELECT G.ENO INTO TEMP2
      FROM   G, TEMP1
      WHERE  G.PNO =TEMP1.PNO;
q13:  SELECT E.ENAME
      FROM   E, TEMP2
      WHERE  E.ENO = TEMP2.ENO;
```

q_1 has been decomposed into $q_{11} \rightarrow q_{12} \rightarrow q_{13}$

Query q_{11} is mono-relation and can be executed. However, q_{12} and q_{13} are not mono-relation and cannot be reduced by detachment.

Query Processing in Centralized DBS

Tuple substitution: Given an *n-relation* query q, the tuples of one relation are substituted by their values, thereby producing a set of *(n-1)-relation* queries.

- One relation in q is chosen for tuple substitution. Let R_1 be that relation.
- For each tuple $t_{1i} \in R_1$, the attributes referred to by in q are replaced by their actual values in t_{1i}, thereby generating a query q' with n-1 relations.

$q(R_1,R_2,\ldots,R_n)$ is replaced by $\{q'(t_{1i},R_2,R_3,\ldots,R_n), t_{1i} \in R_1\}$

Query Processing in Centralized DBS

Example: Let's consider the query q_{13}

q_{13}: **SELECT** E.ENAME
FROM E, TEMP2
WHERE E.ENO = TEMP2.ENO ;

The relation TEMP2 is over a single attribute (ENO). Assume that it contains only two tuples: <E1> and <E2>. The substitution of TEMP2 generates two one-relation subqueries:

q_{131}: **SELECT** E.ENAME
FROM E
WHERE E.ENO = "E1";

q_{132}: **SELECT** E.ENAME
FROM E
WHERE E.ENO = "E2";

These queries may then be executed

Hoa Dinh Nguyen, PhD.
Department of Information Technology

INGRES- QOA Algorithm

Input: MRQ: multirelation query with *n* relations

Output: Result of execution

Begin

Output $\leftarrow \phi$

If n=1 **then**

Output $\leftarrow$ run(MRQ)

Else

$\{ORQ_1, \ldots, ORQ_m, MRQ'\} \leftarrow MRQ$

For i$\leftarrow$1 **to** m

Output' $\leftarrow$ run(ORQ_i)

Output $\leftarrow$ output $\cup$ output'

Endfor

R $\leftarrow$ CHOOSE_ RELATION(MRQ')

For each tuple t $\in$ R

MRQ" $\leftarrow$ substitute values for t in MRQ'

Output' $\leftarrow$ INGRES-QOA(MRQ")

Output $\leftarrow$ output $\cup$ output'

Endfor

Endif

Query Decomposition

Normalization: to transform the query to a normalized form to facilitate further processing.

- 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

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

where p_{ij} is a simple predicate

Query Decomposition

Equivalence rules

- $p_1 \wedge p_2 \Leftrightarrow p_2 \wedge p_1$
- $p_1 \vee p_2 \Leftrightarrow p_2 \vee p_1$
- $p_1 \wedge (p_2 \wedge p_3) \Leftrightarrow (p_1 \wedge p_2) \wedge p_3$
- $p_1 \vee (p_2 \vee p_3) \Leftrightarrow (p_1 \vee p_2) \vee p_3$
- $p_1 \wedge (p_2 \vee p_3) \Leftrightarrow (p_1 \wedge p_2) \vee (p_1 \wedge p_3)$
- $p_1 \vee (p_2 \wedge p_3) \Leftrightarrow (p_1 \vee p_2) \wedge (p_1 \vee p_3)$
- $\neg(p_1 \vee p_2) \Leftrightarrow \neg p_2 \vee \neg p_1$
- $\neg(p_1 \wedge p_2) \Leftrightarrow \neg p_2 \wedge \neg p_1$
- $\neg(\neg p) \Leftrightarrow p$

Query Decomposition

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);
```

conjunctive normal form is

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

disjunctive normal form is

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

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

Query Decomposition

Analysis: enables rejection of normalized queries for which further processing is either impossible or unnecessary.

- A query is ***type incorrect***: if any of its attribute or relation names are not defined in the global schema, or if operations are being applied to attributes of the wrong type.
- A query is ***semantically incorrect*** if its components do not contribute in any way to the generation of the result.

Hoa Dinh Nguyen, PhD.
Department of Information Technology

Query Decomposition

Example: This query is type incorrect

SELECT E#

FROM EMP

WHERE ENAME > 200;

For 2 reasons:

- Attribute E# is not declared in the schema
- the operation ">200" is incompatible with the type string of ENAME.

Hoa Dinh Nguyen, PhD.
Department of Information Technology

Query Decomposition

- In the context of relational calculus, it is not possible to determine the semantic correctness of general queries.
- It is possible to do so based on the representation of the query as a graph, called a ***query graph*** or ***connection graph***

Query Decomposition

Query graph or ***connection graph***

- One node indicates the result relation, and any other node indicates an operand relation
- An edge between two nodes one of which does not correspond to the result represents a join, whereas an edge whose destination node is the result represents a project
- A non-result node may be labeled by a select or a self-join (join of the relation with itself) predicate
- An important subgraph of the query graph is the ***join graph***, in which only the joins are considered

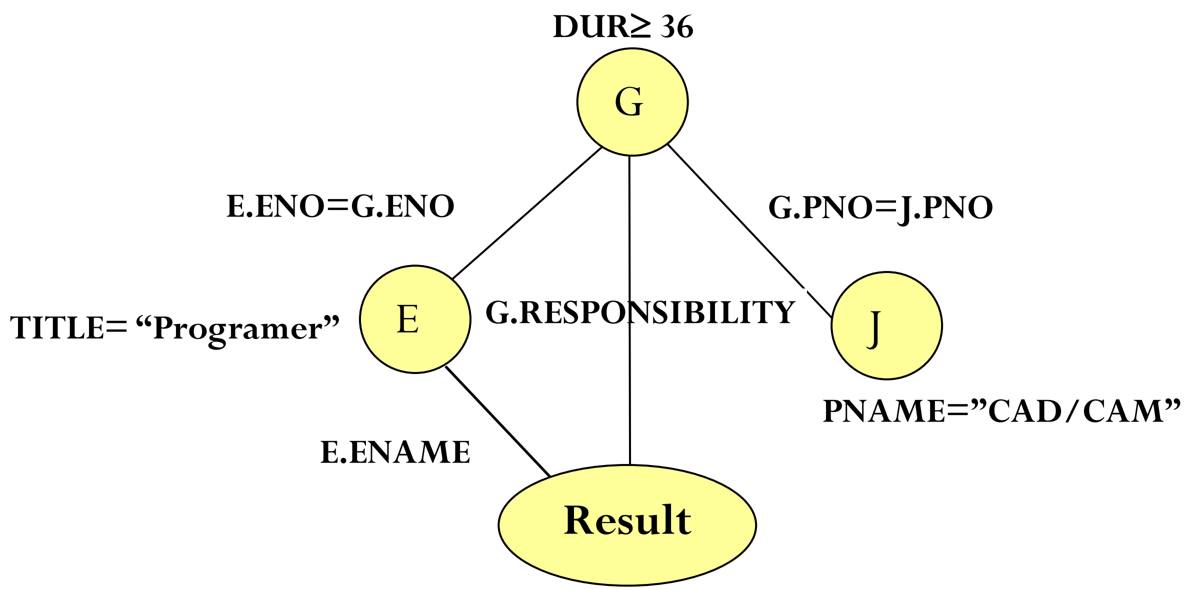
Hoa Dinh Nguyen, PhD.
Department of Information Technology

Query Decomposition

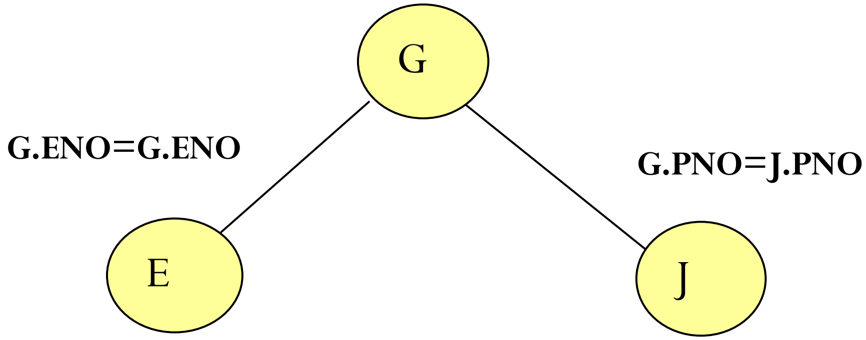
Example: "Find the names and responsibilities of programmers who have been working on the CAD/CAM project for more than 3 years."

SELECT ENAME, RESP
FROM E, G, J
WHERE E.ENO = G.ENO
AND G.PNO = J.PNO
AND PNAME = "CAD/CAM"
AND DUR > 36
AND TITLE = "Programmer";

Query Decomposition



(a) Query graph



(b) Joint graph

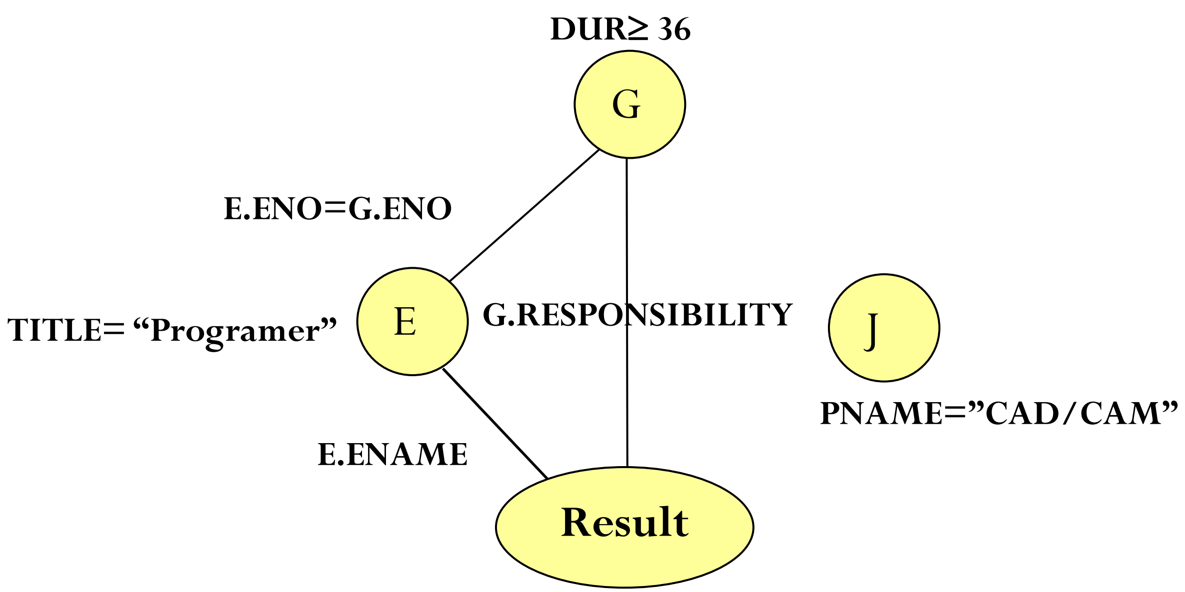
Query Decomposition

Example: Consider the following query

```
SELECT    ENAME, RESP
FROM      E, G, J
WHERE     E.ENO = G.ENO
AND       PNAME = "CAD/CAM"
AND       DUR > 36
AND       TITLE = "Programmer”;
```

Query Decomposition

- Its query graph is disconnected, which tells us that the query is semantically incorrect.



Query graph

Query Decomposition

Solutions to this problem:

- Reject the query
- Assume that there is an implicit Cartesian product between relations G and J,
- Infer (using the schema) the missing join predicate

G.PNO = J.PNO which transforms the query into that of previous Example.

Hoa Dinh Nguyen, PhD.
Department of Information Technology

Query Decomposition

- Elimination of Redundancy: The enriched query qualification may contain ***redundant predicates***. A naive evaluation of a qualification with redundancy can well lead to duplicated work. Such redundancy and thus redundant work may be eliminated by simplifying the qualification with the following well-known idempotency rules:

Hoa Dinh Nguyen, PhD.
Department of Information Technology

Query Decomposition

1. $p \wedge p \Leftrightarrow p$
2. $p \vee true \Leftrightarrow true$
3. $p \vee p \Leftrightarrow p$
4. $p \wedge \neg p \Leftrightarrow false$
5. $p \wedge true \Leftrightarrow p$
6. $p \vee \neg p \Leftrightarrow true$
7. $p \vee false \Leftrightarrow p$
8. $p_1 \wedge (p_1 \vee p_2) \Leftrightarrow p_1$
9. $p \wedge false \Leftrightarrow false$
10. $p_1 \vee (p_1 \wedge p_2) \Leftrightarrow p_1$

Hoa Dinh Nguyen, PhD.
Department of Information Technology

Query Decomposition

Example: the SQL query

SELECT TITLE

FROM E

WHERE (**NOT** (TITLE = "Programmer")

AND (TITLE = "Programmer" **OR** TITLE = "Elect. Eng.")

AND NOT (TITLE = "Elect. Eng."))

OR ENAME = “J. Doe”;

Hoa Dinh Nguyen, PhD.
Department of Information Technology

Query Decomposition

Let:

p_1: TITLE = "Programmer",

p_2: TITLE = "Elect. Eng.",

p_3: ENAME = "J. Doe".

The query application is

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

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

$\Leftrightarrow ((\text{false} \wedge \neg p_2) \vee (\neg p_1 \wedge \text{false}))\vee p_3$

$\Leftrightarrow (\text{false} \vee \text{false}) \vee p_3$

$\Leftrightarrow p_3$

Query Decomposition

The query can be simplified using the previous rules to become

SELECT TITLE

FROM E

WHERE ENAME = "J. Doe";

Query Decomposition

Rewriting: rewrites the query in relational algebra. it is customary 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.

Query Decomposition

The transformation of a tuple relational calculus query into an operator tree can easily be achieved as follows

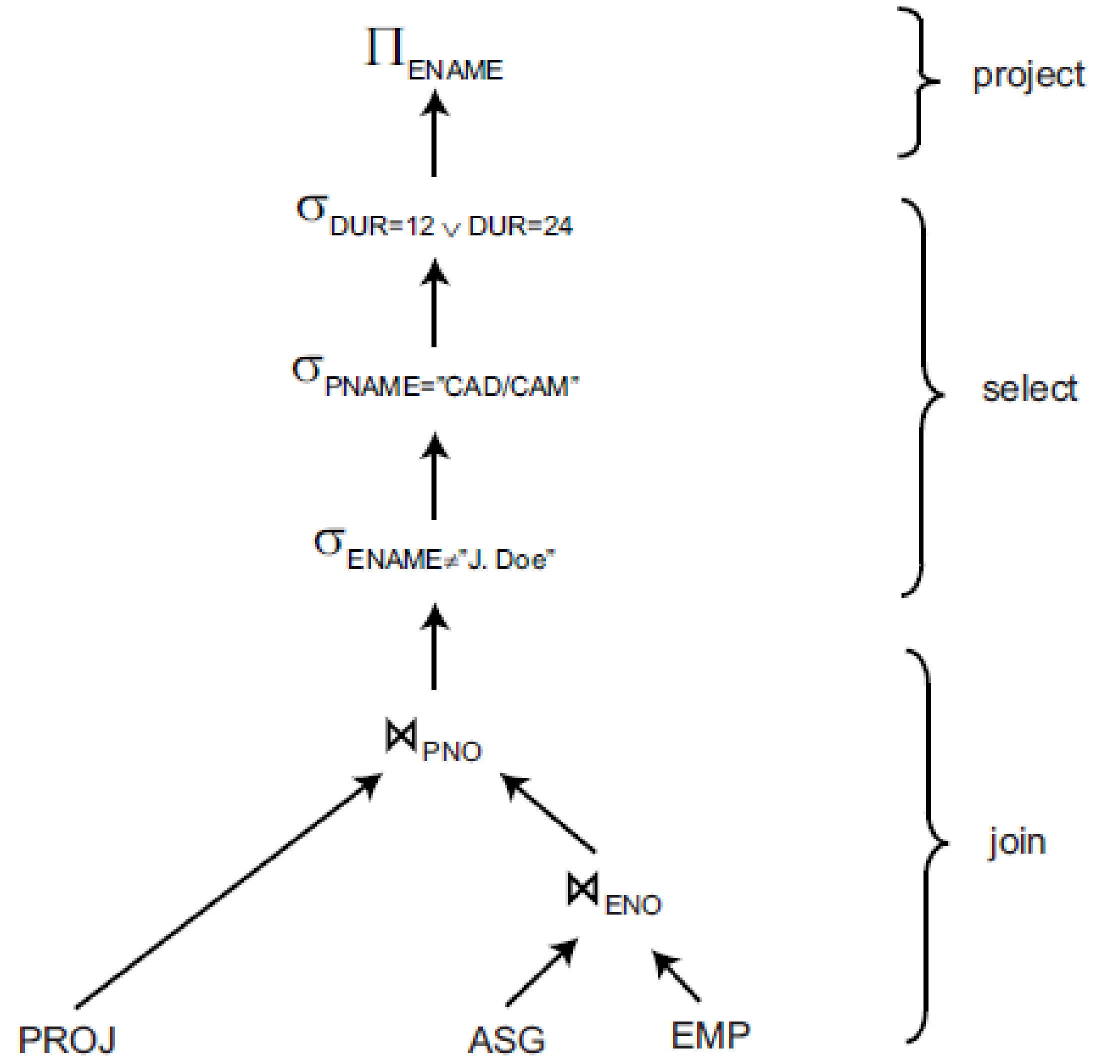
- A different leaf is created for each different tuple variable (corresponding to a relation). In SQL, the leaves are immediately available in the FROM clause.
- The root node is created as a project operation involving the result attributes. These are found in the SELECT clause in SQL.
- The qualification (SQL WHERE clause) is translated into the appropriate sequence of relational operations (select, join, union, etc.) going from the leaves to the root.

Query Decomposition

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

```
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) ;
```

Query Decomposition



Query Decomposition

1. Commutativity of binary operators

$R \times S \Leftrightarrow S \times R$

$R \bowtie S \Leftrightarrow S \bowtie R$

2. Associativity of binary operators

$R \times (S \times T) \Leftrightarrow (R \times S) \times T$

$R \bowtie (S \bowtie T) \Leftrightarrow (R \bowtie S) \bowtie T$

3. Idempotence of unary operators

$\Pi_{A'}(\Pi_{A''}(R)) \Leftrightarrow \Pi_{A'}(R)$, where A', A" $\subseteq$R and A' $\subseteq$ A"

$$\sigma_{p_1(A_1)}(\sigma_{p_2(A_2)}(R)) = \sigma_{p_1(A_1) \wedge p_2(A_2)}(R)$$

Query Decomposition

4. Commuting selection with projection

$$\Pi_{A_1,\ldots,A_n}(\sigma_{p(A_p)}(R)) = \Pi_{A_1,\ldots,A_n}(\sigma_{p(A_p)}(\Pi_{A_1,\ldots,A_n,A_p}(R)))$$

- Note that if Ap is already a member of $\{A_1,\ldots,A_n\}$, the last projection on $[A_1,\ldots,A_n]$ on the right-hand side of the equality is useless.

$$\Pi_{A_1,\ldots,A_n}(\sigma_{p(A_p)}(R)) = \sigma_{p(A_p)}(\Pi_{A_1,\ldots,A_n,A_p}(R))$$

Query Decomposition

5. Commuting selection with binary operators

$$\sigma_{p(A_p)}(R \times S) \Leftrightarrow \sigma_{p(A_p)}(R) \times S$$

$$\sigma_{p(A_i)}(R \bowtie_{p(A_i, B_k)} S) \Leftrightarrow \sigma_{p(A_i)}(R) \bowtie_{p(A_i, B_k)} \mathrm{S}$$

$$\sigma_{p(A_i)}(R \cup T) \Leftrightarrow \sigma_{p(A_i)}(R) \cup \sigma_{p(A_i)}(T)$$

Query Decomposition

www.ptit.edu.vn

6. Commuting projection with binary operators

- If C=A'∪B', where A'⊆ A, B'⊆ B, and A, B are the sets of attributes over which relations R and S, respectively, are defined, we have

$$\Pi_C(R \times S) = \Pi_{A'}(R) \times \Pi_{B'}(S)$$

$$\Pi_C(R \bowtie_{p(A_i, B_j)} S) = \Pi_{A'}(R) \bowtie_{p(A_i, B_j)} \Pi_{B'}(S)$$

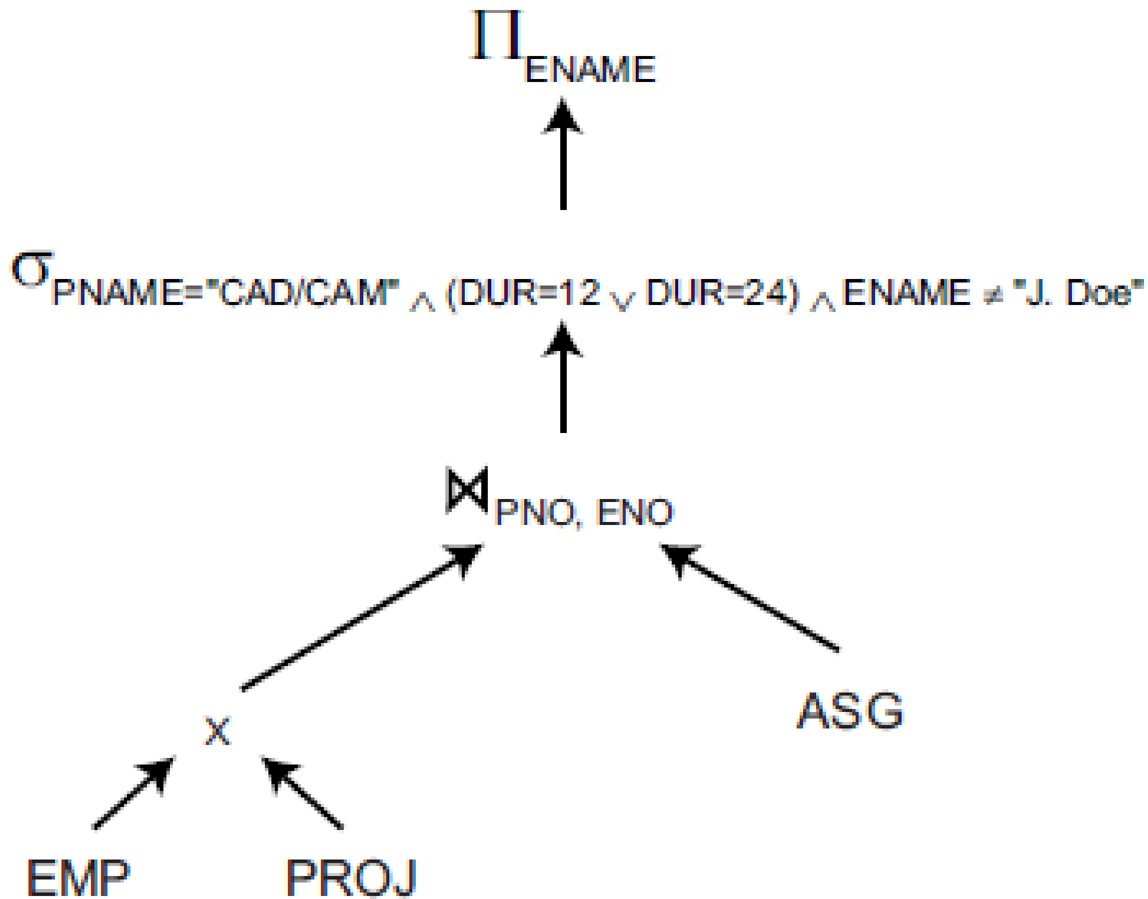
$$\Pi_C(R \cup S) = \Pi_{A'}(R) \cup \Pi_{B'}(S)$$

Query Decomposition

These rules can be used in four different ways:

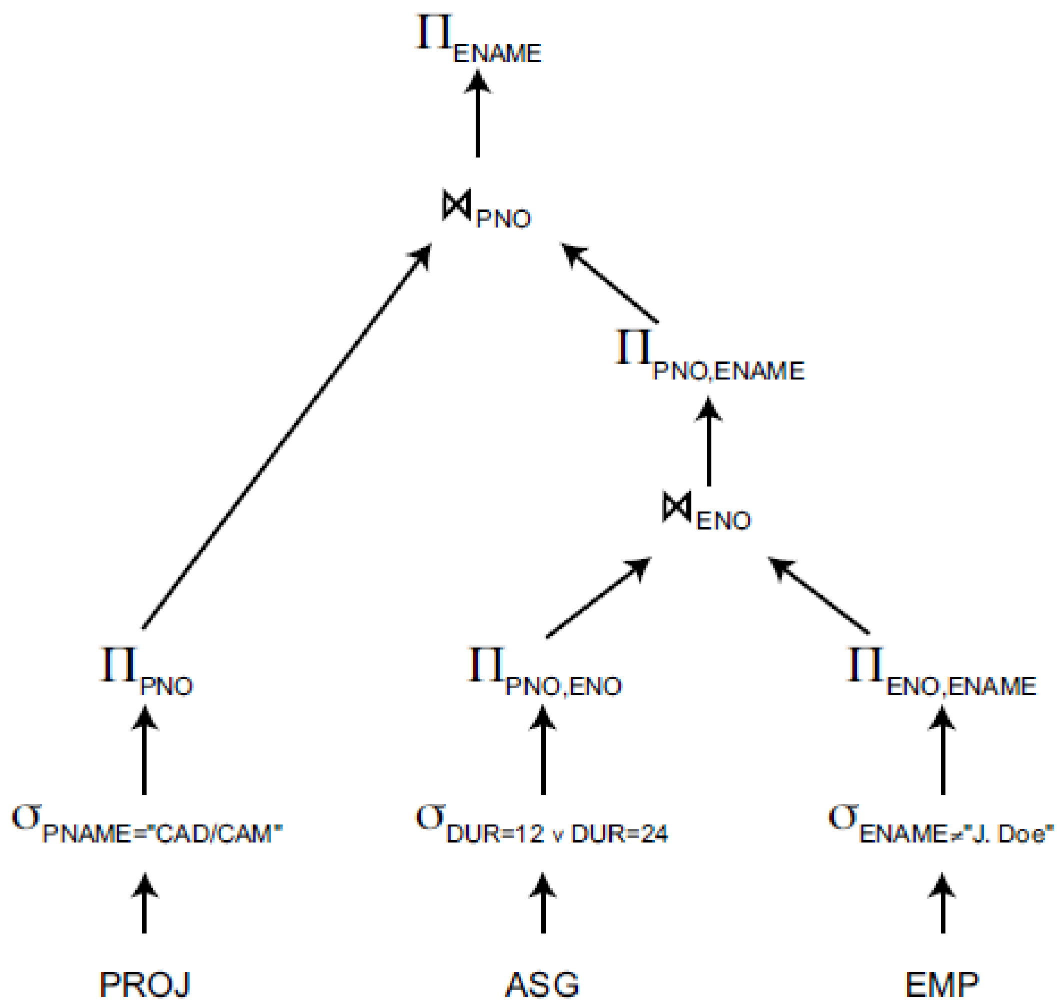
- First, they allow the separation of the unary operations, simplifying the query expression.
- Second, unary operations on the same relation may be grouped so that access to a relation for performing unary operations can be done only once.
- Third, unary operations can be commuted with binary operations so that some operations (e.g., selection) may be done first.
- Fourth, the binary operations can be ordered.

Query Decomposition



Equivalent Operator Tree

Query Decomposition



Rewritten Operator Tree

Localization of Distributed Data

- General techniques for decomposing and restructuring queries apply to both centralized and distributed DBMSs and do not take into account the distribution of data.
- 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
- 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***.

Localization of Distributed Data

- A naive way to localize a distributed query is to generate a query where each global relation is substituted by its localization program.
- ***Localized query***: replacing the leaves of the operator tree of the distributed query with subtrees corresponding to the localization programs
- ***Reduction techniques***: generate simpler and optimized queries by using the transformation rules and the heuristics, such as pushing unary operations down the tree

Reduction for PHF

- The horizontal fragmentation function distributes a relation based on selection predicates

Example: Relation EMP(ENO, ENAME, TITLE) can be split into three horizontal fragments EMP1, EMP2, and EMP3

1. $\text{EMP1} = \sigma_{ENO \leq "E3"}(\text{EMP})$
2. $\text{EMP2} = \sigma_{"E3" < ENO \leq "E6"}(\text{EMP})$
3. $\text{EMP3} = \sigma_{ENO > "E6"}(\text{EMP})$

- The localization program for an horizontally fragmented relation is the union of the fragments

$$\text{EMP} = \text{EMP1} \cup \text{EMP2} \cup \text{EMP3}$$

Reduction with Selection

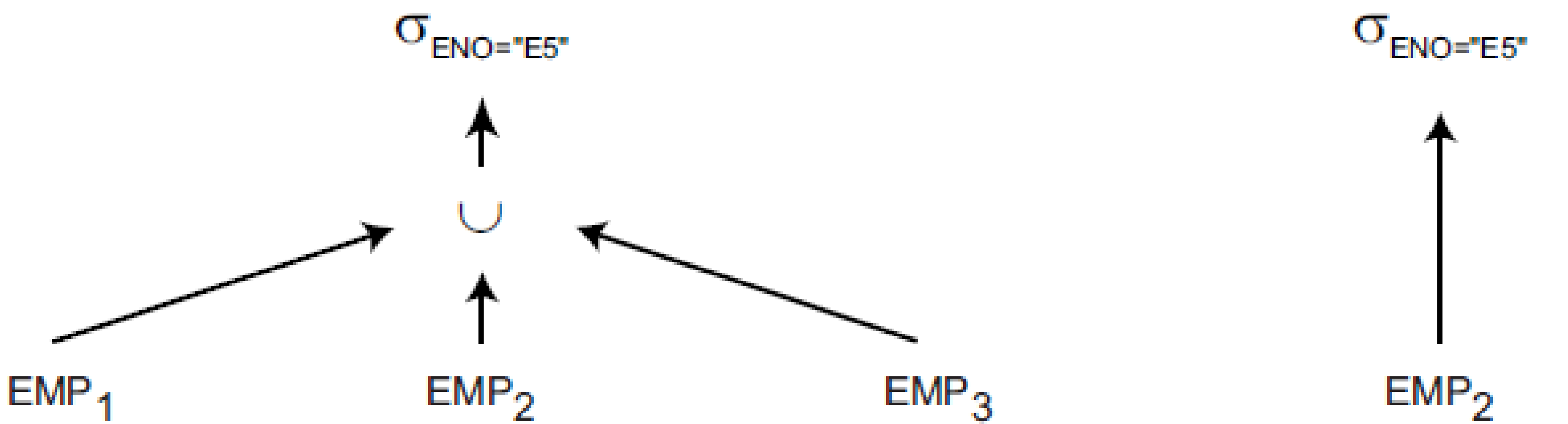
- Selections on fragments that have a qualification contradicting the qualification of the fragmentation rule generate empty relations.
- Given a relation R that has been horizontally fragmented as $R_1, R_2, \ldots, R_N$, where $R_i = \sigma_{p_i}(R)$, the rule can be stated formally as follows:
- **Rule 1**: $\sigma_{p_j}(R_i) = \emptyset$ if $\forall x$ in R: $\neg\big(p_i(x) \wedge p_j(x)\big)$

where p_i and p_j are selection predicates, x denotes a tuple, and p(x) denotes "predicate p holds for x."

Reduction with Selection

- Example:

SELECT *
FROM EMP
WHERE ENO = “E5” ;



(a) Localized query

(b) Reduced query

Reduction with Join

- Joins on horizontally fragmented relations can be simplified when the joined relations are fragmented according to the join attribute.
- The simplification consists of distributing joins over unions and eliminating useless joins

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

where R_i are fragments of R and S is a relation.

- **Rule 2**: $R_i \bowtie R_j = \emptyset$ if $\forall x$ in R_i, $\forall y$ in R_j : $\neg\left(p_i(x) \wedge p_j(y)\right)$

fragments R_i and R_j are defined, respectively, according to predicates p_i and p_j on the same attribute

Reduction with Join

- Example: Assume that relation EMP is fragmented between EMP1, EMP2, and EMP3, as above, and that relation ASG is fragmented as

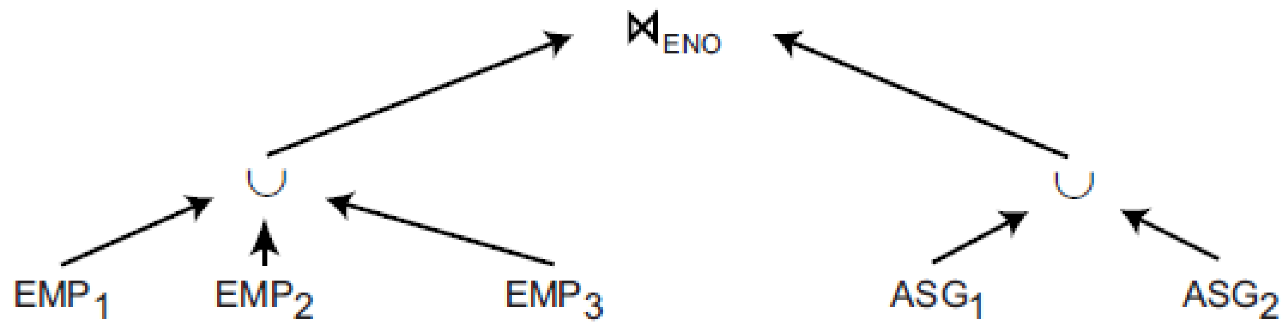
1. ASG1 $= \sigma_{ENO \leq "E3"}$ (ASG)
2. ASG2 $= \sigma_{ENO > "E3"}$ (ASG)

- Consider the join query:

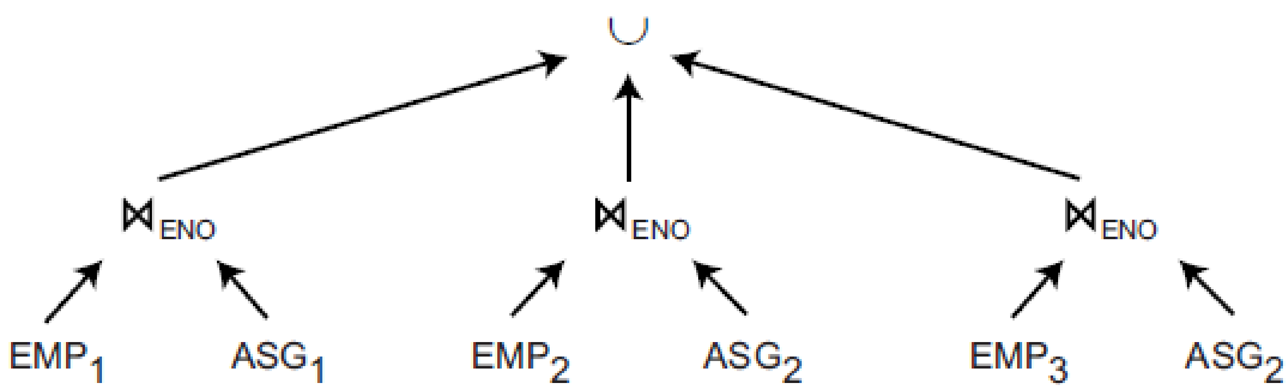
```
SELECT   *
FROM     EMP, ASG
WHERE    EMP.ENO = ASG.ENO;
```


Hoa Dinh Nguyen, PhD.
Department of Information Technology

Reduction with Join



(a) Localized query



(b) Reduced query

Hoa Dinh Nguyen, PhD.
Department of Information Technology

Reduction for Vertical Fragmentation

- The vertical fragmentation function distributes a relation based on projection attributes
- The localization program for a VF 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:

1. $\text{EMP1} = \prod_{ENO,ENAME}(\text{EMP})$
2. $\text{EMP2} = \prod_{ENO,TITLE}(\text{EMP})$

- The localization program is

$$\text{EMP} = \text{EMP1} \bowtie_{ENO} \text{EMP2}$$

Reduction for Vertical Fragmentation

- Queries on vertical fragments can be reduced by determining the useless intermediate relations and removing the subtrees that produce them
- Projections 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.
- **Rule 3**: $\prod_{D,K}(R_i)$ is useless if the set of projection attributes D is not in A'

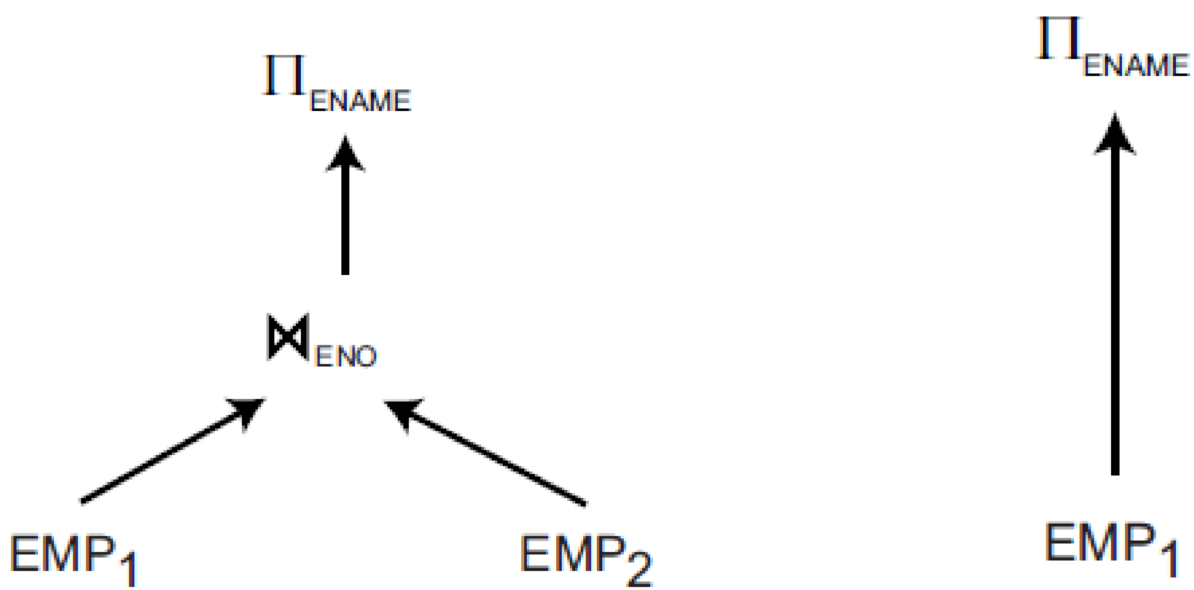
Where relation R($A_1, A_2, \ldots, A_n$) is vertically fragmented as $R_i = \prod_{A'}(R)$, and $A' \subseteq \{A_1, A_2, \ldots, A_n\}$

Hoa Dinh Nguyen, PhD.
Department of Information Technology

Reduction for Vertical Fragmentation

- Example: consider the query

SELECT ENAME
FROM EMP



(a) Localized query (b) Reduced query

Reduction for Derived Fragmentation

- If relation R is subject to derived horizontal fragmentation due to relation S, the fragments of R and S that have the same join attribute values are located at the same site.
- S can be fragmented according to a selection predicate.
- Derived fragmentation should be used only for one-to-many (hierarchical) relationships of the form S → R, where a tuple of S can match with *n* tuples of R, but a tuple of R matches with exactly one tuple of S.

Note that derived fragmentation could be used for many-to-many relationships provided that tuples of S (that match with n tuples of R) are replicated.

Reduction for Derived Fragmentation

- 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:

1. $\text{ASG1} = \text{ASG} \bowtie_{ENO} \text{EMP1}$
2. $\text{ASG2} = \text{ASG} \bowtie_{ENO} \text{EMP2}$

Where: $\text{EMP1} = \sigma_{TITLE="Programmer"}(\text{EMP})$

$$\text{EMP2} = \sigma_{TITLE\neq"Programmer"}(\text{EMP})$$

- The localization program for a horizontally fragmented relation is the union of the fragments

$$\text{ASG} = \text{ASG1} \cup \text{ASG2}$$

Reduction for Derived Fragmentation

- Queries on derived fragments can also be reduced: certain joins will produce empty relations if the fragmentation predicates conflict.
- Example: the predicates of ASG1 and EMP2 conflict, thus

$$ASG1 \bowtie EMP2 = \emptyset$$

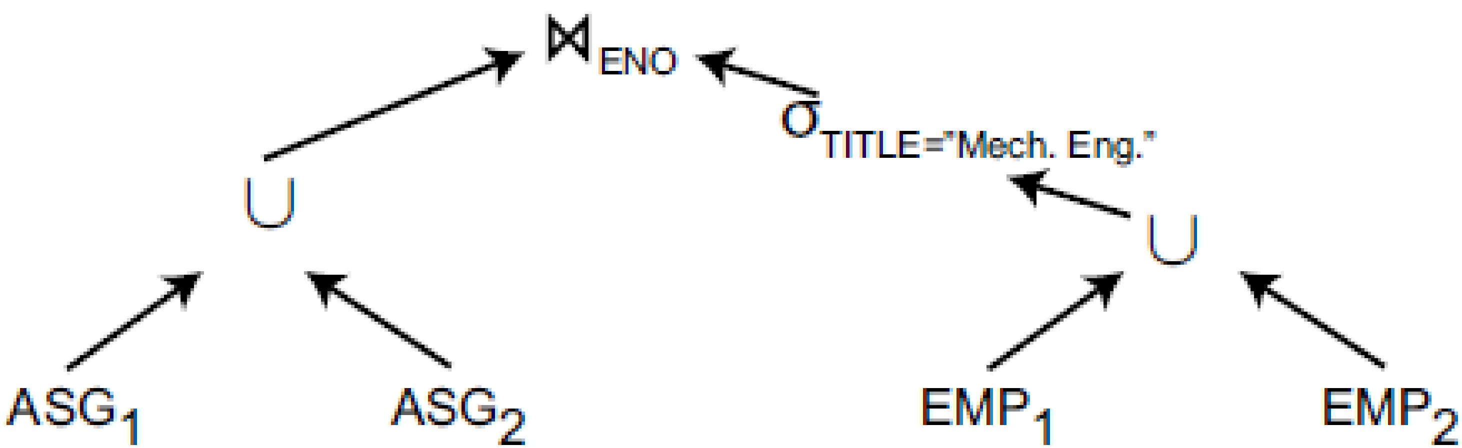
- The reduced query is always preferable to the localized query because the number of partial joins usually equals the number of fragments of R.

Hoa Dinh Nguyen, PhD.
Department of Information Technology

Reduction for Derived Fragmentation

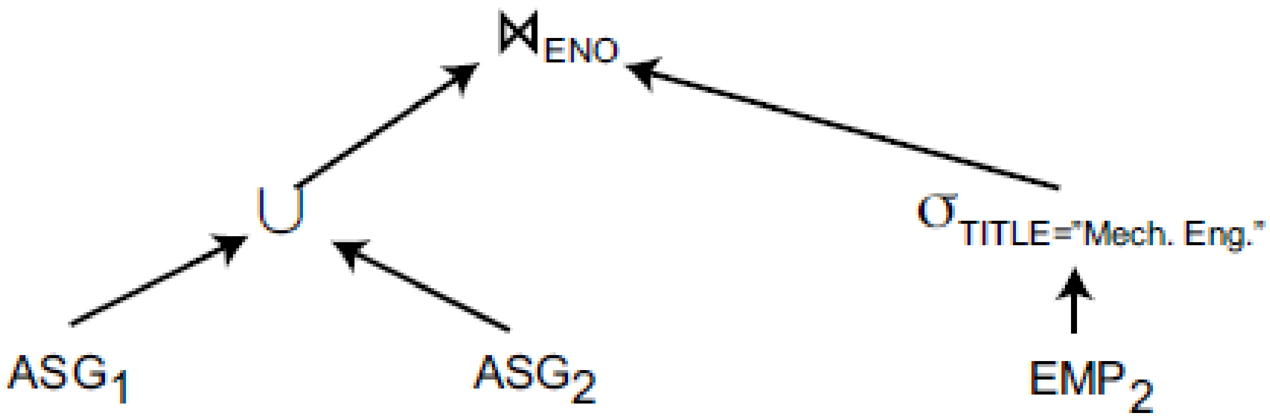
- Example:

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

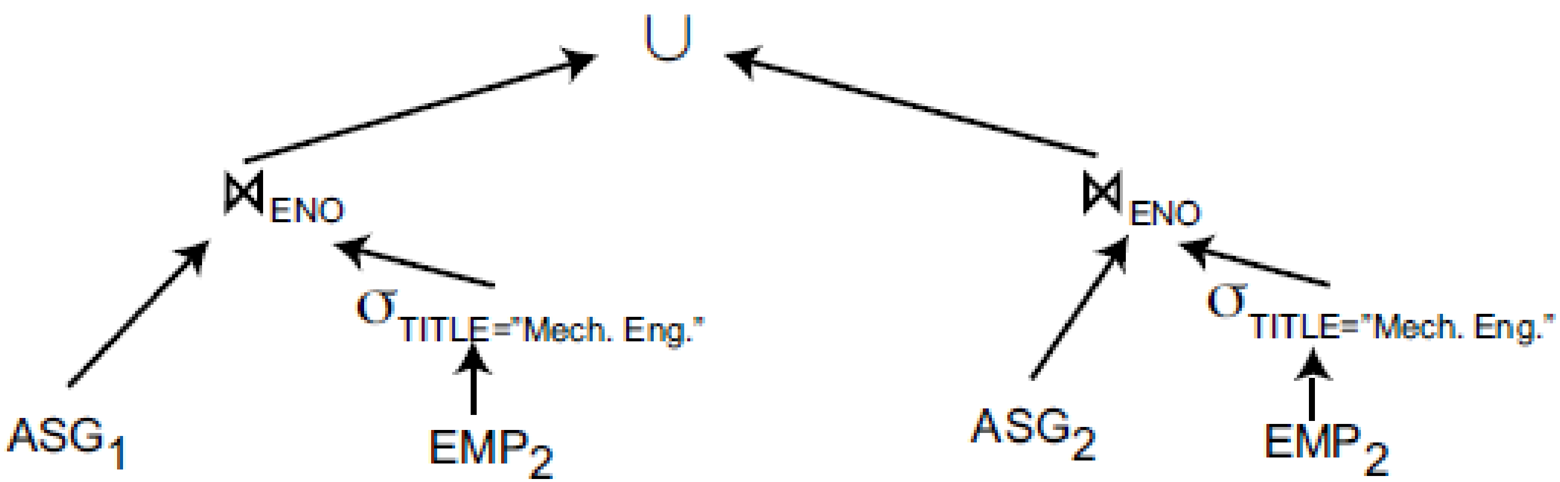


(a) Localized query

Reduction for Derived Fragmentation



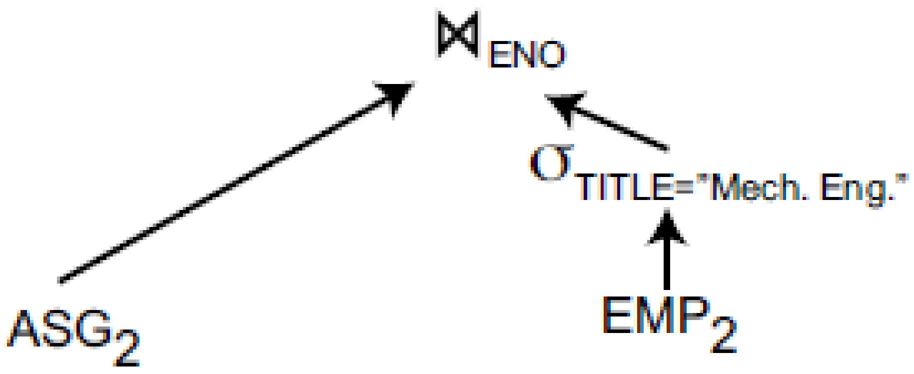
(b) Query after pushing selection down



(c) Query after moving unions up

Hoa Dinh Nguyen, PhD.
Department of Information Technology

Reduction for Derived Fragmentation



(d) Reduced query after eliminating the left subtree

Reduction for Hybrid Fragmentation

- The goal of hybrid fragmentation is to efficiently support queries involving projection, selection, and join.
- The optimization of an operation or of a combination of operations is always done at the expense of other operations.
- The localization program for a hybrid fragmented relation uses unions and joins of fragments

Hoa Dinh Nguyen, PhD.
Department of Information Technology

Reduction for Hybrid Fragmentation

- Example:

1. EMP1 $= \sigma_{ENO \leq "E4"}\left(\prod_{ENO,ENAME}(EMP)\right)$
2. EMP2 $= \sigma_{ENO > "E4"}\left(\prod_{ENO,ENAME}(EMP)\right)$
3. EMP3 $= \prod_{ENO,TITLE}(EMP)$

The localization program is

$$EMP = (EMP1 \cup EMP2) \bowtie_{ENO} EMP3$$

Reduction for Hybrid Fragmentation

- Queries on hybrid fragments can be reduced by combining the rules:

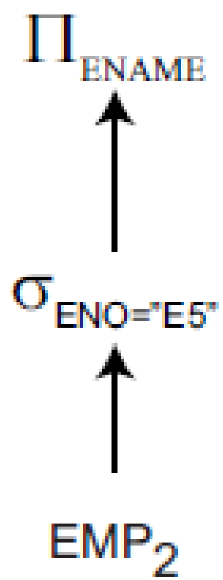
1. Remove empty relations generated by contradicting selections on horizontal fragments.
2. Remove useless relations generated by projections on vertical fragments.
3. Distribute joins over unions in order to isolate and remove useless joins.

Hoa Dinh Nguyen, PhD.
Department of Information Technology

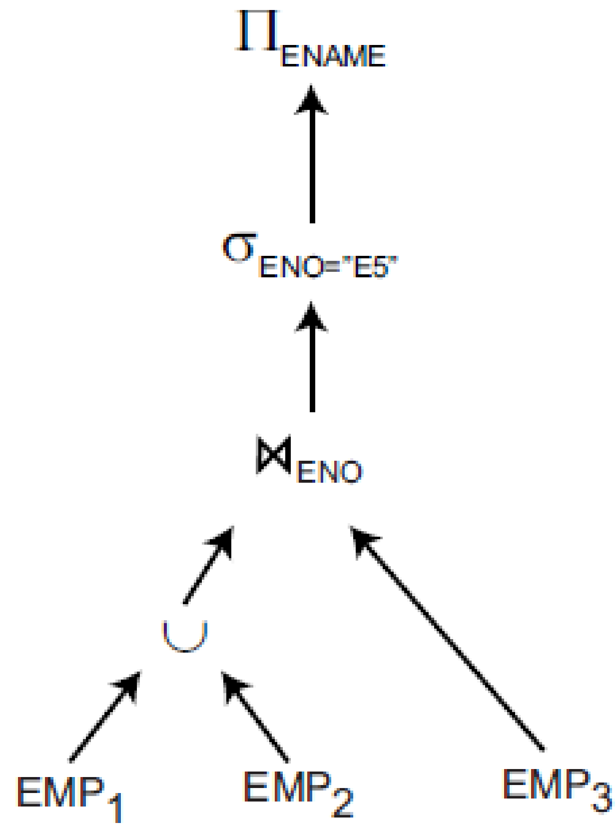
Reduction for Hybrid Fragmentation

- Example: application of rules (1) and (2)

SELECT ENAME
FROM EMP
WHERE ENO="E5"



(b) Reduced query



(a) Localized query

Hoa Dinh Nguyen, PhD.
Department of Information Technology

Query Optimization in DDBS

- Query decomposition and data localization are the two successive functions that map a calculus query, expressed on distributed relations, into an algebraic query (query decomposition), expressed on relation fragments (data localization).
- Query decomposition can generate an algebraic query simply by translating into relational operations the predicates and the target statement as they appear.
- Data localization can, in turn, express this algebraic query on relation fragments, by substituting for each distributed relation an algebraic query corresponding to its fragmentation rules.