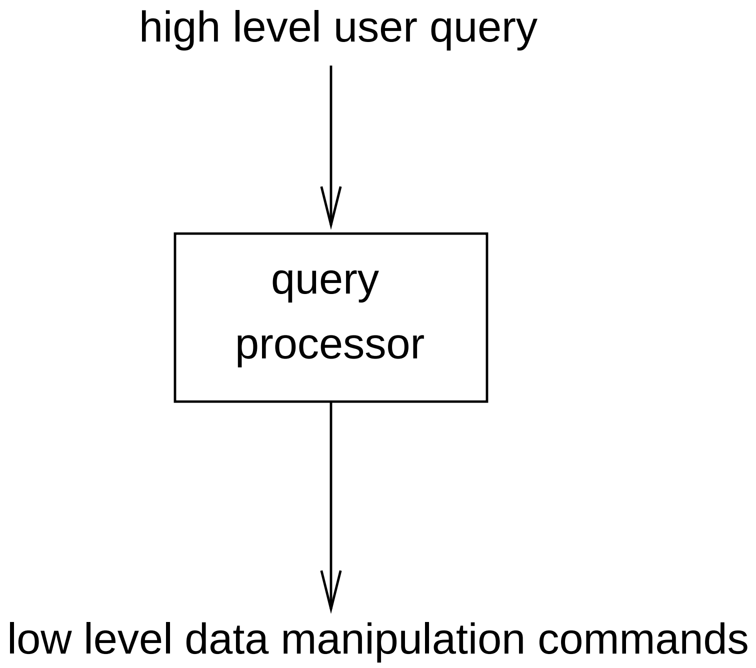
DDBMS QUERY PROCESSING



QUERY PROCESSING COMPONENTS

- query language
  - SQL: a standard
- query execution methodology
  - the steps that one goes through in executing high-level (declarative) user queries
- query optimization
  - determines the “best” execution plan

SELECTING ALTERNTIVES

Database Schema:

E(ENO, ENAME, TITLE)
G(ENO, JNO, RESP, DUR)

Query: Find the names of employees who are managing a project.

```
SELECT  ENAME
FROM    E, G
WHERE   E.ENO = G.ENO
AND     RESP = "Manager"
```

Strategy 1:

$$\pi_{ENAME}(\sigma_{RESP=\text{"}Manager\text{"} \wedge E.ENO=G.ENO}(E \times G))$$

Strategy 2:

$$\pi_{ENAME}(E \bowtie_{ENO} (\sigma_{RESP=\text{"}Manager\text{"}}(G)))$$

Strategy 2 avoids Cartesian product, so is “better”

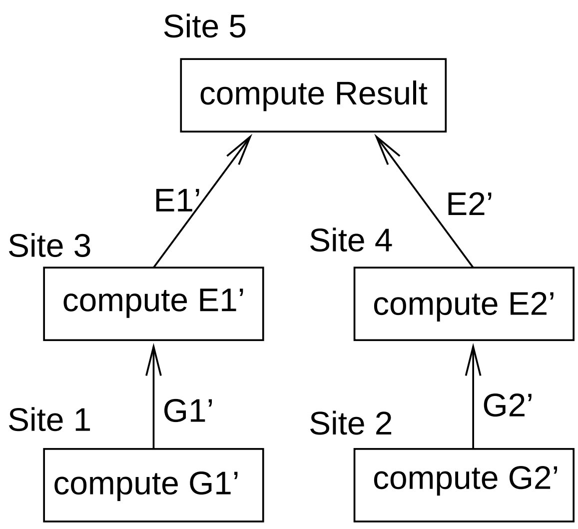
SELECTING ALTERNTIVES (CONT)

Suppose we have the following:

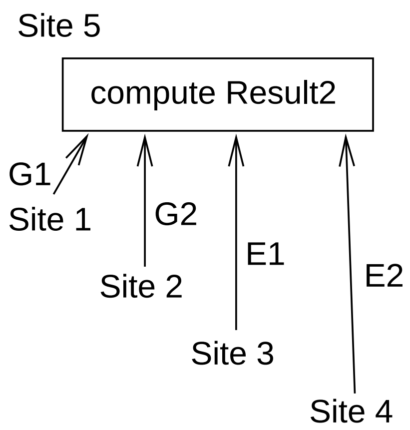
- site 1: $G_1 = \sigma_{ENO \leq e3}(G)$ site 2: $G_2 = \sigma_{ENO > e3}(G)$
- site 3: $E_1 = \sigma_{ENO \leq e3}(E)$ site 4: $E_2 = \sigma_{ENO > e3}(E)$
- site 5: RESULT

Let

- $G_1' = \sigma_{RESP="Manager"}(G_1)$
  $G_2' = \sigma_{RESP="Manager"}(G_2)$
- $E_1' = E_1 \bowtie_{ENO} G_1'$
  $E_2' = E_2 \bowtie_{ENO} G_2'$
- Result = $E_1' \cup E_2'$
- Result2 =
  $(E_1 \cup E_2) \bowtie_{ENO} (\sigma_{RESP="Manager"}(G_1 \cup G2))$



Strategy A



Strategy B

COST OF ALTERNATIVES

- Assume
  - size(E) = 400, size(G) = 1000
  - there are 20 managers in relation G
  - E and G are locally clustered on attributes ENO and RESP, resp.
  - data is uniformly distributed among sites
  - tuple access cost, AC = 1 unit; tuple transfer cost, TC = 10 units (Note: Tuple access cost actually depends on buffer and index)
- Strategy A

1.	produce G': $20 \times$ AC	20
2.	transfer G' to sites of E: $20 \times$ TC	200
3.	produce E': $(10 \times 10) \times$ AC $\times 2$	200
4.	transfer E' to site 5: $20 \times$ TC	200
	Total Cost	620

- Strategy B

1.	transfer E to site 5: $400 \times$ TC	4,000
2.	transfer G to site 5: $1000 \times$ TC	10,000
3.	produce G': $1000 \times$ AC	1,000
	Note: No index for E and G after transferring to site 5	
4.	join E and G': $400 \times 20 \times$ AC	8,000
	Total Cost	23,000

QUERY OPTIMIZATION OBJECTIVES

minimize a cost function

I/O cost + CPU cost + communication cost

These might have different weights in different distributed environments

- wide area networks
  - communication cost will dominate
    - low bandwidth
    - low speed
    - high protocol overhead
- local area networks
  - communication cost not that dominant
  - total cost function should be considered

can also maximize throughtput

COMPLEXITY OF RELATIONAL OPERATIONS

Assume relations of cadinality n and sequential scan

Operation	**Complexity**
Select Project (without duplicate elimination)	O(n)
Project (with duplicate elimination) Group By	O($n\ log\ n$)
Join Semijoin Division Set Operators	O($n\ log\ n$)
Cartesian Product	O(n^2)

Exercise: Find the exact formulas for the complexities of the relational operations.

QUERY OPTIMIZATION ISSUES

Types of optimizers

- exhaustive search
  - optimal
  - combinatorial complexity in the number of relations
- heuristics
  - not optimal
  - regroup common sub-expressions
  - perform selection, projection first
  - replace a join by a series of semijoins
  - reorder operations to reduce intermediate relation size
  - optimize individual operations

QUERY OPTIMIZATION ISSUES (CONT)

optimization granularity

- single query at a time
  - cannot use common intermediate results
- multiple queries at a time
  - efficient if many similar queries
  - decision space is much larger

QUERY OPTIMIZATION ISSUES (CONT)

optimization timing

- static
  - compilation – optimize prior to execution
  - difficult to estimate the size of the intermediate results – error propagation
  - can amortize over many executions
  - R*
- dynamic
  - run time optimization
  - exact information on the intermediate relation sizes
  - have to reoptimize for multiple executions
  - Distributed INGRES
- Hybrid
  - compile using a static algorithm
  - if the error in estimate sizes $>$ threshold, reoptimize at run time
  - MERMAID

QUERY OPTIMIZATION ISSUES (CONT)

Statistics

- relation
  - cardinality
  - size of a tuple
  - fraction of tuples participating in a join with another relation
- attribute
  - cardinality of domain
  - actual number of distinct values
- common assumptions
  - independence between different attribute values
  - uniform distribution of attribute values within their domain

QUERY OPTIMIZATION ISSUES (CONT)

decision sites

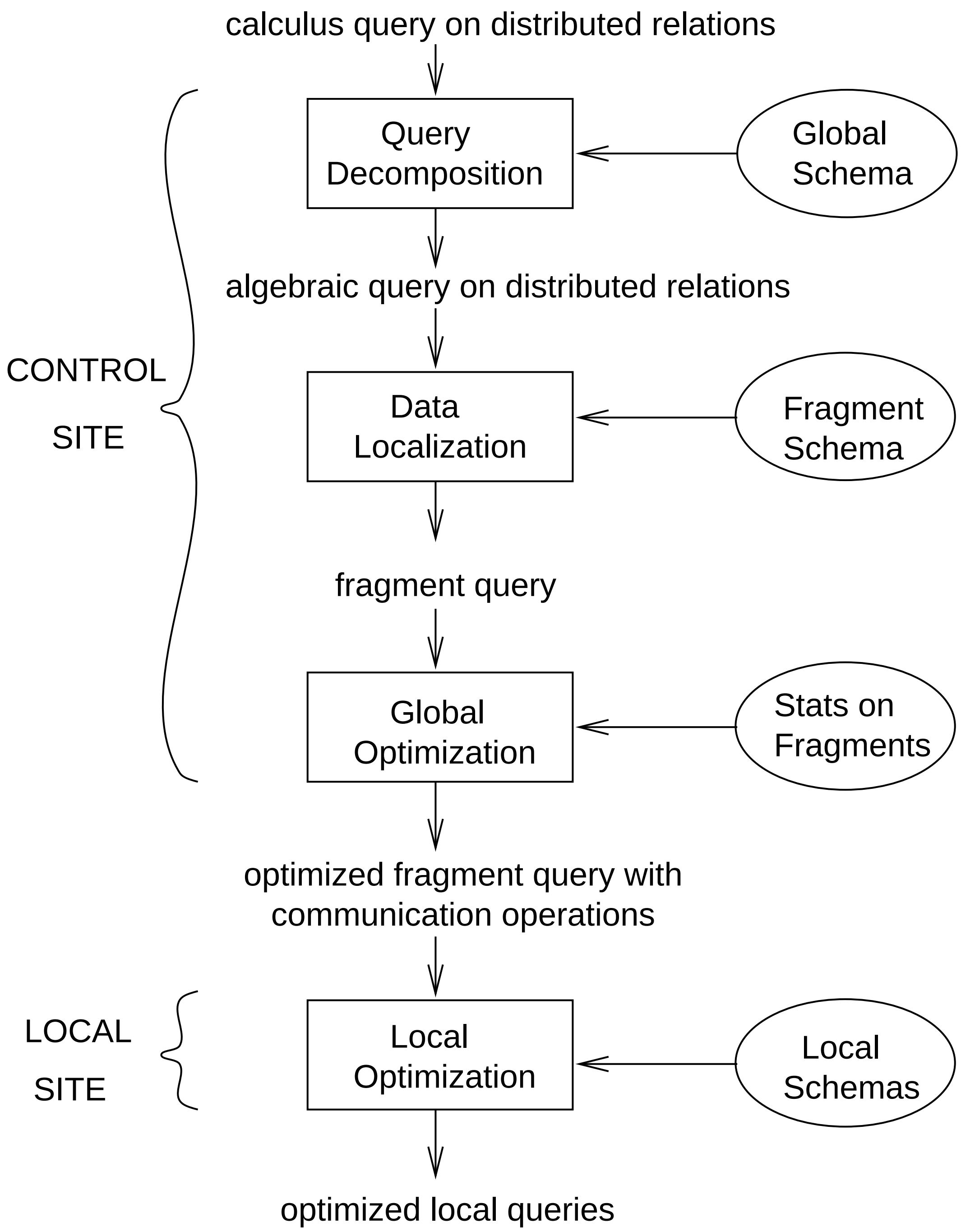
- centralized
  - single site determines the ‘best’ schedule
  - simple
  - need knowledge about the entire distributed database
- distributed
  - cooperation among sites to determine the schedule
  - need only local information
  - cost of cooperation
- hybrid
  - one site determines the global schedule
  - each site optimizes the local subqueries

QUERY OPTIMIZATION ISSUES (CONT)

network topology

- wide area networks (WAN) – point-to-point
  - characteristics
    - low bandwidth
    - low speed
    - high protocol overhead
  - communication cost will dominate; ignore all other cost factors
  - global schedule to minimize communication cost
  - local schedules according to centralized query optimization
- local area networks (LAN)
  - communication cost not that dominant
  - total cost function should be considered
  - broadcasting can be exploited (joins)
  - special algorithms exist for star networks

DISTRIBUTED QUERY PROCESSING METHODOLOGY



STEP 1: QUERY DECOMPOSITION

Input: 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
  - transform calculus query to algebraic query
  - more than one translation is possible
  - use transformation rules

NORMALIZATION

- lexical and syntactic analysis
  - check validity (similar to compilers)
  - check for attributes and relations
  - type checking on the qualification
- put into normal form
  - conjunctive normal form
    $(p_{11} \vee p_{12} \vee \ldots p_{1n}) \wedge \ldots \wedge (p_{m1} \vee \ldots p_{mn})$
  - disjunctive normal form
    $(p_{11} \wedge p_{12} \wedge \ldots p_{1n}) \vee \ldots \vee (p_{m1} \wedge \ldots p_{mn})$
  - OR is mapped into union
  - AND is mapped into join or selection

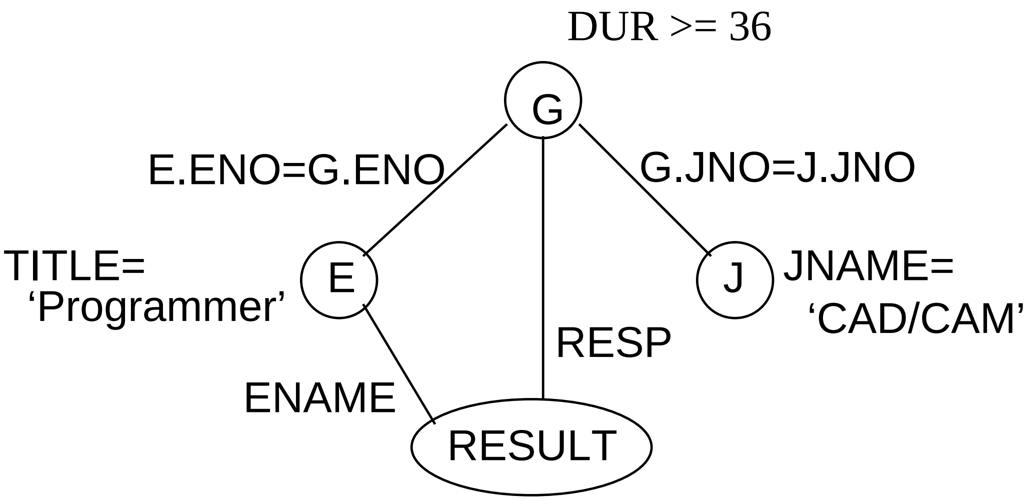
ANALYSIS

- refute 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
- semantically incorrect
  - components do not contribute in any way to the generation of the result
  - only a subset of relational calculus queries can be tested for correctness
  - those that do not contain disjunction and negation
  - to detect
    - connection graph (query graph)
    - join graph

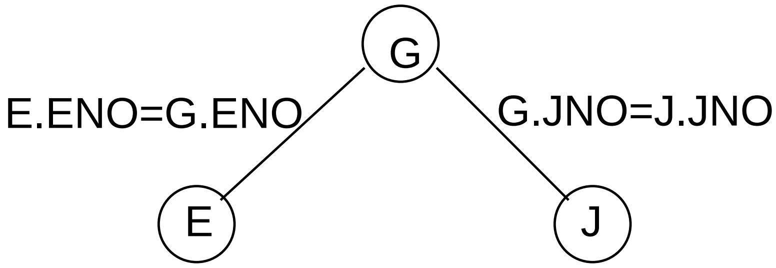
ANALYSIS – EXAMPLE

SELECT	ENAME, RESP
FROM	E, G, J
WHERE	E.ENO = G.ENO
AND	G.JNO = J.JNO
AND	JNAME = ‘CAD/CAM’
AND	DUR $\geq$ 36
AND	TITLE = ‘Programmer’

Query Graph



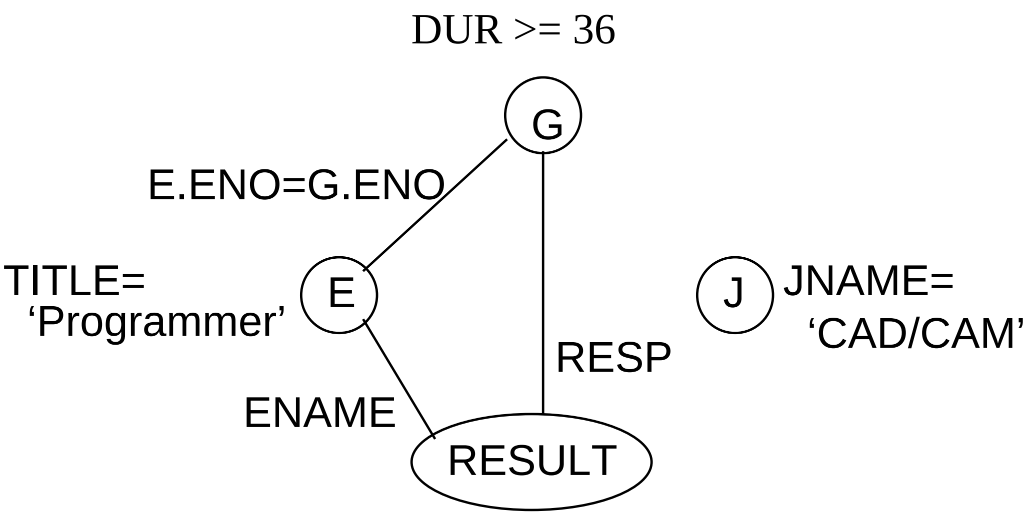
Join Graph



ANALYSIS – EXAMPLE

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

SELECT	ENAME, RESP
FROM	E, G, J
WHERE	E.ENO = G.ENO
AND	JNAME = ‘CAD/CAM’
AND	DUR $\geq$ 36
AND	TITLE = ‘Programmer’



SIMPLIFICATION

- why simplify?
  - remember the example
- how? use transformation rules
  - elimination of redundancy
    - idempotency rules

      $p_1 \wedge \neg(p_1) \leftrightarrow false$

      $p_1 \wedge (p_1 \vee p_2) \leftrightarrow p_1$

      $p_1 \vee false \leftrightarrow p_1$
  - application of transitivity
  - use of integrity rules

SIMPLIFICATION – EXAMPLE

SELECT	TITLE
FROM	E
WHERE	E.ENAME = 'J.DOE'
OR	(**NOT**(E.TITLE = 'Programmer')
AND	(E.TITLE = 'Programmer'
OR	E.TITLE = 'Elect. Eng.')
AND	**NOT**(E.TITLE = 'Elect. Eng.'))

becomes

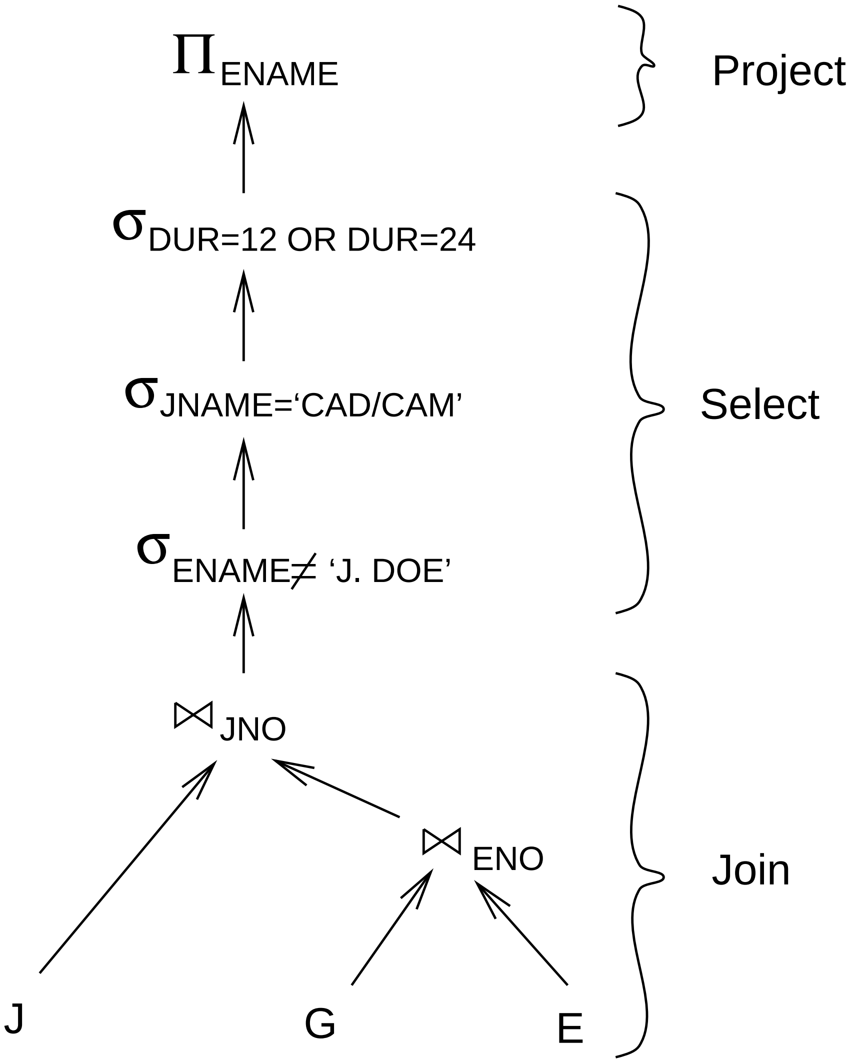
SELECT	TITLE
FROM	E
WHERE	E.ENAME = 'J.DOE'

RESTRUCTURING

- convert relational calculus to relational algebra
- make use of query trees
- Example

SELECT	ENAME
FROM	E, G, J
WHERE	E.ENO = G.ENO
AND	G.JNO = J.JNO
AND	ENAME $\neq$ ‘J.DOE’
AND	JNAME = ‘CAD/CAM’
AND	(DUR = 12 **OR** DUR = 24)

RESTRUCTURING (CONT)



RESTRUCTURING – TRANSFORMATION RULES

- commutativity of binary operations
  - $R \times S \leftrightarrow S \times R$
  - $R \bowtie S \leftrightarrow S \bowtie R$
  - $R \cup S \leftrightarrow S \cup R$
- associativity of binary operations
  - $(R \times S) \times T \leftrightarrow R \times (S \times T)$
  - $(R \bowtie S) \bowtie T \leftrightarrow R \bowtie (S \bowtie T)$
- idempotence of unary unary operations
  - $\pi_{A'}(\pi_{A'}(R)) \leftrightarrow \pi_{A'}(R)$
  - $\sigma_{p1(A1)}(\sigma_{p2(A2)}(R)) \leftrightarrow \sigma_{p1(A1) \wedge p2(A2)}(R)$
- commuting selection with projection

RESTRUCTURING – TRANSFORMATION RULES (CONT)

- commuting selection with binary operations
  - $\sigma_{p(A)}(R \times S) \leftrightarrow (\sigma_{p(A)}(R)) \times S$

    where A is in R only
  - $\sigma_{p(A_i)}(R \bowtie_{(A_j,B_k)} S) \leftrightarrow (\sigma_{p(A_i)}(R)) \bowtie_{(A_j,B_k)} S$

    where A_i is in R only
  - $\sigma_{p(A_i)}(R \cup S) \leftrightarrow \sigma_{p(A_i)}(R) \cup \sigma_{p(A_i)}(S)$

    where A_i is in $R \cup S$
- commuting projection with binary operations
  - $\pi_C(R \times S) \leftrightarrow \pi_{A'}(R) \times \pi_{B'}(S)$

    where $C = A' \cup B'$ and A' is in R only and B' is in S only
  - $\pi_C(R \bowtie_{(A_j,B_k)} S) \leftrightarrow \pi_{A'}(R) \bowtie_{(A_j,B_k)} \pi_{B'}(S)$

    where $C = A' \cup B'$ and A' is in R only and B' is in S only
  - $\pi_C(R \cup S) \leftrightarrow \pi_C(R) \cup \pi_C(S)$

    where C is in $R \cup S$

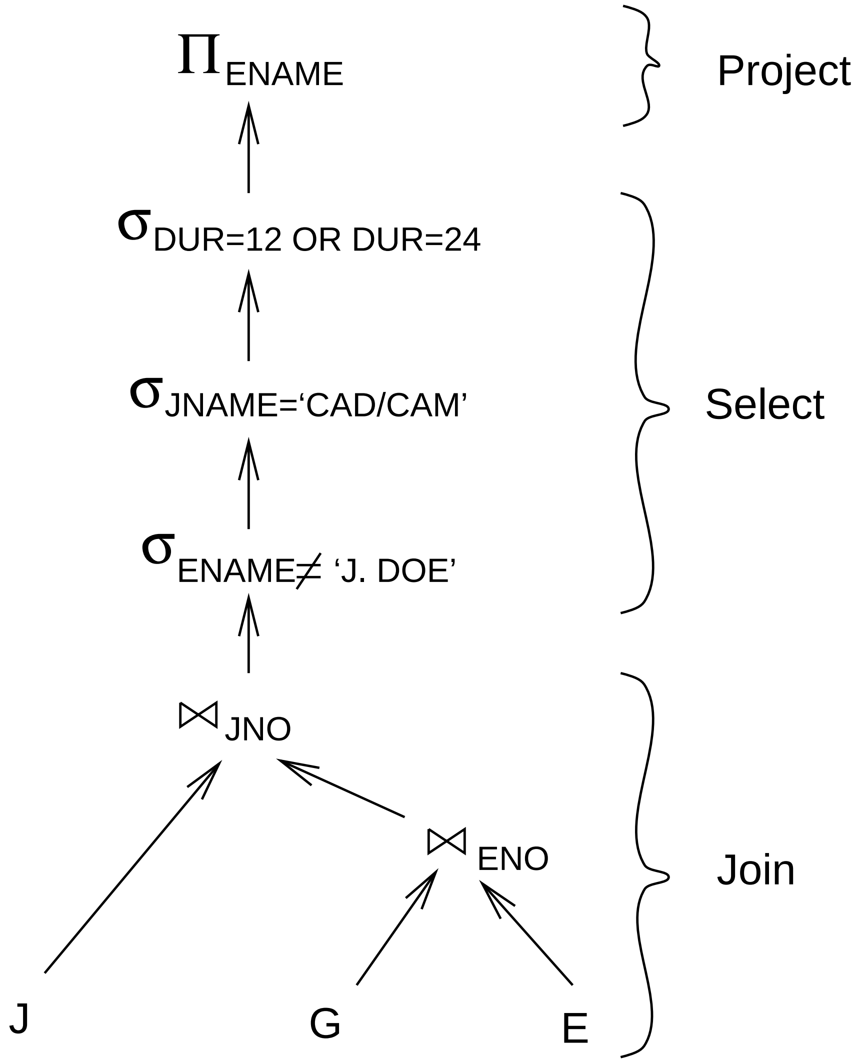
EXAMPLE

Recall the previous example:

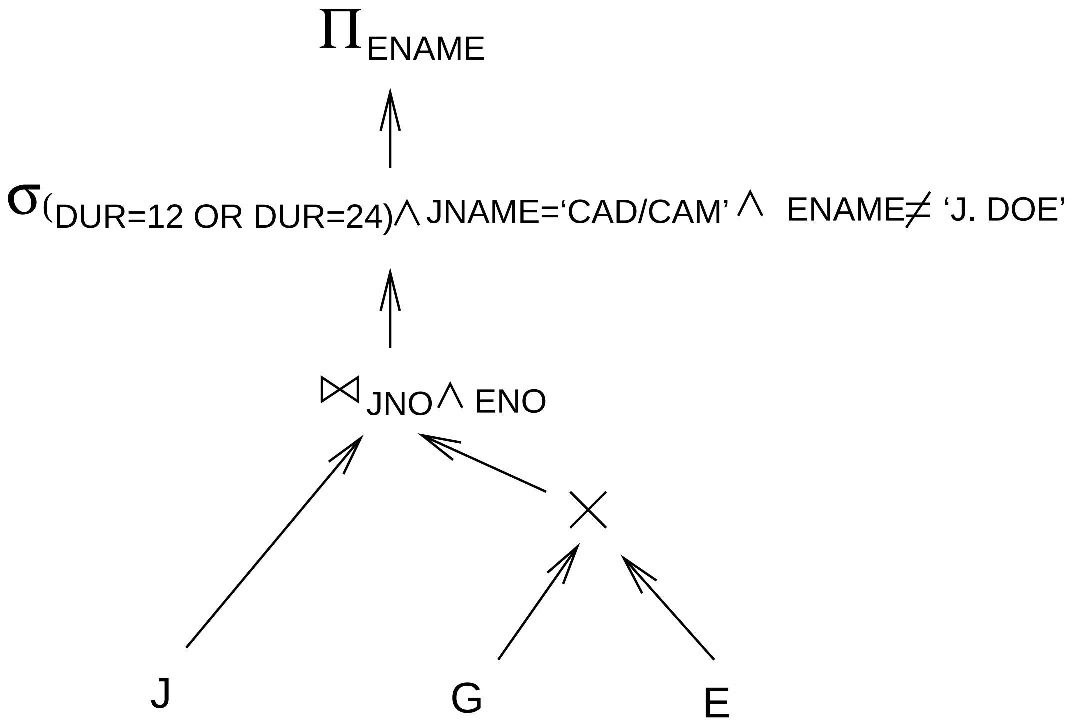
- Database schema
  E(ENO, ENAME, TITLE)
  G(ENO, JNO, RESP, DUR)
  J(JNO, JNAME, BUDGET, LOC)
- Find the names of employees other than J. Doe who worked on the CAD/CAM project for either one or two years

SELECT	ENAME
FROM	E, G, J
WHERE	E.ENO = G.ENO
AND	G.JNO = J.JNO
AND	ENAME $\neq$ 'J.DOE'
AND	JNAME = 'CAD/CAM'
AND	(DUR = 12 **OR** DUR = 24)

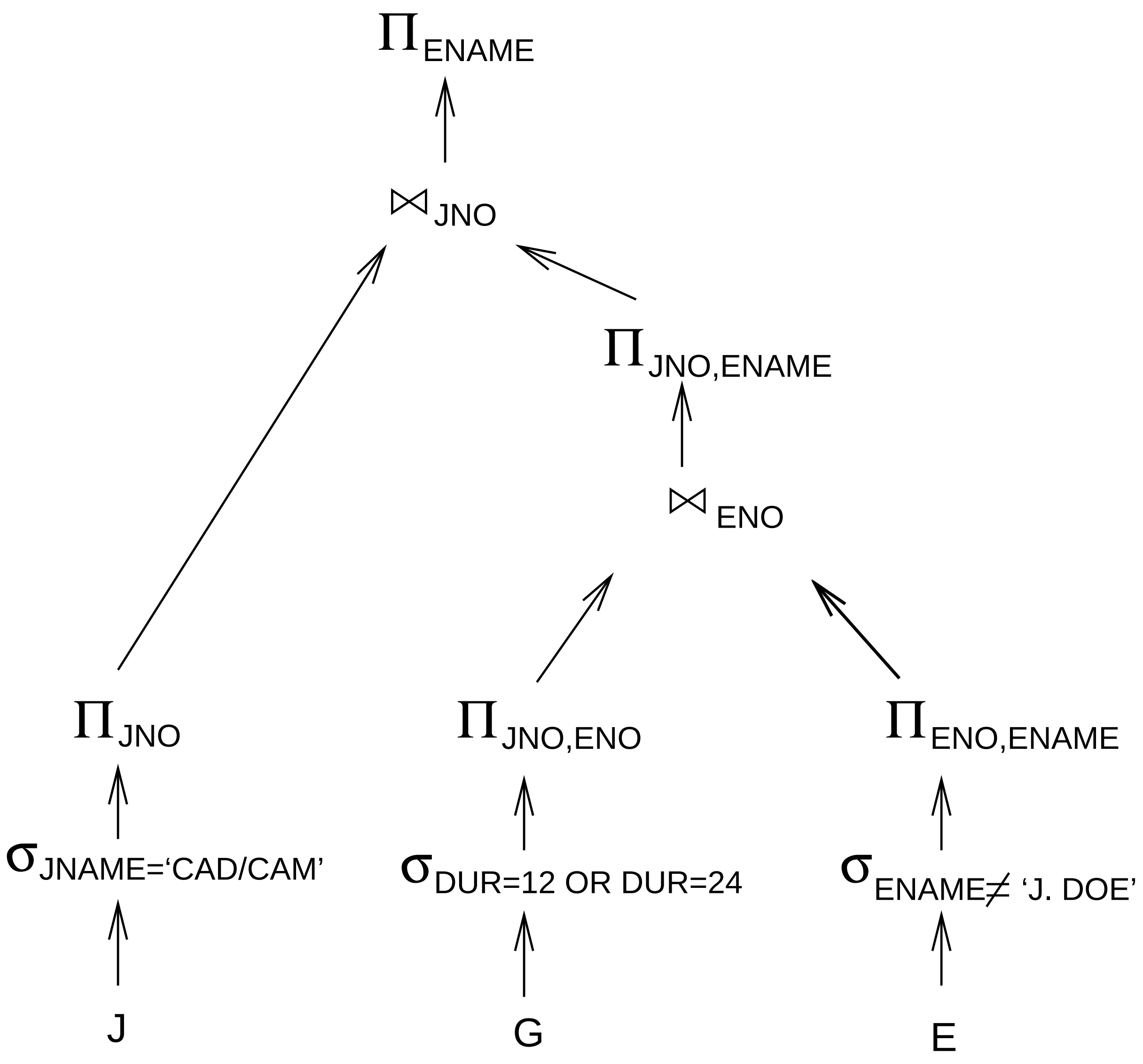
EXAMPLE (CONT)



EQUIVALENT QUERY



RESTRUCTURING



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

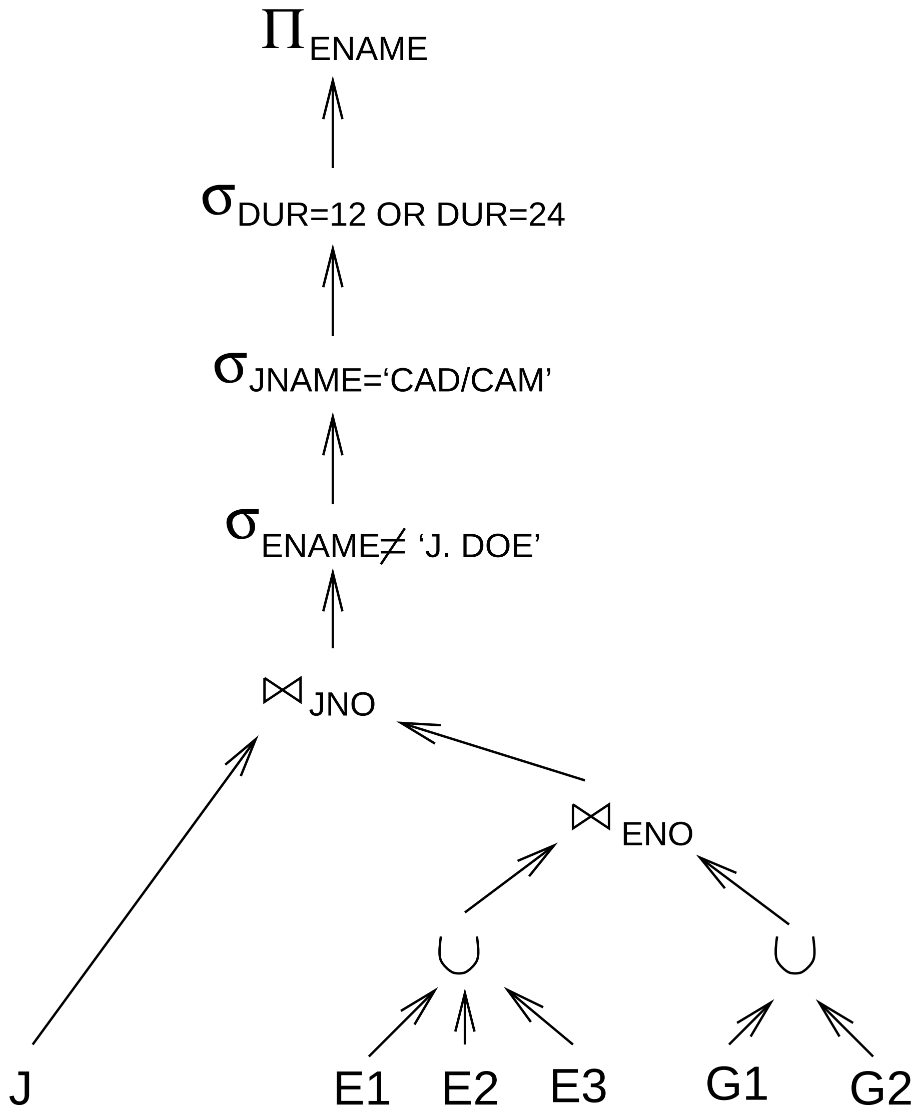
EXAMPLE

Assume

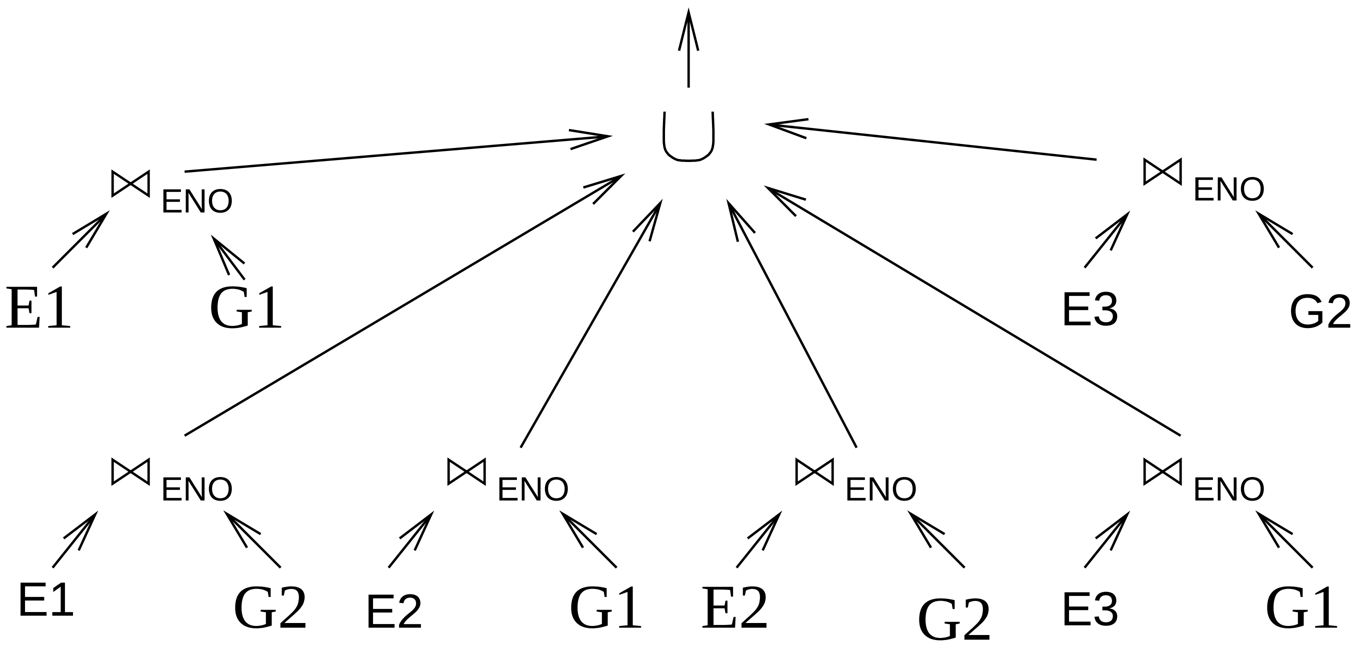
- E is fragmented into E1, E2 and E3 as follows:
  - $E_1 = \sigma_{ENO \leq e3}(E)$
  - $E_2 = \sigma_{e3 < ENO \leq e6}(E)$
  - $E_3 = \sigma_{ENO > e6}(E)$
- G is fragmented into G_1 and G_2 as follows:
  - $G_1 = \sigma_{ENO \leq e3}(G)$
  - $G_2 = \sigma_{ENO > e3}(G)$

Replace E by ($E_1 \cup E_2 \cup E_3$) and
G by ($G_1 \cup G_2$)

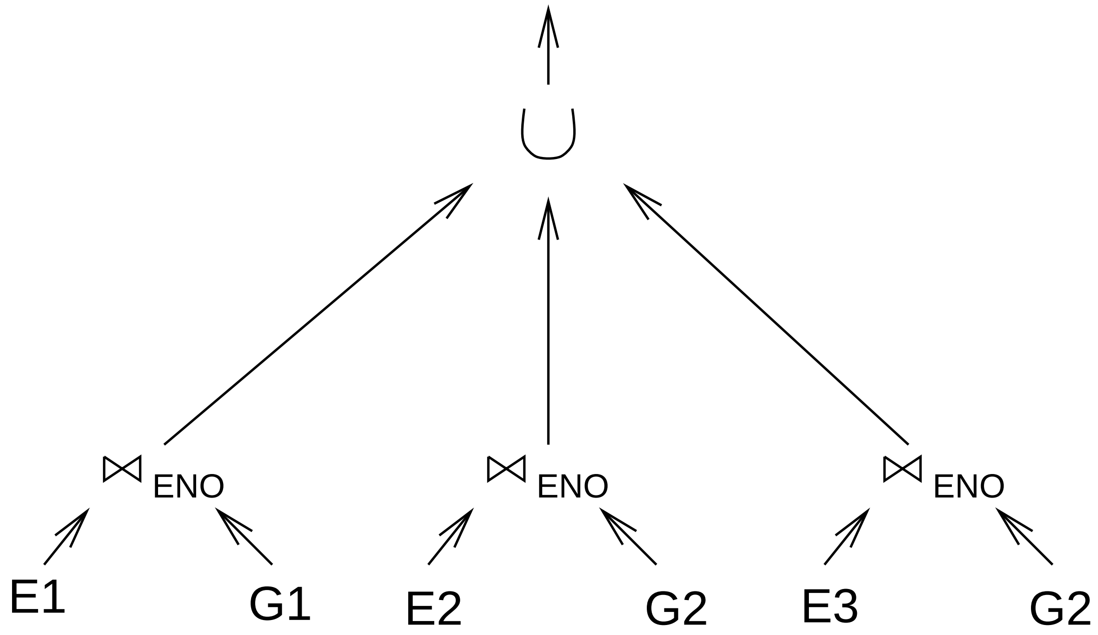
EXAMPLE (CONT)



PROVIDES PARALLELISM



ELIMINATES UNNECESSARY WORK



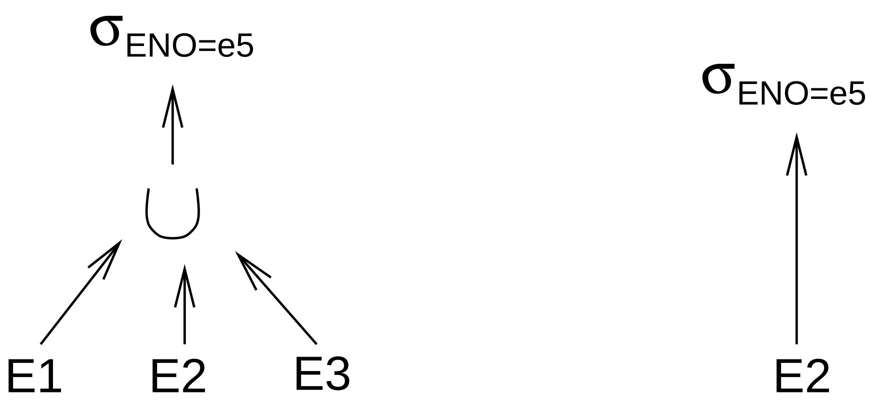
REDUCTION FOR PHF

reduction with selection

- Relation R and $F_R = \{R_1, R_2, \ldots, R_w\}$ where
  $R_j = \sigma_{p_j}(R)$,
  $\sigma_{p_i}(R_j) = \emptyset$ if $\forall x \in R : \neg(p_i(x) \wedge p_j(x))$

- Example:

SELECT	*
FROM	E
WHERE	ENO = e5



REDUCTION FOR PHF (CONT)

reduction with join

- possible if fragmentation is done on join attribute
- distribute join over union
- given $R_i = \sigma_{p_i}(R)$ and $R_j = \sigma_{p_j}(R)$
  $R_i \bowtie R_j = \emptyset$
  if $\forall x \in R_i, \forall y \in R_j : \neg(p_i(x) \wedge p_j(y))$

EXAMPLE

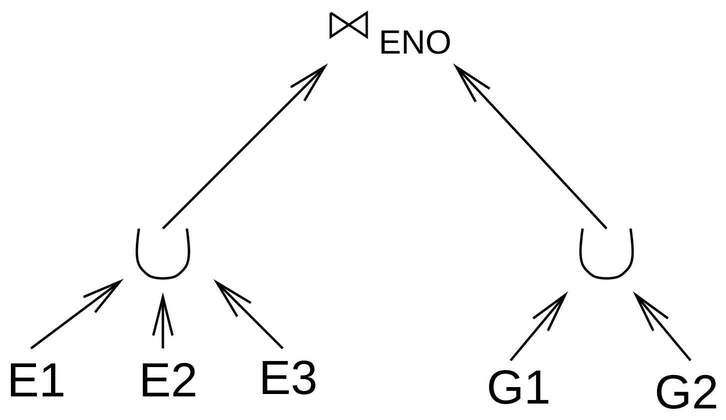
- assume E is fragmented as before and

  $G_1 : \sigma_{ENO \leq e3}(G)$

  $G_2 : \sigma_{ENO > e3}(G)$

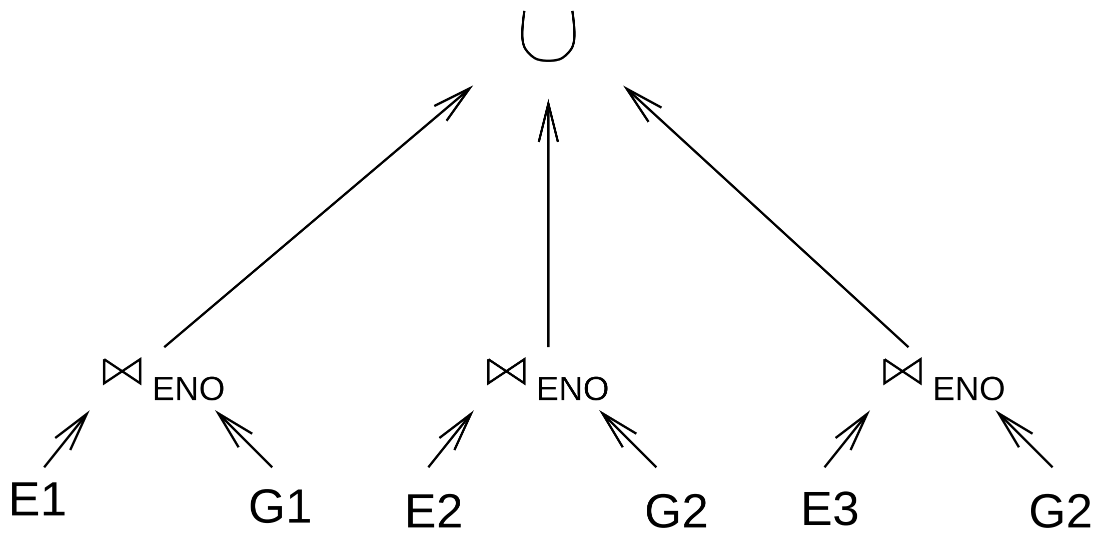
- consider the query

SELECT *
FROM E, G
WHERE E.ENO = G.ENO



EXAMPLE (CONT)

- distribute join over unions
- apply the reduction rule



REDUCTION FOR VF

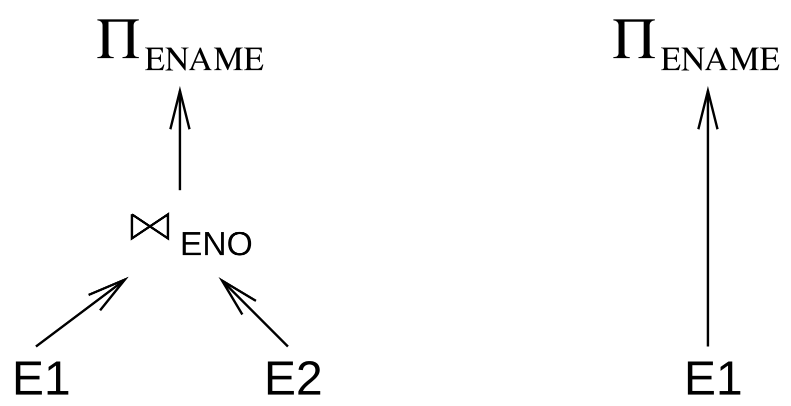
find useless (not empty) intermediate relations

- relation R defined over attributes $A = \{A_1, \ldots, A_n\}$ vertically fragmented as $R_i = \pi_{A'}(R)$ where $A' \subset A$.

  $\pi_{D,K}(R_i)$ is useless if the set of projection attributes D is not in A' where K is the primary key of R.

- Example: $E_1 = \pi_{ENO,ENAME}(E)$
  $E_2 = \pi_{ENO,TITLE}(E)$

SELECT ENAME
FROM E



REDUCTION FOR DHF

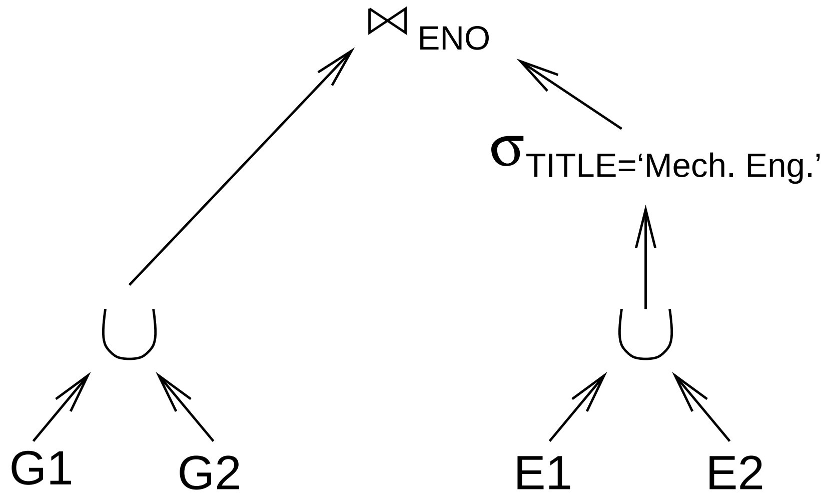
- rule:
  - distribute joins over unions
  - apply the join reduction for horizontal fragmentation
- Example:

$$G_1 : G \ltimes_{ENO} E_1$$
$$G_2 : G \ltimes_{ENO} E_2$$
$$E_1 : \sigma_{TITLE=\text{`}Programmer'}(E)$$
$$E_2 : \sigma_{TITLE\neq\text{`}Programmer'}(E)$$

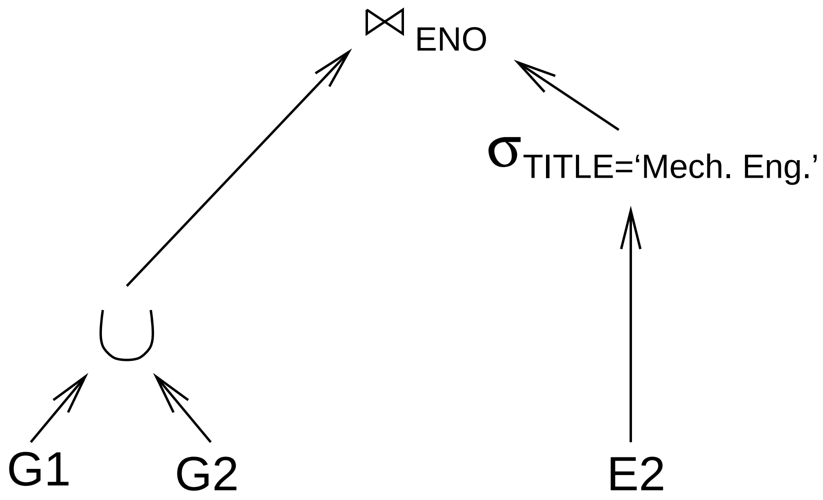
SELECT	*
FROM	E, G
WHERE	E.ENO = G.ENO
AND	E.TITLE = ‘Mech. Eng.’

EXAMPLE

Generic Query

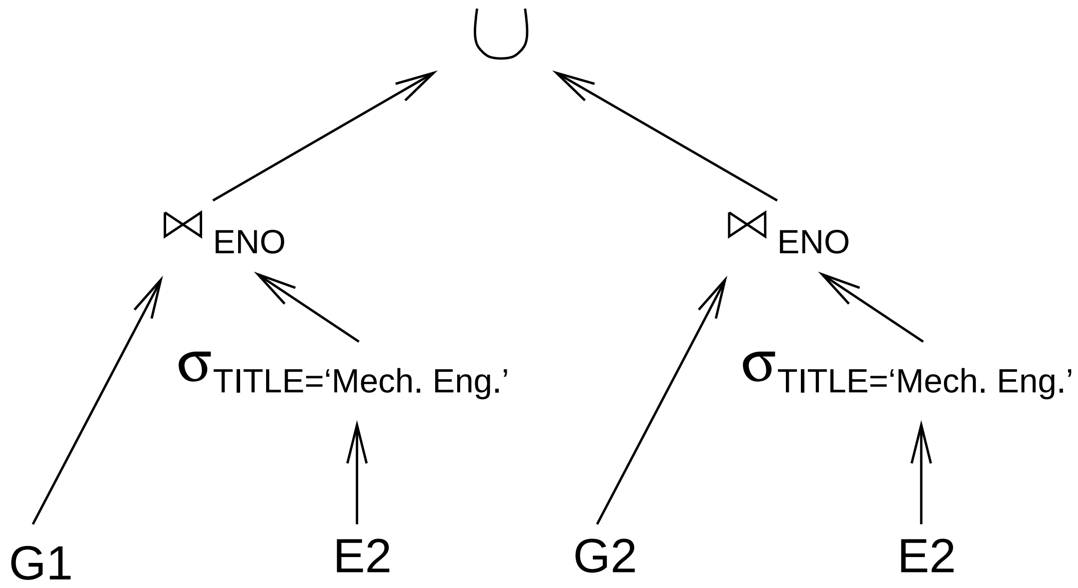


Selections first

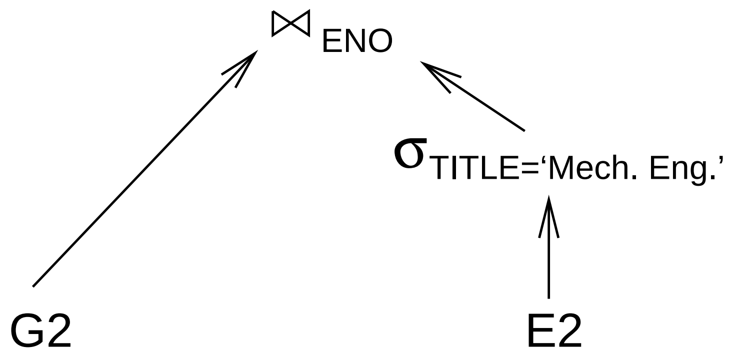


EXAMPLE (CONT)

Joins over unions



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



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

Consider the following hybrid fragmentation:

$E_1 = \sigma_{ENO \leq e4}(\pi_{ENO,ENAME}(E))$
$E_2 = \sigma_{ENO > e4}(\pi_{ENO,ENAME}(E))$
$E_3 = \pi_{ENO,TITLE}(E)$

and the query

SELECT	ENAME
FROM	E
WHERE	ENO = e5