CS54200: Distributed Database Systems

Query Processing

9 March 2009

Prof. Chris Clifton

Introduction

- Query Processing
  - Converting user commands from the query language (SQL) to low level data manipulation commands.
  - SQL is declarative – it describes the properties of the result, not the operations to produce it.
- Query Optimization
  - Determining the "best" or a good execution plan for the query.

 2

Selecting Alternatives

SELECT ENAME
FROM EMP, ASG
WHERE EMP.ENO = ASG.ENO
AND DUR > 37.

Strategy 1:

$$\prod_{ENAME}(\sigma_{EMP.ENO=ASG.ENO \wedge DUR>37)}(EMP \times ASG))$$

Strategy 2:

$$\prod_{ENAME}(EMP \rhd\lhd_{ENO} (\sigma_{DUR>37)}(ASG)))$$

Strategy 2, avoids cartesian product.

(c)Oszu & Valduriez 3

Problem

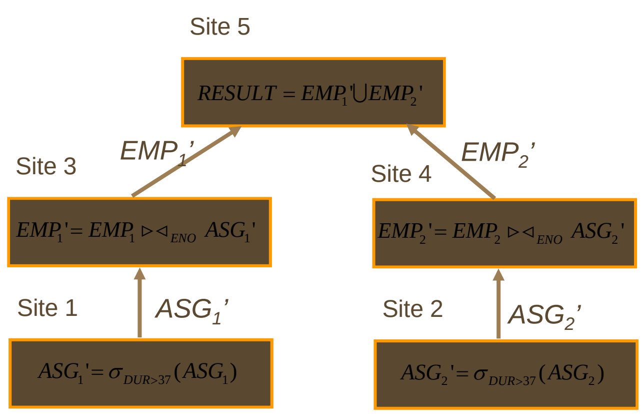
Site1: $ASG_1 = \sigma_{ENO \leq "E3"}(ASG)$

Site2: $ASG2 = \sigma_{ENO > "E3"}(ASG)$

Site3: $EMP_1 = \sigma_{ENO \leq "E3"}(EMP)$

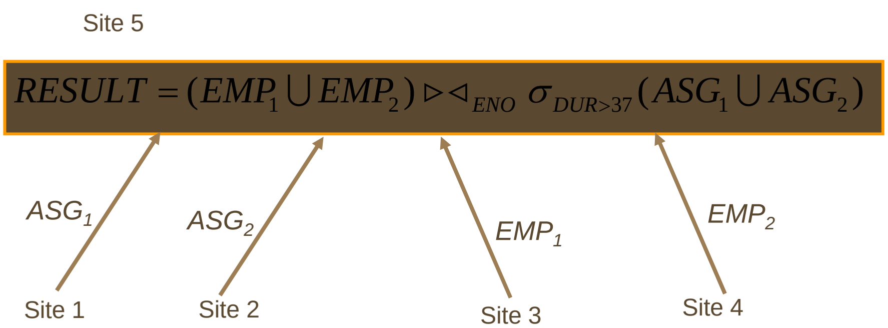
Site4: $EMP_2 = \sigma_{ENO > "E3"}(EMP)$

Site5: $RESULT$



(c)Oszu & Valduriez 4

Alternative 2



(c)Oszu & Valduriez 5

Cost of Alternatives

- Assume
  - Size(EMP) = 400; size(ASG)=1000
  - Tuple access cost (TAC) = 1unit; tuple xfer cost (TXC) =10units
- Strategy 1
  - Produce ASG’: (10+10)*TAC = 20
  - Transfer ASG’: (10+10)*TXC = 200
  - Produce EMP’: (10+10)*TAC*2 = 40
  - Transfer EMP’ to result site: (10+10)*TXC = 200
  - Total COST = 460.

(c)Oszu & Valduriez 6

Cost of alternatives (cont)

- Strategy 2
  - Transfer EMP to site 5: 400*TXC = 4000
  - Transfer ASG to site 5: 1000*TXC = 10,000
  - Produce ASG': 1000*TAC = 1,000
  - Join EMP and ASG': 400*20*TAC = 8,000

  - TOTAL COST = 23,000!!

(c)Oszu & Valduriez 7

Query Optimization Objectives

- Minimize a cost function
  - I/O cost + CPU cost + communication cost
- These may have different weights in different distributed environments
- Wide area networks
  - Communication cost will dominate
    - Low bandwidth
    - Low speed
    - High protocol overhead
  - Most algorithms ignore all other cost components

(c)Oszu & Valduriez 8

Query Optimization Objectives

- Local area networks
  - Communication cost not that dominant
  - Total cost function should be considered
- Can also maximize throughput.

(c)Oszu & Valduriez 9

PURDUE UNIVERSITY

CS54200: Distributed Database Systems

Query Processing

13 March 2009

Prof. Chris Clifton

Indiana Center for Database Systems

Complexity of Relational Operators

Assume
Relations of cardinality *n*
Sequential scan

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

Issues: Types of Optimizers

- Exhaustive Search
  - Cost-based
  - Optimal
  - Combinatorial complexity in # 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

Issues: 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

 13

Issues: 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 reln. Sizes
  - Have to reoptimize for multiple executions
  - Distributed INGRES

 14

Issues: Optimization Timing

- Hybrid:
  - Compile a static algorithm
  - If the error in estimate sizes > threshold, reoptimize at runtime
  - MERMAID

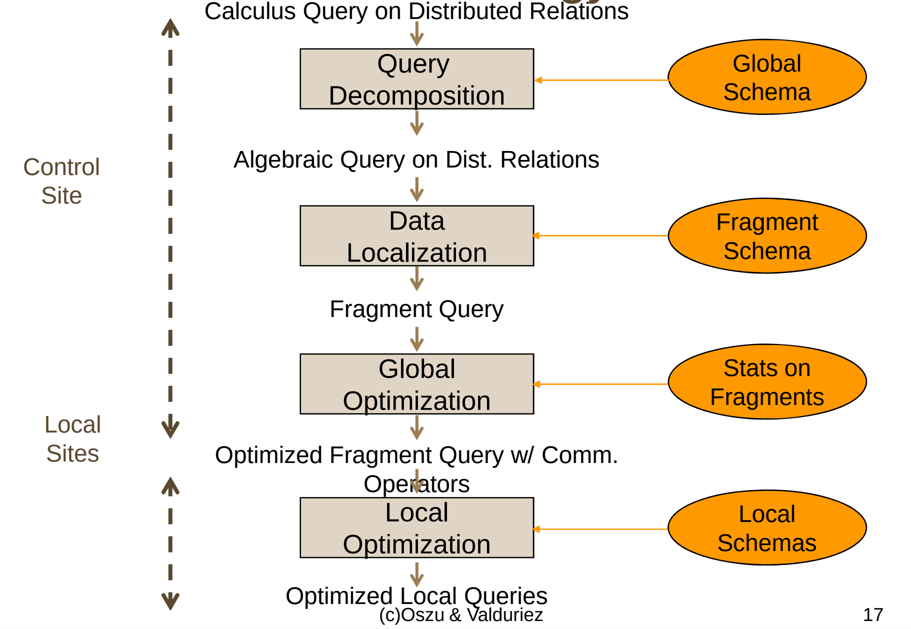
(c)Oszu & Valduriez 15

Issues: Statistics

- Relation
  - Cardinality
  - Size of a tuple
  - Fraction of tuples participating in a join
- Attribute
  - Cardinality of domain
  - Actual number of distinct values
- Common assumptions
  - Independence between different attribute values
  - Uniform distribution of attribute values within their domain

(c)Oszu & Valduriez 16

Methodology



(c)Oszu & Valduriez 17

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 or reln. Calculus
- Simplification
  - Eliminate redundant predicates
- Restructuring
  - Calculus query → algebra query
  - More than one translation is possible
  - Use transformation rules.

(c)Oszu & Valduriez 18

Normalization

- Lexical and syntactic analysis
  - Check validity (similar to compilers)
  - Check for attributes and relations
  - Type checking on quantification
- Put into normal form
  - Conjunctive normal form
  - Disjunctive normal form
  - ORs mapped into union
  - ANDs mapped into join or selection

 19

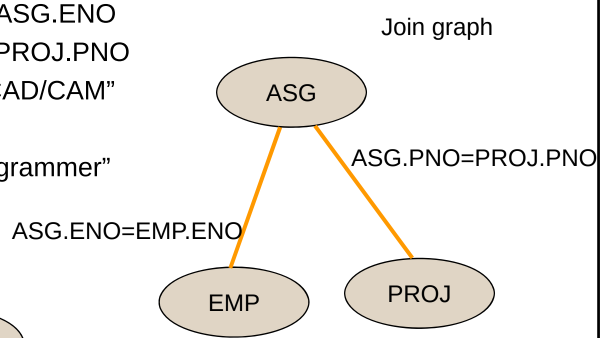
Analysis

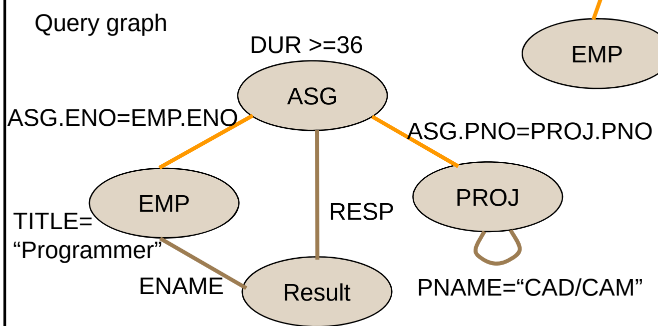
- Refute incorrect queries
- Type incorrect
  - If any of its attribute or relation names are not defined in the global schema
  - If operations are applied to attributes of the wrong type
- Semantically incorrect
  - Components do not contribute to result
  - Only a subset of reln. Calculus can be tested for correctness
  - Those that do not contain disjunction and negation
  - To detect
    - Connection graph (query graph)
    - Join graph

 20

Analysis Example

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



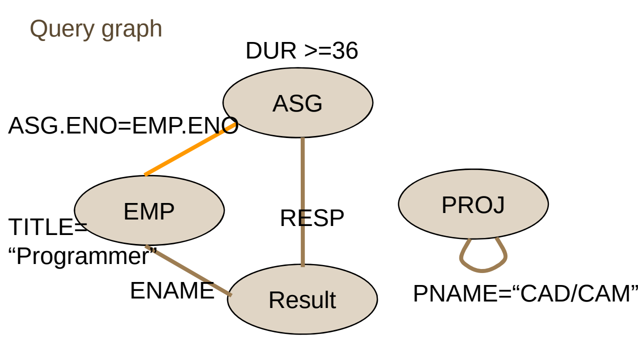


(c)Oszu & Valduriez 21

Analysis Example

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

If the query graph is not connected, the query is incorrect.



(c)Oszu & Valduriez 22

Simplification

- Why simplifiy?
  - Remember the example
- How? Use transformation rules
  - Elimination of redundancy
    - Idempotency rules
  - Application of transitivity
  - Use of integrity rules

(c)Oszu & Valduriez 23

Simplification Example

```
SELECT   TITLE
FROM     EMP
WHERE    EMP.ENAME = “J. DOE”
AND      (NOT(EMP.TITLE=“Programmer”)
AND      (EMP.TITLE=“Programmer”
OR       EMP.TITLE=“Elect. Engg.”)
AND      NOT(EMP.TITLE=“Elect. Engg.”))
```

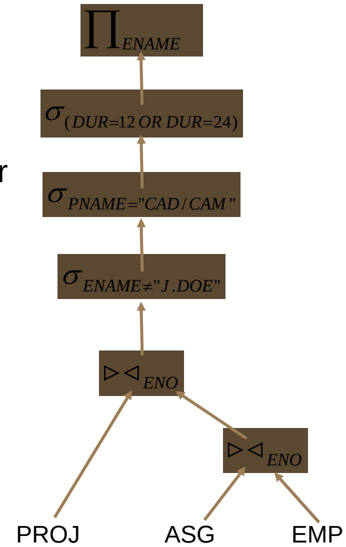
```
SELECT   TITLE
FROM     EMP
WHERE    EMP.ENAME = “J. DOE”
```

(c)Oszu & Valduriez 24

Restructuring

- Convert calculus to algebra
- Make use of query trees
- Example
  - Find names of employees other than J. Doe who worked on the CAD/CAM project for 1 or 2 years.

SELECT ENAME
FROM EMP, ASG, PROJ
WHERE EMP.ENO = ASG.ENO
AND ASG.PNO = PROJ.PNO
AND EMP.ENAME<>"J. DOE"
AND PNAME = "CAD/CAM"
AND **(**DUR =12 **OR** DUR = 24)



(c)Oszu & Valduriez 26

Transformation Rules

- Commutativity of binary operators
  - $R \times S \Leftrightarrow S \times R$
  - $R \bowtie S \Leftrightarrow S \bowtie R$
  - $R \cup S \Leftrightarrow S \cup R$
- Associativity of binary operators
  - $(R \times S) \times T \Leftrightarrow R \times (S \times T)$
  - $(R \bowtie S) \bowtie T \Leftrightarrow R \bowtie (S \bowtie T)$
- Idempotence of Unary operators
  - $(\prod_{A'}(\prod_{A'}(R)) \Leftrightarrow \prod_{A'}(R)$
  - $(\sigma_{p1(A1)}(\sigma_{p2(A2)}(R)) \Leftrightarrow \sigma_{p1(A1) \wedge p2(A2)}(R)$
  - Where *R[A]* and $A' \subseteq A$
- Commuting selection with projection

(c)Oszu & Valduriez 27

Transformation Rules

- Commuting selection with binary operators

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

$$\sigma_{p(Ai)}(R \bowtie_{Aj,Bk} S) \Leftrightarrow (\sigma_{p(Ai)}(R)) \bowtie_{Aj,Bk} S$$

$$\sigma_{p(Ai)}(R \cup S) \Leftrightarrow (\sigma_{p(Ai)}(R)) \cup S$$

  – Where R_i belongs to *R* and *T*

- Commuting projection with binary operators

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

$$\Pi_C(R \bowtie_{Aj,Bk} S) \Leftrightarrow \Pi_{A'}(R) \bowtie_{Aj,Bk} \Pi_{B'}(S)$$

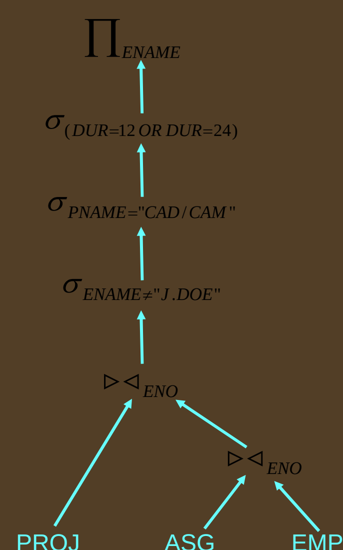
$$\Pi_C(R \cup S) \Leftrightarrow \Pi_C(R) \cup \Pi_C(S)$$

  – Where *R[A]* and *S[B]*; where $C = A' \cup B'$ $A' \subseteq A$ $B' \subseteq B$

(c)Oszu & Valduriez 28

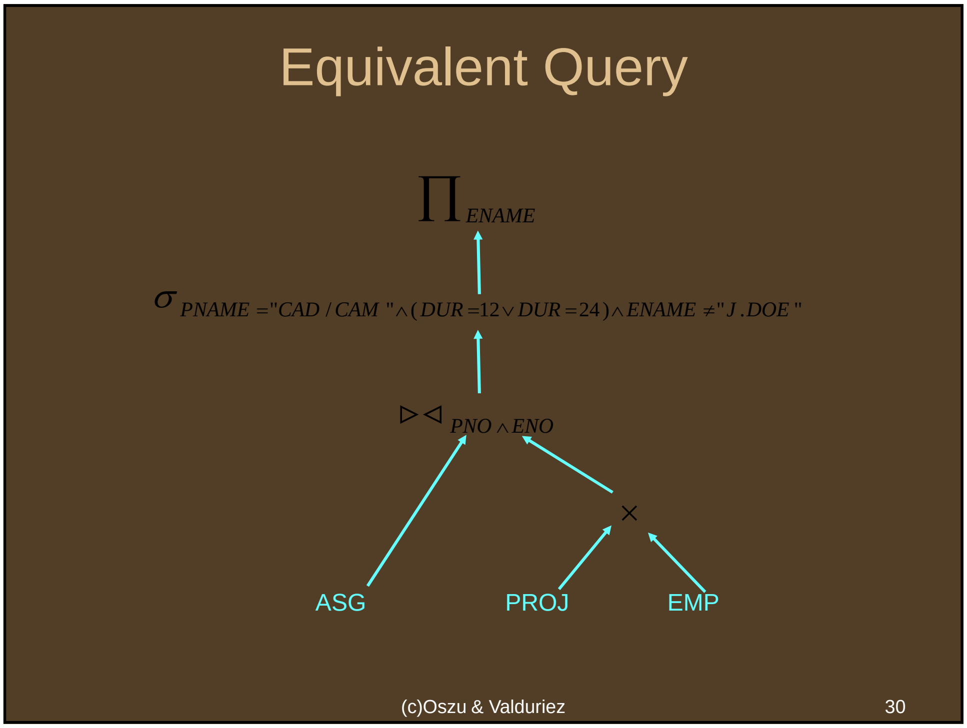
Example

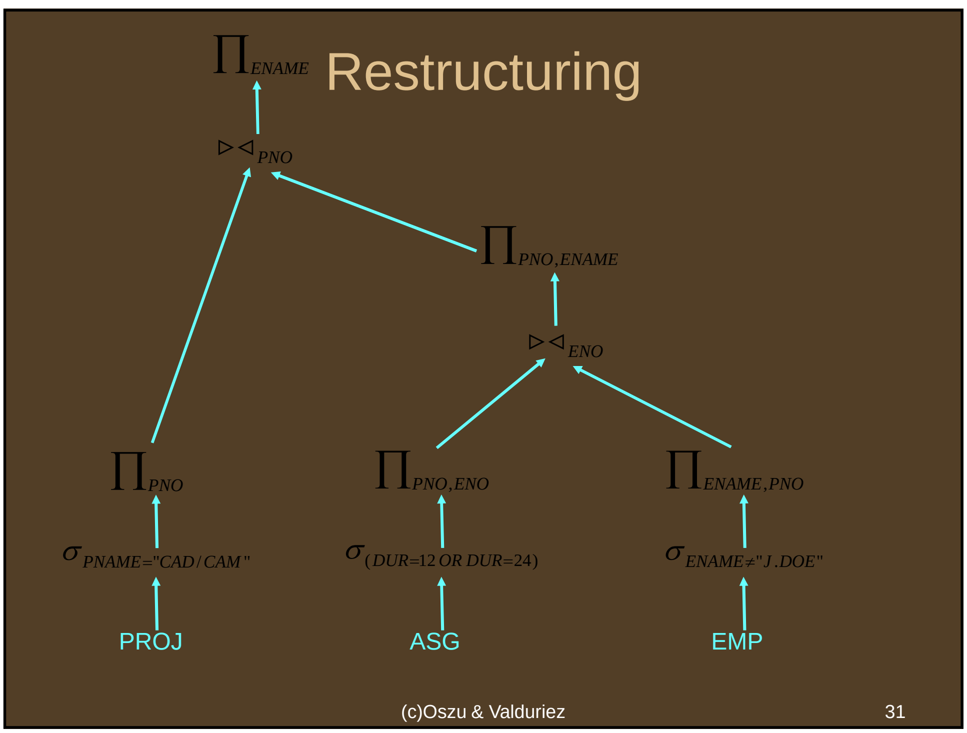
- Same query as before



SELECT	ENAME
FROM	EMP, ASG, PROJ
WHERE	EMP.ENO = ASG.ENO
AND	ASG.PNO = PROJ.PNO
AND	EMP.ENAME<>"J. DOE"
AND	PNAME = "CAD/CAM"
AND	**(**DUR =12 **OR** DUR = 24)

(c)Oszu & Valduriez 29


Equivalent Query
Π_{ENAME}
$\sigma_{PNAME="CAD/CAM" \wedge (DUR=12 \vee DUR=24) \wedge ENAME \neq "J.DOE"}$
$\bowtie_{PNO \wedge ENO}$
$\times$
ASG
PROJ
EMP
(c)Oszu & Valduriez
30


Restructuring
Π_{ENAME}
$\bowtie_{PNO}$
$\Pi_{PNO,ENAME}$
$\bowtie_{ENO}$
Π_{PNO}
$\Pi_{PNO,ENO}$
$\Pi_{ENAME,PNO}$
$\sigma_{PNAME="CAD/CAM"}$
$\sigma_{(DUR=12 \text{ OR } DUR=24)}$
$\sigma_{ENAME \neq "J.DOE"}$
PROJ
ASG
EMP
(c)Oszu & Valduriez
31

Data Localization

- Input: Algebraic query on distributed relations
- Determine which fragments are involved
- Localization program
  - Substitute for each global query its materialization program
  - optimize

(c)Oszu & Valduriez 33

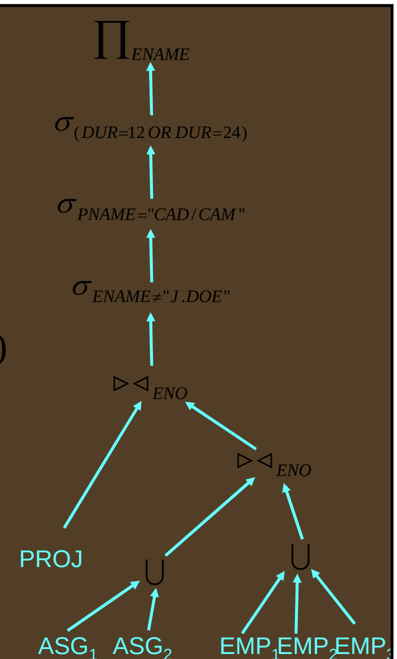
Data Localization

- Assume
  - EMP is fragmented as

$$EMP_1 = \sigma_{ENO \leq "E3"}(EMP)$$
$$EMP_2 = \sigma_{ENO > "E3" \wedge ENO \leq "E6"}(EMP)$$
$$EMP_3 = \sigma_{ENO > "E6"}(EMP)$$

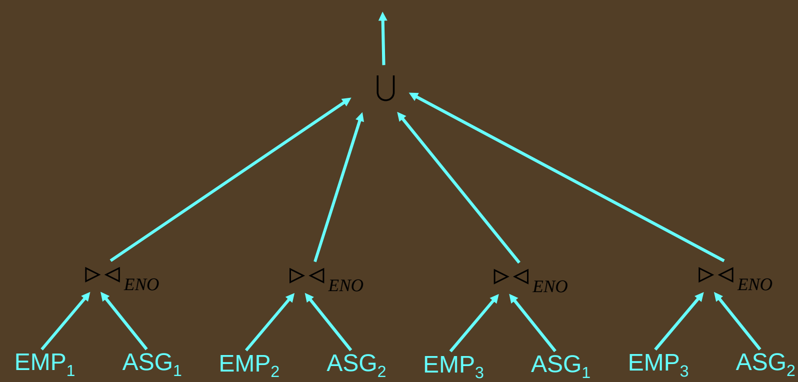
  - ASG is fragmented as

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



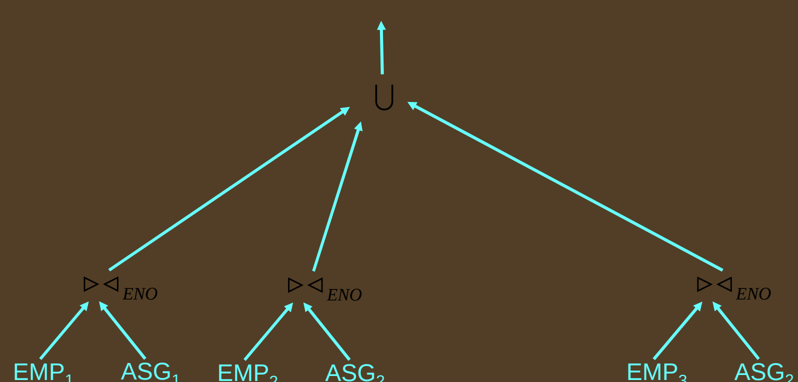
(c)Oszu & Valduriez 34

Provides Parallelism



(c)Oszu & Valduriez 35

Eliminates Unnecessary Work



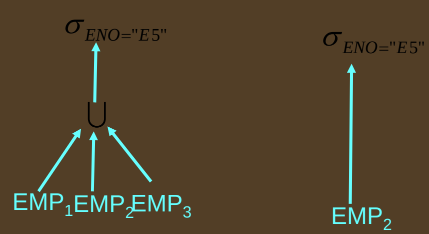
(c)Oszu & Valduriez 36

Reduction for PHF

- Reduction with selection
  - Relation R and F_R={R1, …, Rw}, where $R_j = \sigma_{pj}(R)$

$$\sigma_{pj}(R_i) = \phi \quad \text{if } \forall x \text{ in } R : \neg(p_i(x) \wedge p_j(x))$$

  - Example:
    - SELECT * FROM EMP WHERE ENO="E5"



(c)Oszu & Valduriez 37

Reduction for PHF

- Reduction with join
  - Possible if fragmentation is done on join attribute
  - Distribute join over unions

$$(R_1 \cup R_2) \rhd\lhd S \Leftrightarrow (R_1 \rhd\lhd S) \cup (R_2 \rhd\lhd S)$$

  - Given $R_i = \sigma_{pi}(R)$ and $R_j = \sigma_{pj}(R)$

$$R_i \rhd\lhd R_j = \phi \quad \text{if } \forall x \text{ in } R_i \; \forall y \text{ in } R_j : \neg(p_i(x) \wedge p_j(y))$$

(c)Oszu & Valduriez 38

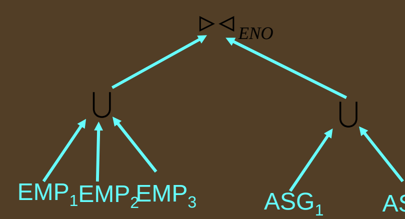
Reduction for PHF

- Reduction with join -- Example
  - Assume EMP fragmented as before, and

$$ASG_1 = \sigma_{ENO \le "E3"}(ASG)$$

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

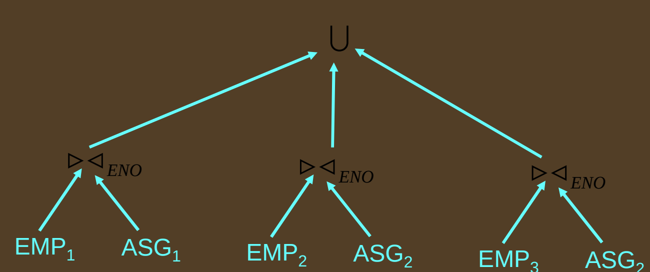
  - Example:
    - SELECT * FROM EMP,ASG WHERE EMP.ENO=ASG.ENO



(c)Oszu & Valduriez 39

Reduction for PHF

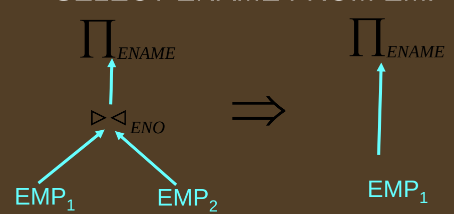
- Reduction with join -- Example
  - Distribute join over unions
  - Apply the reduction rule



(c)Oszu & Valduriez 40

Reduction for VF

- Find useless (not empty) intermediate relations
  - Relation R defined over attributes A={A1, …, An} vertically fragmented as $R_i = \Pi_{A'}(R)$ where A' is a subset of *A*
  - $P_{D,K}(R_i)$ is useless if D is not in A'
  - Example $EMP_1 = \Pi_{ENO,ENAME}(EMP)$, $EMP_2 = \Pi_{ENO,TITLE}(EMP)$
  - SELECT ENAME FROM EMP



(c)Oszu & Valduriez 41

Reduction for DHF

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

$$ASG_1 = ASG \rhd\!< _{ENO} (EMP_1)$$
$$ASG_2 = ASG \rhd\!< _{ENO} (EMP_2)$$
$$EMP_1 = \sigma_{TITLE=\text{"Programmer"}}(EMP)$$
$$EMP_2 = \sigma_{TITLE\neq\text{"Programmer"}}(EMP)$$

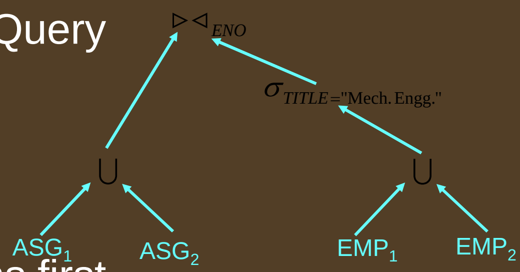
  - Query:

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

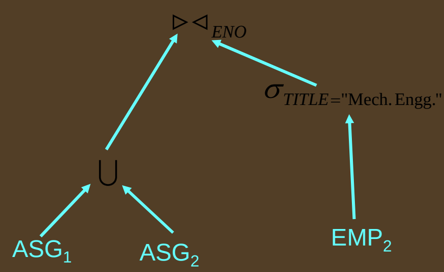
(c)Oszu & Valduriez 42

Reduction for DHF

- Generic Query



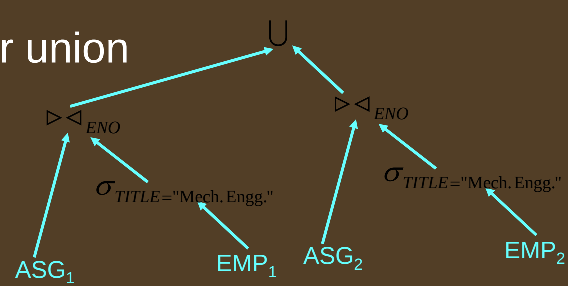
- Selections first



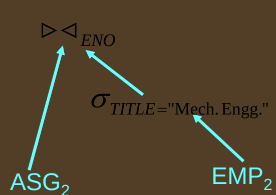
(c)Oszu & Valduriez 43

Reduction for DHF

- Joins over union



- Elimination of empty intermediate relations



(c)Oszu & Valduriez 44

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 eliminate useless joins

(c)Oszu & Valduriez 45

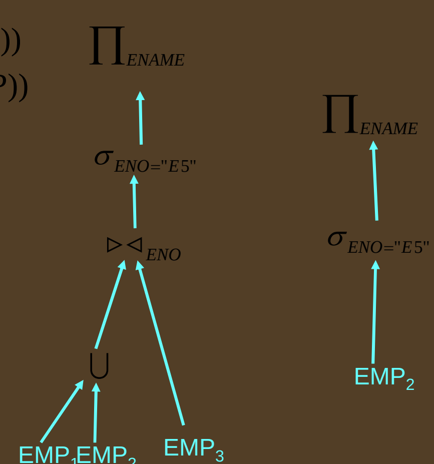
Reduction for HF

- Example

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

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

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



- Query

SELECT ENAME
FROM EMP
WHERE ENO="E5"

(c)Oszu & Valduriez 46

Step 3 – Global Optimization

- Input: Fragment query
- Find the best (not necessarily optimal) global schedule
  - Minimize a cost function
  - Distributed join processing
    - Bushy vs. linear trees
    - Which relation to ship where?
    - Ship-whole vs. ship-as-needed
  - Decide on use of semijoins
  - Join methods
    - Nested loop vs. ordered joins (merge join or hash join)

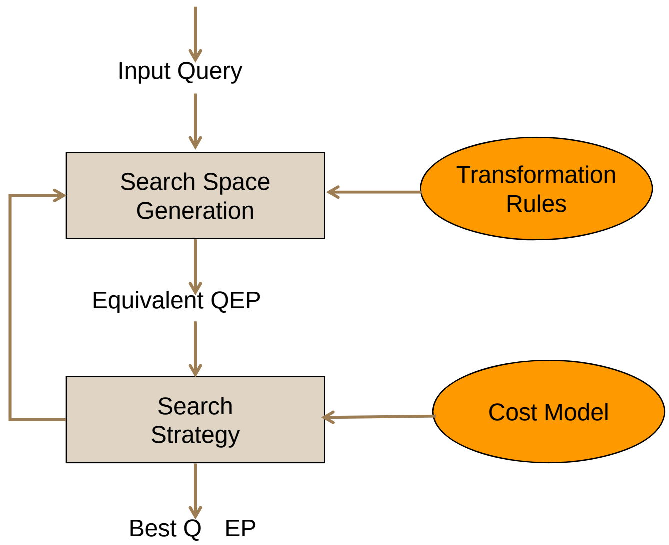
 48

Cost Based Optimization

- Solution space
  - The set of equivalent algebra expression (query trees)
  - Cost function (in terms of time)
    - I/O + CPU + Communication
    - Different weights
    - Can also maximize throughput
  - Search algorithm
    - How do we move inside the solution space?
    - Exhaustive search, heuristic algorithms (iterative improvement, simulated annealing, genetic, …)

 49

Optimization Process



(c)Oszu & Valduriez 50

Search Space

- Search space characterized by alternative execution plans
- Focus on join trees
- For *N* relations, there are *O(N!)* equivalent join trees that can be obtained by applying commutativity and associativity rules

```
SELECT   ENAME, RESP
FROM     EMP, ASG, PROJ
WHERE    EMP.ENO=ASG.ENO
AND      ASG.PNO=PROJ.PNO
```

(c)Oszu & Valduriez 51

Search Space

- Restrict by means of heuristics
  - Perform unary operations before binary operations
- Restrict the shape of the join tree
  - Consider only linear trees, ignore bushy ones

 52

Search Strategy

- How to “move” in the search space.
- Deterministic
  - Start from base relations and build plans adding one relation at each step
  - Dynamic programming: breadth-first
  - Greedy: depth first
- Randomized
  - Search for optimalities around a particular point
  - Trade opt. Time for execution time
  - Better when > 5-6 relations
  - Simulated annealing
  - Iterative improvement

 53

Cost Functions

- Total Time (or Total Cost)
  - Reduce each cost (in terms of time) component individually
  - Do as little of each component as possible
  - Optimizes the utilization of resources ➔ increases system throughput
- Response Time
  - Do as many things as possible in parallel
  - May increase total time because of increased total activity

 54

Total Cost

- Summation of all cost factors
  - Total cost = CPU cost + I/O cost + comm. Cost
  - CPU cost = unit instruction cost * no. of instructions
  - I/O cost = unit disk I/O cost * no. of disk I/Os
  - Communication cost = message initiation + transmission

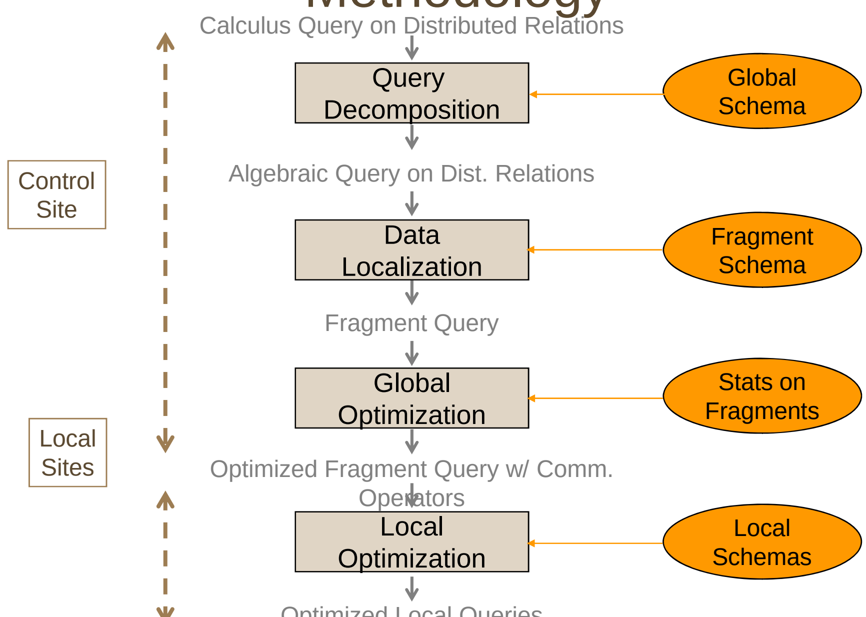
 55

Total Cost Factors

- Wide area networks
  - Message initiation and transmission costs high
  - Local processing cost is low (fast mainframes or minicomputers)
  - Ratio of comm to I/O costs = 20:1
- Local Area networks
  - Communication and local processing costs are more or less equal
  - Ratio = 1:1.6

(c)Oszu & Valduriez 56

Methodology



(c)Oszu & Valduriez 57

Optimization Statistics

- Primary cost factor: size of intermediate relations
- Make them precise ➔ more costly to maintain
  - For each relation
    - Length of each attribute: *length(Ai)*
    - The number of distinct values for each attribute in each fragment: $card(\Pi_{Ai}, Rj)$
    - Max and min values in each domain or each attribute
    - Cardinalities of each domain: *card(dom[Ai])*
    - Cardinalities of each fragment: *card(Rj)*
  - Selectivity factor of each operation for relations
    - For joins: $SF_{\triangleright\triangleleft}(R,S) = \frac{card(R \triangleright\triangleleft S)}{card(R) * card(S)}$

(c)Oszu & Valduriez 58

Intermediate Relation Size

- Selection
  - $Size(R) = card(SF_{\sigma}(R)) * length(R)$
  - $Card(\sigma_F(R)) = SF_{\sigma}(R) * card(R)$
  - *Where*
  - $SF_{\sigma}(A=value) = 1/(card(P_A(R)))$
  - $SF_{\sigma}(A>value) = (max(A)-value)/(max(A)-min(A))$
  - $SF_{\sigma}(A<value) = (value - min(A))/(max(A)-min(A))$
  - $SF_{\sigma}(p(Ai) \wedge p(Aj)) = SF_{\sigma}(P(Ai)) * SF_{\sigma}(P(Aj))$
  - $SF_{\sigma}(p(Ai) \vee p(Aj)) = SF_{\sigma}(p(Ai) + SF_{\sigma}(P(Aj) - SF_{\sigma}(p(Ai) * SF_{\sigma}(P(Aj)$
  - $SF_{\sigma}(A \text{ in } value) = SF_{\sigma}(A=value) * card(\{values\})$

(c)Oszu & Valduriez 59

Intermediate Relation Size

- Projection
  - $Card(P_A(R))=card(R)$
- Cartesian Product
  - Card(R x S) = card(R) * card(S)
- Union
  - Upper bound: card(R U S) = card(R)+ card(S)
  - Lower bound: card(R U S) = max{card(R), card(S)}
- Set difference
  - Upper bounds: card(R-S)= card(R)
  - Lower bounds: 0

(c)Oszu & Valduriez 60

Intermediate Relation Size

- Join
  - Special case: A is a key of R and B is a foreign key of S: $card(R \rhd\lhd_{A=B} S) = card(S)$
  - More general: $card(R \rhd\lhd S) = SF_{\rhd\lhd} * card(R) * card(S)$
- Semijoin

$$card(R \rhd<_A S) = SF_{\rhd<}(S.A) * card(R)$$

  - where

$$SF_{\rhd<}(R \rhd<_A S) = SF_{\rhd<}(S.A) = \frac{card(\prod_A(S))}{card(dom[A])}$$

(c)Oszu & Valduriez 61

Centralized Query Opt.

- INGRES
  - Dynamic
  - Interpretive
- System R
  - Static
  - Exhaustive search

INGRES Algorithm

- Decompose each multi-variable query into a sequence of mono-variable queries with a common variable
- Process each by a one variable query processor
  - Choose an initial execution plan (heuristics)
  - Order the rest by considering intermediate relation sizes

No statistical information is maintained.

INGRES – Decomposition

- Replace an *n* variable query *q* by a series of queries
  - $q_1 \rightarrow q_2 \rightarrow \ldots \rightarrow q_n$
  - Where q_i uses the result of q_{i-1}
- Detachment
  - Query *q* decomposed into $q' \rightarrow q''$ where *q'* and *q''* have a common variable which is the result of *q'*
- Tuple substitution
  - Replace the value of each tuple with actual values and simplify the query
  - $q(V_1, V_2, \ldots V_n) \rightarrow (q', (t_1, V_2, \ldots, V_n), t_1 \text{ in } R)$

Detachment

```
Q:    SELECT    V2.A2, V3.A3, …, Vn.An
      FROMR1 V1, …, Rn Vn
      WHERE     P1(V1.A1’) AND P2(V1.A1, … Vn.An)

Q’:   SELECT    V1.A1 INTO R1’
      FROMR1 V1
      WHERE     P1(V1.A1’)

Q’’:  SELECT    V2.A2, V3.A3, …, Vn.An
      FROMR1’ V1, …, Rn Vn
      WHERE     P2(V1.A1, … Vn.An)
```

Detachment Example

- Names of employees working in CAD/CAM

Q1:	SELECT	EMP.ENAME
	FROM	EMP, ASG, PROJ
	WHERE	EMP.ENO=ASG.ENO
	AND	ASG.PNO=PROJ.PNO
	AND	PROJ.PNAME= “CAD/CAM”
Q11:	SELECT	PROJ.PNO INTO JVAR
	FROM	PROJ
	WHERE	PROJ.PNAME=“CAD/CAM”
Q’:	SELECT	EMP.ENAME
	FROM	EMP, ASG, JVAR
	WHERE	EMP.ENO=ASG.ENO
	AND	ASG.PNO=JVAR.PNO

Detachment Ex. (cont.)

Q’:	SELECT	EMP.ENAME
	FROM	EMP, ASG, JVAR
	WHERE	EMP.ENO=ASG.ENO
	AND	ASG.PNO=JVAR.PNO
Q12:	SELECT	ASG.ENO INTO GVAR
	FROM	JVAR, ASG
	WHERE	ASG.PNO=JVAR.PNO
Q13:	SELECT	EMP.ENAME
	FROM	EMP, GVAR
	WHERE	EMP.ENO=GVAR.ENO

Tuple Substitution

- Q_{11} is a mono-variable query
- Q_{12} and Q_{13} are subject to tuple substitution
- Assume GVAR has two tuples only: <E1> <E2>
- Then q13 becomes

```
Q131:   SELECT   EMP.ENAME
        FROM EMP
        WHERE    EMP.ENO="E1"
Q132:   SELECT   EMP.ENAME
        FROM EMP
        WHERE    EMP.ENO="E2"
```


System R Algorithm

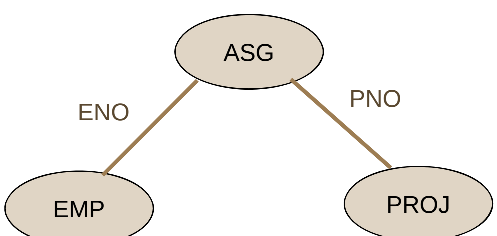
- Simple (I.e. mono-relation) queries are executed according to the best access path
- Execute joins
  - Determine the possible ordering of joins
  - Determine the cost of each ordering
  - Choose the join ordering with minimal cost

System R Algorithm

- For joins, two alternative algos:
- Nested Loops
  - – For each tuple of external relation (N1)
    - For each tuple of internal relation (N2)
      - – Join two tuples if predicate is true
    - End
  - – End
  - – Complexity: N1*N2
- Merge Join
  - – Sort relations
  - – Merge relations
  - – Complexity: N1+N2 if relations are sorted and equijoiin.

System R – Example

- Names of employees working on CAD/CAM
- Assume
  - – EMP has an index on ENO,
  - – ASG has an index on PNO,
  - – PROJ has an index on PNO and one on PNAME



System R Example (cont.)

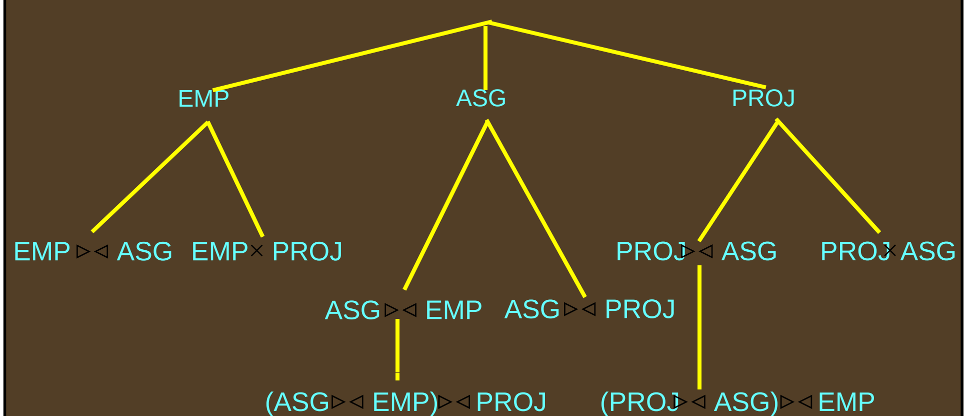
- Choose the best access paths to each relation
  - EMP: sequential scan (no selection on EMP)
  - ASG: sequential scan (no selection on ASG)
  - PROJ: index on PNAME
- Determine the best join ordering

$$EMP \bowtie ASG \bowtie PROJ$$
$$ASG \bowtie PROJ \bowtie EMP$$
$$PROJ \bowtie ASG \bowtie EMP$$
$$ASG \bowtie EMP \bowtie PROJ$$
$$EMP \times PROJ \bowtie ASG$$
$$PROJ \times EMP \bowtie ASG$$

  - Select the best based on join costs

(c)Oszu & Valduriez 73

System R Algorithm



Best total order is one of

$((ASG \bowtie EMP) \bowtie PROJ)$

$((PROJ \bowtie ASG) \bowtie EMP)$

(c)Oszu & Valduriez 74

System R Algorithm

- ((PROJ ASG) EMP) has a useful index on the select attribute and direct access to the join attribute of ASG and EMP
- Therefore, chose it with the following access methods:
  - Select PROJ using index on PNAME
  - Then join with ASG using index on PNO
  - Then join with EMP using index on ENO

(c)Oszu & Valduriez 75

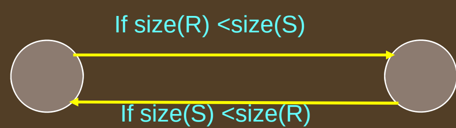
Join Ordering in Fragment Queries

- Ordering joins
  - Distributed INGRES
  - System R*
- Semijoin ordering
  - SDD-1

(c)Oszu & Valduriez 76

Join Ordering

- Consider two relation only



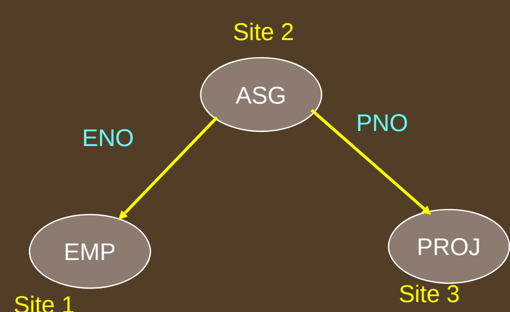
- Multiple relation more difficult because too many alternatives
  - Compute the cost of all alternatives and select the best one.
    - Necessary to compute the size of intermediate relations which is difficult
  - Use heuristics

(c)Oszu & Valduriez 77

Join Ordering – Example

Consider

PROJ $\bowtie$ ASG $\bowtie$ EMP



(c)Oszu & Valduriez 78

Semijoin Algorithms

- Consider the join of two relations:
  - R[A] (located at site 1)
  - S[A] (located at site 2)
- Alternatives
  - Do the join $R \bowtie_A S$
  - Perform one of the semijoin equivalents

$$R \bowtie_A S \Leftrightarrow (R \ltimes_A S) \bowtie_A S$$
$$R \bowtie_A S \Leftrightarrow R \bowtie_A (S \ltimes_A R)$$
$$R \bowtie_A S \Leftrightarrow (R \ltimes_A S) \bowtie_A (S \ltimes_A R)$$

(c)Oszu & Valduriez 79

Semijoin Algorithms

- Perform the join
  - Send R to site 2
  - Site 2 computes the join
- Consider semijoin $R \bowtie_A S \Leftrightarrow (R \ltimes_A S) \bowtie_A S$
  - $S' \leftarrow \prod_A(S)$
  - S’ → Site 1
  - Site 1 computes $R' = R \ltimes_A S'$
  - R’ → Site 2
  - Site 2 computes $R' \bowtie_A S$

Semijoin is better if

$$size(\prod_A(S)) + size(R \ltimes_A S)) < size(R)$$

(c)Oszu & Valduriez 80

Distributed INGRES Algorithm

- Same as centralized version except
  - Movement of relation (and fragments) need to be considered
  - Optimization with respect to communication cost or response time possible

R* Algorithm

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

R* Algorithm

- Performing Joins
- Ship Whole
  - Larger data transfer
  - Smaller number of messages
  - Better if relation are small
- Fetch as needed
  - Number of message – O(card of external relation)
  - Data transfer per message is minimal
  - Better if relations are large and selectivity is good.

 84

R*: Vertical part. and joins

- Move outer relation tuples to the site of the inner relation
  - Retrieve outer tuples
  - Send them to the inner relation site
  - Join them as they arrive
  - Total cost = cost(retrieving qualified outer tuples)+ no. of outer tuples fetched * cost(retrieving qualified inner tuples) + msg. Cost *(no. outer tuples fetched * avg. outer tuple size) /msg. size

 85

R*: Vertical part. and joins

- Move inner relation to the site of the outer reln.
  - Cannot join as they arrive; must be stored
  - Total cost = cost(retrieving qualified outer tuples) + no. of outer tuples fetched * cost(retrieving matching inner tuples from temp storage) + cost (retrieving qualified inner tuples) + cost(storing all qualified inner tuples in temp storage) + msg. Cost * (no. of inner tuples fetched * avg. inner tuple size) / msg. size

R*: Vertical part. & joins

- Move both inner and outer relation to another site
- Total cost = cost(retrieving qualified outer tuples) + cost (retrieving qualified inner tuples) + cost(storing inner tuples in storage) + msg. Cost* (no. of outer tuples fetched * avg. outer tuple size)/ msg. Size + msg. Cost*(# inner tuples fetched * avg. inner tuple size) / msg. Size + # outer tuples fetched * cost(retrieving inner tuples from temp storage)

R*: vertical part. & joins

- Fetch inner tuples as needed
  - Retrieve qualified tuples at outer relation site
  - Send request containing join column value(s) for outer tuples to inner relation site
  - Retrieve matching inner tuples at inner relation site
  - Send the matching inner tuples to outer relation site
  - Join as they arrive
  - Cost = cost(retr. Qual. Outer tuples) + msg. Cost * (# outer tuples fetched) + # inner tuples fetched*(#inner tuples fetched*avg. inner tuple size * msg. Cost/msg. Size) + # outer tuples fetched * cost(retrieving matching inner tuples for one outer value).

(c)Oszu & Valduriez 88

SDD-1 Algorithm

- Based on hill climbing algorithm
  - Semijoins
  - No replication
  - No fragmentation
  - Cost of transferring the result to the user site from the final result site is not considered
  - Can minimize either total time or response time

(c)Oszu & Valduriez 90

Hill Climbing Algorithm

- Assume join in between three relations
- Step 1: do initial processing
- Step 2: select initial feasible solution (ES_0)
  - Determine the candidate result sites – sites where a relation referenced in the query exists
  - Compute the cost of transferring all the other relns to each candidate site
  - ES_0 = candidate site with minimum cost
- Step 3: determine candidate splits of ES_0 into {ES_1, ES_2}
  - ES_1 consists of sending one of the relations to the other relations site
  - ES_2 consists of sending the join of the relations to the final result site.

(c)Oszu & Valduriez 91

Hill Climbing algorithm

- Step 4: Replace ES_0 with the split schedule which gives

  cost(ES_1) + cost(local join) + cost (ES_2) < cost(ES_0)
- Step 5: Recursively apply steps 3-4 on ES_1 and ES_2 until no such plans can be found
- Step 6: Check for redundant transmissions in the final plan and eliminate them.

(c)Oszu & Valduriez 92

Hill Climbing Example

- What are the salaries of engineers who work on the CAD/CAM project?

$$\Pi_{SAL}(PAY \rhd\lhd_{TITLE} (EMP \rhd\lhd_{ENO} (ASG \rhd\lhd_{PNO} (\sigma_{CAD/CAM}(PROJ)))))$$

Relation	Size	Site
EMP	8	1
PAY	4	2
PROJ	4	3
ASG	10	4

- Assume
  - Size of relations is defined as their cardinality
  - Minimize total cost
  - Transmission cost between two sites is 1
  - Ignore local processing cost

(c)Oszu & Valduriez 93

Hill Climbing example

- Step 1:
  - Selection on PROJ; result has cardinality 1

Relation	Size	Site
EMP	8	1
PAY	4	2
PROJ	1	3
ASG	10	4

(c)Oszu & Valduriez 94

Hill Climibing example

- Step 2: initial feasible solution
  - Alt 1: resulting site is site 1
    - Total cost = cost(PAY→ site1) + cost(ASG→site1) + cost(PROJ→ site1) = 4+10+1=15
  - Alt 2: Resulting site is site 2
    - Total cost = 8+10+1 = 19
  - Alt 3: Resulting site is site 3
    - Total cost = 8 + 4 + 10 = 22
  - Alt 4: Resulting site is site 4
    - Total cost = 8 + 4 + 1 = 13
  - Therefore ES_0={EMP→ Site 4; S → site 4; PROJ → Site 4}

(c)Oszu & Valduriez 95

Hill Climbing example

- Step 3: Determine candidate splits
  - Alternative 1: {ES1, ES2, ES3} where
    - ES1: $EMP \rightarrow Site2$
    - ES2: $(EMP \triangleright\triangleleft PAY) \rightarrow Site4$
    - ES3: $PROJ \rightarrow Site4$
  - Alternative 2: {ES1, ES2, ES3} where
    - ES1: $PAY \rightarrow Site1$
    - ES2: $(PAY \triangleright\triangleleft EMP) \rightarrow Site4$
    - ES3: $PROJ \rightarrow Site4$

(c)Oszu & Valduriez 96

Hill Climbing

- Step 4: Determine the cost of split alternative
  - Cost(Alt 1) = cost(EMP→ Site 2) + cost (Join) + cost (PROJ→ Site 4
    = 8 + 8 + 1 = 17
  - Cost(Alt 2) = cost(PAY→ Site 1) + cost (join) + cost (PROJ → site 4)
    = 4 + 8 + 1 = 13
  - Decision : do not split
- Step 5: ES_0 is the best
- Step 6: No redundant transmissions.

(c)Oszu & Valduriez 97

Hill Climbing

- Problems
  - Greedy algo → determines an initial feasible solution and iteratively tries to improve it
  - If there are local minimas, it may not find global minima
  - If the optimal schedule has a high initial cost, it won't find it since it won't choose it as the initial feasible solution
  - Example: a better solution is
    - PROJ → Site 4
    - ASG' = (PROJ join ASG) → site 1
    - (ASG' join EMP) → site 2
    - Total cost = 1 + 2 + 2 = 5

(c)Oszu & Valduriez 98