BITS Pilani

BITS Pilani
Hyderabad Campus

Dr. Manik Gupta
Associate Professor
Department of CSIS

innovate achieve lead

BITS Pilani
Hyderabad Campus

Distributed Data Systems (CS G544)
Lecture 15-1

Thursday, 19th Sept 2024

Lecture Recap

- Overview of Query Processing

Introduction

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

Query decomposition

- The successive steps of Query decomposition are:
  - Normalization
  - Analysis
  - Elimination of redundancy
  - Rewriting

Step 1 - Normalization

- The input query may be arbitrarily complex depending on the facilities provided by the language.
- It is the goal of normalization to transform the query to a normalized form to facilitate further processing.
- In SQL the most important transformation is that of query qualification (the WHERE clause), which may be an arbitrarily complex, quantifier-free predicate, preceded by all necessary quantifiers.

Normal forms

– The conjunctive normal form is the conjunction (Λ predicate) of disjunctions (V predicate).

$$(p_{11} \vee p_{12} \vee p_{13} \vee p_{1n}) \wedge \ldots \wedge (p_{m1} \vee p_{m2} \vee p_{m3} \vee p_{mn})$$

– The disjunctive normal form is the disjunction (V predicate) of conjunctions (Λ predicate)

$$(p_{11} \wedge p_{12} \wedge p_{13} \wedge p_{1n}) \vee \ldots \vee (p_{m1} \wedge p_{m2} \wedge p_{m3} \wedge p_{mn})$$

Normalization

– The transformation of the quantifier-free predicate is straightforward using the well-known equivalence rules for logical operations.

1. $p_1 \wedge p_2 \Leftrightarrow p_2 \wedge p_1$
2. $p_1 \vee p_2 \Leftrightarrow p_2 \vee p_1$
3. $p_1 \wedge (p_2 \wedge p_3) \Leftrightarrow (p_1 \wedge p_2) \wedge p_3$
4. $p_1 \vee (p_2 \vee p_3) \Leftrightarrow (p_1 \vee p_2) \vee p_3$
5. $p_1 \wedge (p_2 \vee p_3) \Leftrightarrow (p_1 \wedge p_2) \vee (p_1 \wedge p_3)$
6. $p_1 \vee (p_2 \wedge p_3) \Leftrightarrow (p_1 \vee p_2) \wedge (p_1 \vee p_3)$
7. $\neg(p_1 \wedge p_2) \Leftrightarrow \neg p_1 \vee \neg p_2$
8. $\neg(p_1 \vee p_2) \Leftrightarrow \neg p_1 \wedge \neg p_2$
9. $\neg(\neg p) \Leftrightarrow p$

Example

EMP(ENO, ENAME, TITLE)

PROJ(PNO, PNAME, BUDGET)

ASG(ENO, PNO, DUR,RESP)

PAY(TITLE, SAL)

- Find the names of employees who have been working on 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
```

Example

Qualification in conjunctive NF is

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

Qualification in disjunctive NF is

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

Which one leads to redundant work?

The latter form lead to redundant work because common sub expressions are not eliminated

Step 2 - Analysis

- Query analysis enables rejection of normalized queries for which further processing is impossible or not necessary.
- The main reasons for rejection are:
  - Type incorrectness:
    - A query is **type incorrect** if any of its attribute or relation names are not defined in the global schema.
    - Or if the operations are being applied to attributes of wrong type.
  - Semantic incorrectness:
    - If components of query do not contribute to the results.
    - It is not possible to determine semantic correctness of general queries

Analysis

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

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

  - First, attribute E# is not declared in the schema.
  - Second, the operation “>200” is incompatible with the type string of ENAME.

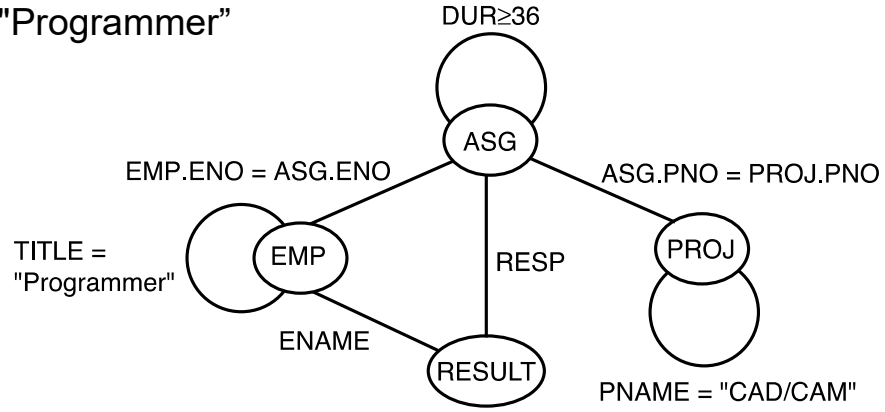
Query graph

- Query graph or connection graph can be used to determine the semantic correctness.
- In a query 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.**
- Furthermore, 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. The join graph is particularly useful in the query optimization phase.

Example

- “Find the names and responsibilities of programmers who have been working on the CAD/CAM project for more than 3 years.”
- The query expressed in SQL is

```
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”
```



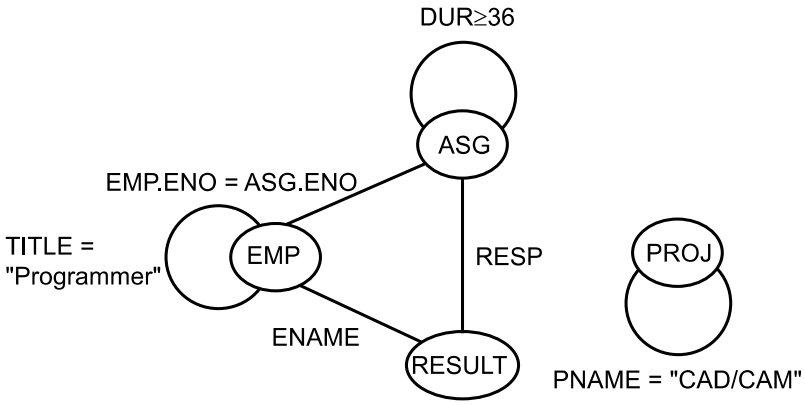
How is query graph useful?

- The query graph is useful to determine the **semantic correctness** of a conjunctive multivariable query without negation.
- Such a query is **semantically incorrect** if its **query graph is not connected**. In this case one or more subgraphs (corresponding to subqueries) are disconnected from the graph that contains the result relation.

Disconnected Query graph

Let us consider the following SQL query:

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



Problem is that join predicates are missing and the query should be rejected. What are the other solutions to fix it?

Step 3 - Elimination of redundancy

- Query qualification may contain redundant predicates.
- Redundancy can be eliminated by applying 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. $p \wedge \neg p \Leftrightarrow false$
8. $p \vee \neg p \Leftrightarrow true$
9. $p_1 \wedge (p_1 \vee p_2) \Leftrightarrow p_1$
10. $p_1 \vee (p_1 \wedge p_2) \Leftrightarrow p_1$

Example

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

Can be written as

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

Let $p1$ be TITLE = “Programmer”,

$p2$ be TITLE = “Elect. Eng.”, and

$p3$ be ENAME = “J. Doe”.

The query qualification is $(\neg p1 \wedge (p1 \vee p2) \wedge \neg p2) \vee p3$

Example

The query qualification is $(\neg p1 \wedge (p1 \vee p2) \wedge \neg p2) \vee p3$

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

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

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

$$false \vee false \vee p_3$$

Step 4 - Rewriting

– This step of decomposition rewrites the query in relational algebra.

Operator Tree

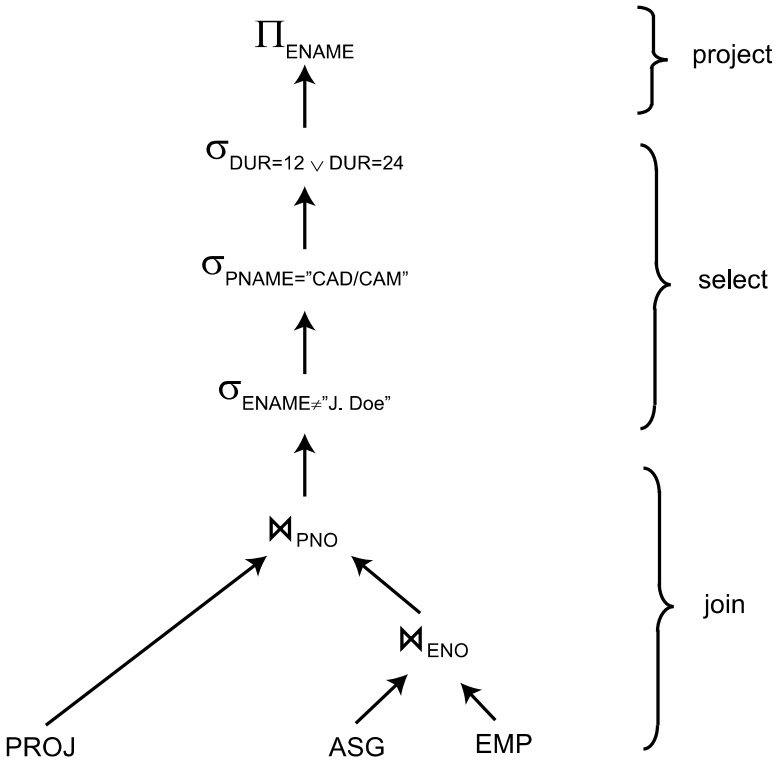
- It is customary to represent the relational algebra query graphically by an *operator tree*.
- An operator tree is a tree in which a leaf node is a relation stored in the database, and a non-leaf node is an intermediate relation produced by a relational algebra operator.
- The sequence of operations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

Example

- “Find the names of employees other than J. Doe who worked on the CAD/CAM project for either one or two years” whose SQL expression is

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

Example of Operator Tree



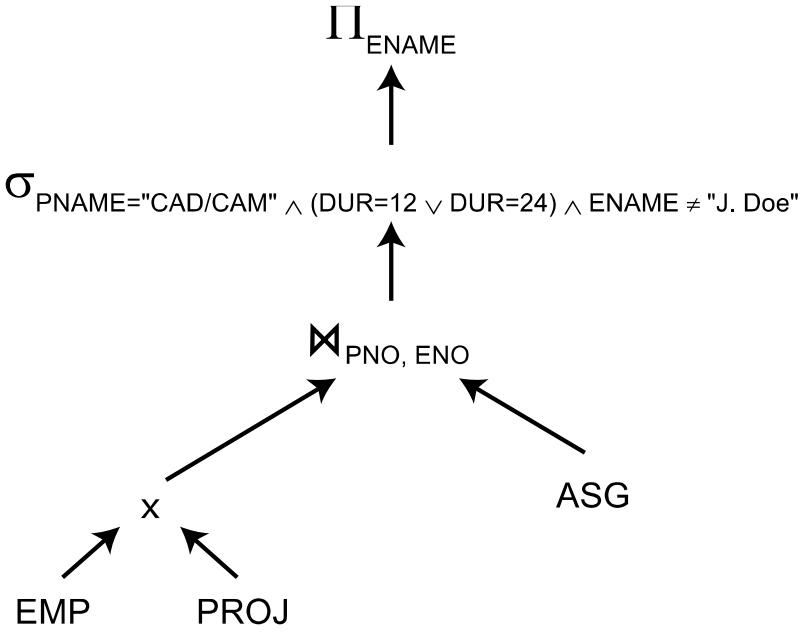
- The transformation of a tuple relational calculus query into an operator tree can easily be achieved as follows.
  - First, a different leaf is created for each different tuple variable (corresponding to a relation). In SQL, the leaves are immediately available in the **FROM clause**.
  - Second, the root node is created as a project operation involving the result attributes. These are found in the **SELECT clause** in SQL.
  - Third, 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. The sequence can be given directly by the order of appearance of the predicates and operators.

Transformation rules

- Application of six rules enable generation of many equivalent trees
  - Commutativity of binary operators
  - Associativity of binary operators
  - Idempotence of unary operators
  - Commuting selection with projection
  - Commuting selection with binary operators
  - Commuting projection with binary operators

Exercise: To study the expressions from textbook

Equivalent Operator Tree



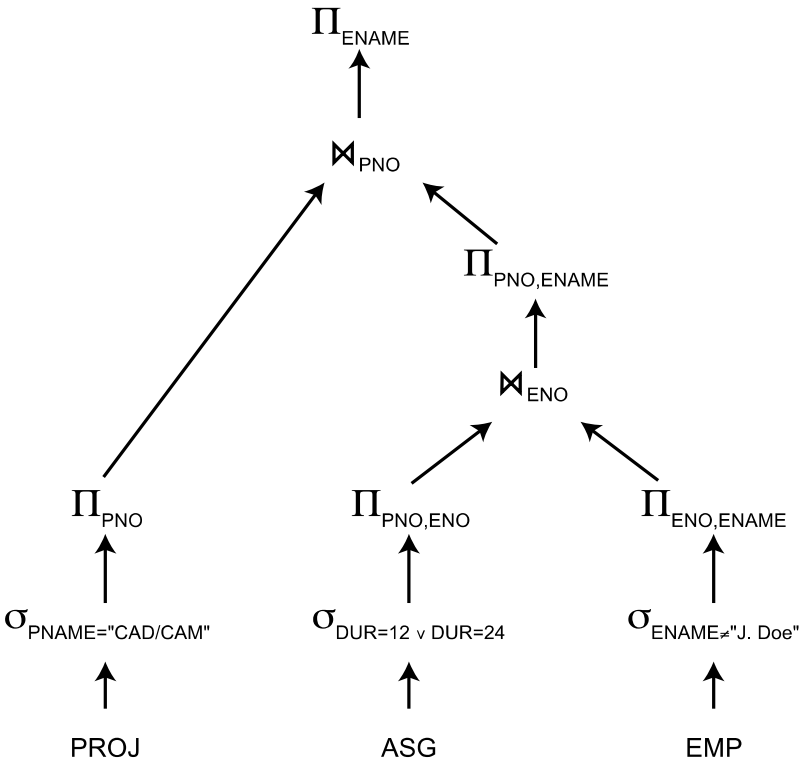
Is this tree good?

Requires a Cartesian product of relations EMP and PROJ, and may lead to a higher execution cost than the original tree

Transformation rules

- The application of the six rules enables the **generation of many equivalent trees** and can be used restructure the tree in a systematic way so that the **“bad” operator trees are eliminated.**
  - 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.**

Rewritten Operator Tree



Why is tree good?

Repeated access to the same relation is avoided and that the most selective operations are done first.
However, this tree is far from optimal. For example, the select operation on EMP is not very useful before the join because it does not greatly reduce the size of the operand relation.

Localization of distributed data

- So far, in the query decomposition, we have not considered the data distribution into account. This is the role of localization layer.
- The localization layer translates an relational algebra query on global relations into relational algebraic query expressed on physical fragments.
- Localization uses the information stored in fragment schema.

Localization of distributed data

- Input: Relational algebra expression on global, distributed relations (distributed query)
- Determine which fragments are involved in a query (over global, distributed relations) and transform a query into an equivalent one over such fragments (localized query)

Localization of distributed data

- Recall that fragmentation is obtained by several application of rules expressed by relational algebra
  - primary horizontal fragmentation: selection σ
  - derived horizontal fragmentation: semijoin ⋉
  - vertical fragmentation: projection Π
- Reconstruction (reverse fragmentation) rules are also expressed in relational algebra
  - horizontal fragmentation: union ∪
  - vertical fragmentation: join ⋈

Localization program

- 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

- **Localization program:** Relational algebra expression that **reconstructs a global relation from its fragments**, by reverting the rules employed for fragmentation
- A localized query is obtained from distributed, global query by replacing leaves (global relations) with (the tree of) its corresponding localization program

  ➡ Leaves of localized queries are fragments

Localization of distributed data

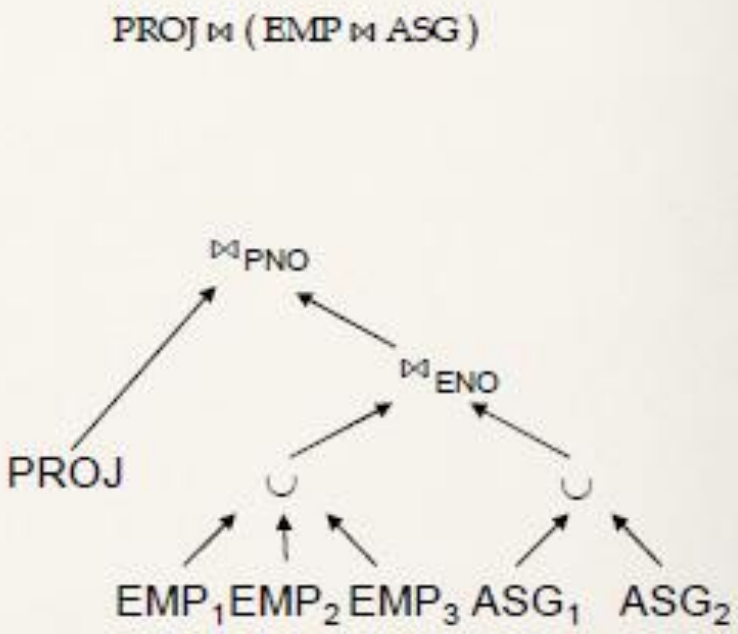
- A naïve way of localizing a distributed query is to generate a query where **each global relation is substituted by it localization program**.
- This can be viewed as replacing the leaves of the operator tree of the distributed query with subtrees corresponding to the localization programs to obtain the **localized query**.

Example

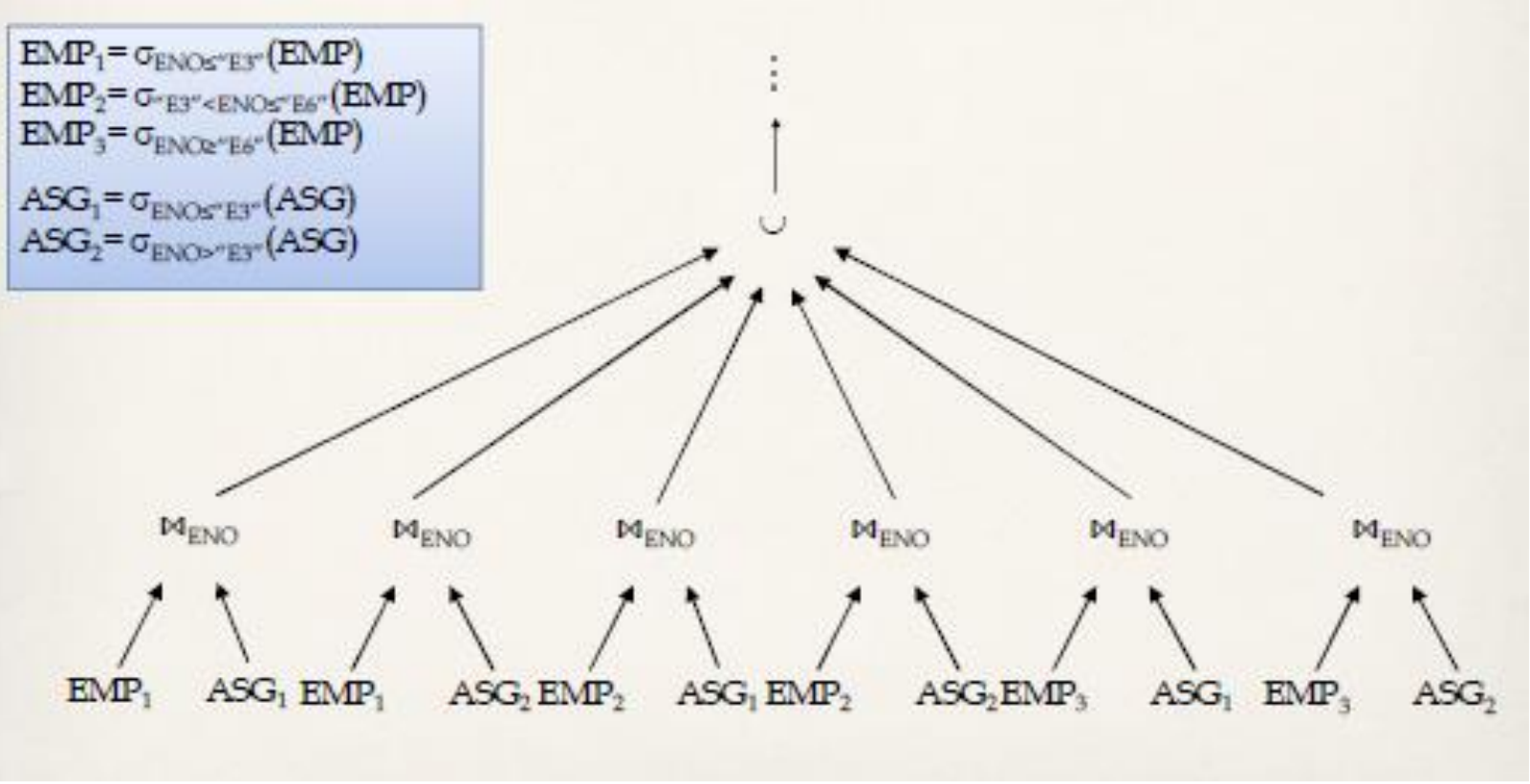
Assume

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

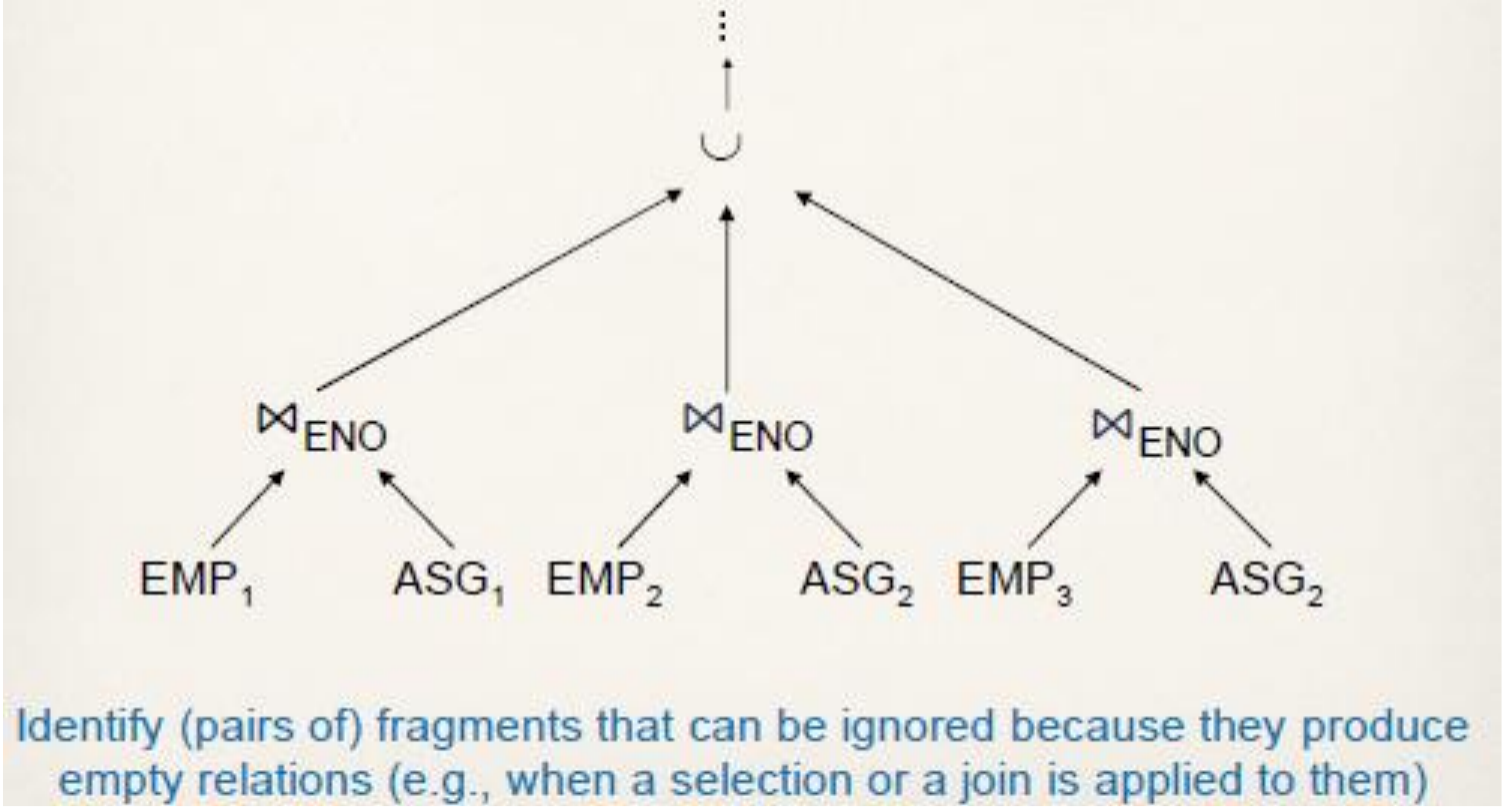
Replace EMP by $(EMP_1 \cup EMP_2 \cup EMP_3)$ and ASG by $(ASG_1 \cup ASG_2)$ in any query



Provides Parallellism



Eliminates Unnecessary Work



Reduction for primary horizontal fragmentation

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

  EMP(eno, ename, title)

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

  $EMP_2 = \sigma_{"E3" < eno \geq "E6"}(EMP)$

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

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

  $EMP = EMP_1 \cup EMP_2 \cup EMP_3$

- Thus the localized form of any query specified on EMP is obtained by replacing it by

  $EMP_1 \cup EMP_2 \cup EMP_3$

Reduction for primary horizontal fragmentation

- The reduction of queries on horizontally fragmented relations consists primarily of determining, after restructuring the subtrees, those that will produce empty relations, and removing them.
- Horizontal fragmentation can be exploited to simplify both **selection and join** operations.

Reduction for PHF with Selection

- Selections on fragments that have a qualification contradicting the qualification of the fragmentation rule generate empty relations.

Reduction for PHF with selection

Reduction with selection (ignore a fragment if selection predicate and fragment predicate are contradictory)

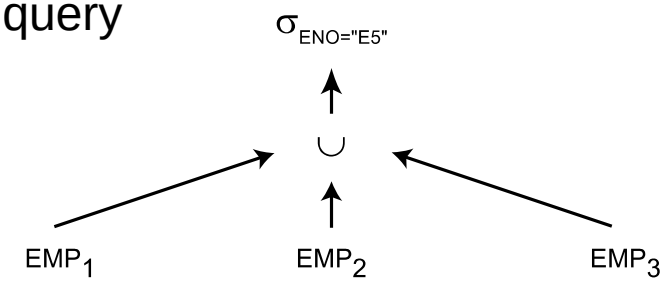
Relation R and $F_R=\{R_1, R_2, \ldots, R_w\}$ where $R_j=\sigma_{p_j}(R)$

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

Example - Reduction for PHF with selection

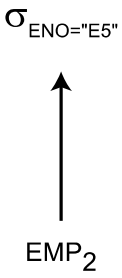
Select * from EMP where eno=“E5”;

- Applying the naive approach to localize EMP from EMP_1, EMP_2, and EMP_3 gives the localized query



(a) Localized query

- By **commuting the selection with the union operation**, it is easy to detect that the selection predicate contradicts the predicates of EMP_1 and EMP_3, thereby producing empty relations. The reduced query is simply applied to EMP_2



(b) Reduced query

Reduction for PHF 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.

- The distribution of join over union can be stated as:

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

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

- With this transformation, unions can be moved up in the operator tree so that all possible joins of fragments are exhibited.
- Useless joins of fragments can be determined when the qualifications of the joined fragments are contradicting, thus yielding an empty result.

Reduction for PHF with Join

Reduction with join (ignore the join of 2 fragments if their fragment predicates are contradictory over the join attributes)

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

$$R \bowtie S \Leftrightarrow (R_1 \cup R_2) \bowtie (S_1 \cup S_2)$$
$$\Leftrightarrow (R_1 \bowtie S_1) \cup (R_1 \bowtie S_2) \cup (R_2 \bowtie S_1) \cup (R_2 \bowtie S_2)$$

- Then, join between 2 fragments can be simplified in some cases
  - Given $R_i = \sigma_{p_i}(R)$ and $R_j = \sigma_{p_j}(R)$ [p_i and p_j defined over join attributes]

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

Example - Reduction for PHF with join

Select * from EMP, ASG where EMP.eno=ASG.eno;

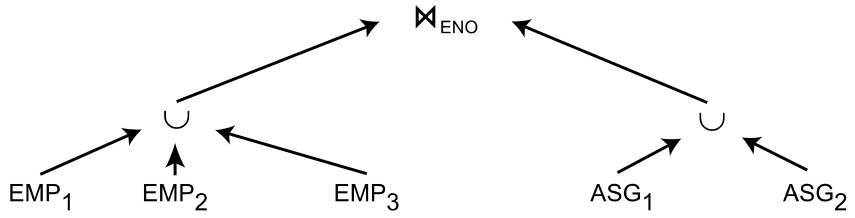
$ASG_1 = \sigma_{eno < \text{“E3”}}(ASG)$

$ASG_2 = \sigma_{eno > \text{“E3”}}(ASG)$

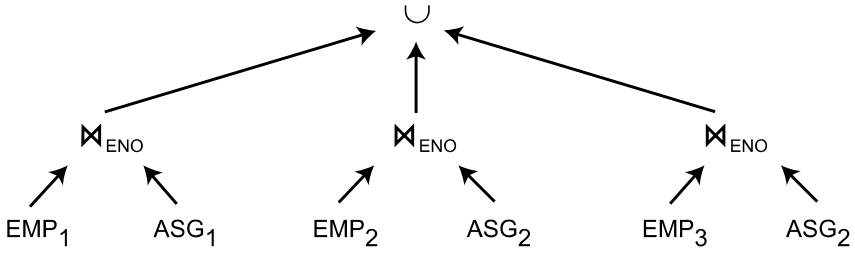
- EMP_1 and ASG_1 are defined by the same predicate.
- Furthermore, the predicate defining ASG_2 is the union of the predicates defining EMP_2 and EMP_3.

Example - Reduction for PHF with join

- The equivalent localized query is shown below.
- The query is reduced by **distributing joins over unions** and implemented as a union of three partial joins that can be done in parallel.



(a) Localized query



(b) Reduced query

Reduction for vertical fragmentation

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

  EMP(eno, ename, title)

  $EMP_1 = \Pi_{eno,\ ename}(EMP)$

  $EMP_2 = \Pi_{eno,\ title}(EMP)$

  Localization program is

  $EMP = EMP_1 \bowtie_{eno} EMP_2$

Reduction for vertical fragmentation

- Similar to horizontal 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) can produce useless, though not empty relations.

Example

Select ename from EMP;

- The equivalent localized query on EMP_1 and EMP_2 is shown below.
- By commuting the projection with the join (i.e., projecting on ENO, ENAME), we can see that the projection on EMP_2 is useless because ENAME is not in EMP_2. Therefore, the projection needs to apply only to EMP_1

(a) Localized query (b) Reduced query

Reduction for Derived Fragmentation

DHF: 2 relations R (member) and S (owner) in association one-to-many

- R participates with cardinality 1, S participates with cardinality N
- R can be fragmented following fragmentation on S
- Fragments that agree on the values of join attributes are placed at the same site
- Localization program: union

Rule :

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

Reduction for Derived Fragmentation

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

$$ASG_1 = ASG \ltimes_{eno} (EMP_1)$$
$$ASG_2 = ASG \ltimes_{eno} (EMP_2)$$

$$EMP_1 = \sigma_{title=\text{“programmer”}}(EMP)$$
$$EMP_2 = \sigma_{title\neq\text{“programmer”}}(EMP)$$

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

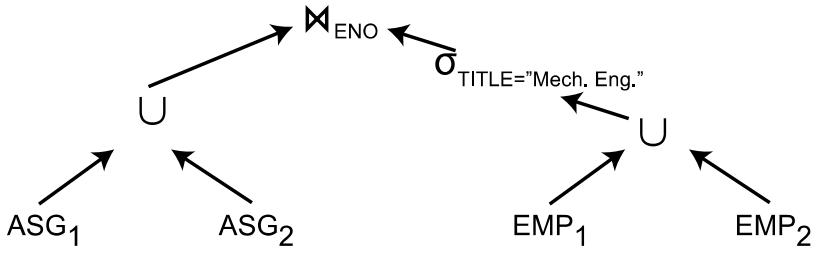
$$ASG = ASG_1 \cup ASG_2$$

Example - Reduction for Derived Fragmentation


```
Select *
From  ASG, EMP
WHERE ASG.eno=EMP.eno
AND title="MechEngr";
```

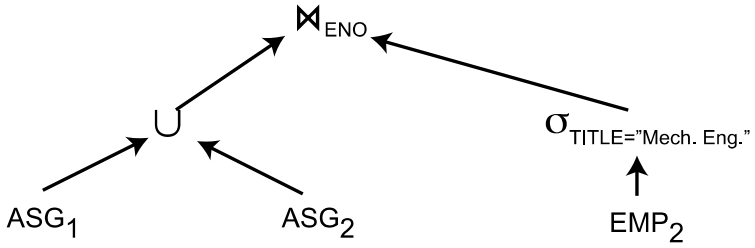
Example - Reduction for Derived Fragmentation

- The localized query on fragments EMP_1, EMP_2, ASG_1, and ASG_2



(a) Localized query

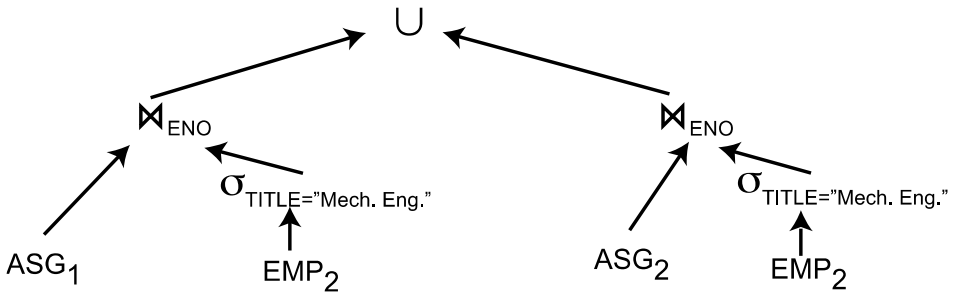
- By pushing selection down to fragments EMP_1 and EMP_2, the query reduces. This is because the selection predicate conflicts with that of EMP_1, and thus EMP_1 can be removed.



(b) Query after pushing selection down

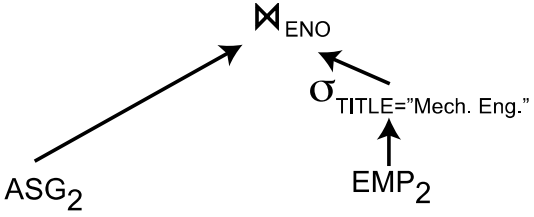
Example - Reduction for Derived Fragmentation

- In order to discover conflicting join predicates, distribute joins over unions.



(c) Query after moving unions up

- The left subtree joins two fragments, ASG_1 and EMP_2, whose qualifications conflict because of predicates TITLE = "Programmer" in ASG_1, and TITLE ≠"Programmer" in EMP_2. Therefore the left subtree which produces an empty relation can be removed, and the reduced query is obtained.



(d) Reduced query after eliminating the left subtree

Reduction for Hybrid Fragmentation

- The goal of hybrid fragmentation is to support, efficiently, queries involving projection, selection, and join. The localization program for a hybrid fragmented relation uses unions and joins of fragments.
- Here is an example of hybrid fragmentation of relation EMP:

$$\text{EMP}_1 = \sigma_{\text{ENO} \leq \text{"E4"}}(\Pi_{\text{ENO,ENAME}}(\text{EMP}))$$
$$\text{EMP}_2 = \sigma_{\text{ENO} > \text{"E4"}}(\Pi_{\text{ENO,ENAME}}(\text{EMP}))$$
$$\text{EMP}_3 = \Pi_{\text{ENO,TITLE}}(\text{EMP})$$

- The localization program is

$$\text{EMP} = (\text{EMP}_1 \cup \text{EMP}_2) \bowtie_{\text{ENO}} \text{EMP}_3$$

Example - Reduction for Hybrid Fragmentation

- Queries on hybrid fragments can be reduced by combining the rules used, respectively, in primary horizontal, vertical, and derived horizontal fragmentation.
- These rules can be summarized as follows:
  - 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.

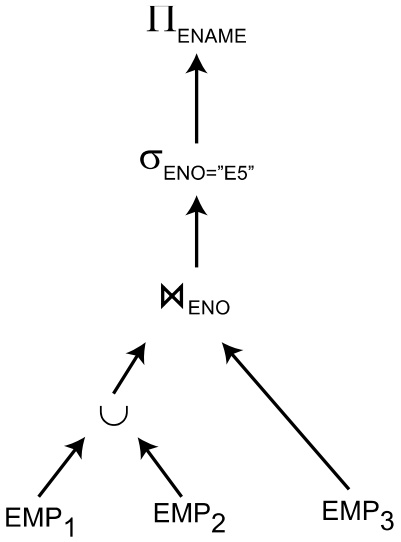
Example - Reduction for Hybrid Fragmentation

- Following example query in SQL illustrates the application of rules (1) and (2) to the horizontal-vertical fragmentation of relation EMP into EMP_1, EMP_2 and EMP_3 :

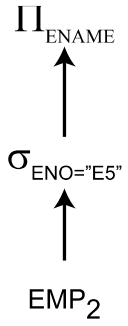
```
SELECT ENAME
FROM EMP
WHERE ENO="E5"
```

Example Reduction for Hybrid Fragmentation

- The localized query can be reduced by first pushing selection down, eliminating fragment EMP_1, and then pushing projection down, eliminating fragment EMP_3.



(a) Localized query



(b) Reduced query

Lecture Summary

innovate achieve lead

- Query Decomposition
- Data Localization

Thanks…

- Next Lecture

Query Optimization

- Questions??