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.
Selecting Alternatives

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

Strategy 1:

\[ \Pi_{ENAME} \left( \sigma_{EMP.ENO=ASG.ENO \land DUR>37} (EMP \times ASG) \right) \]

Strategy 2:

\[ \Pi_{ENAME} (EMP \bowtie_{ENO} (\sigma_{DUR>37}(ASG))) \]

Strategy 2, avoids cartesian product.

(c)Oszu & Valduriez

Problem

Site 1

ASG_1 = \sigma_{DUR>37}(ASG)

Site 2

EMP_2 = \sigma_{DUR>37}(EMP)

Site 3

ASG_2 = \sigma_{DUR>37}(ASG)

Site 4

EMP_4 = \sigma_{DUR>37}(EMP)

Site 5

RESULT

Site 1

\[ EMP_1' = EMP_1 \bowtie_{ENO} ASG_1' \]

Site 2

\[ EMP_2' = EMP_2 \bowtie_{ENO} ASG_2' \]

Site 3

\[ ASG_1' = \sigma_{DUR>37}(ASG) \]

Site 4

\[ ASG_2' = \sigma_{DUR>37}(ASG) \]

Site 5

RESULT = EMP_1' \cup EMP_2'

(c)Oszu & Valduriez
Alternative 2

\[
\text{RESULT} = (\text{EMP}_1 \cup \text{EMP}_2) \bowtie_{\text{ENO}} \sigma_{\text{DUR} > 37} (\text{ASG}_1 \cup \text{ASG}_2)
\]

Site 5

ASG\_1
Site 1

ASG\_2
Site 2

EMP\_1
Site 3

EMP\_2
Site 4

Cost of Alternatives

- **Assume**
  - \(\text{Size}(\text{EMP}) = 400; \, \text{size}(\text{ASG}) = 1000\)
  - Tuple access cost (TAC) = 1 unit; tuple xfer cost (TXC) = 10 units

- **Strategy 1**
  - Produce ASG\': \((10+10) \times \text{TAC} = 20\)
  - Transfer ASG\': \((10+10) \times \text{TXC} = 200\)
  - Produce EMP\': \((10+10) \times \text{TAC} \times 2 = 40\)
  - Transfer EMP\' to result site: \((10+10) \times \text{TXC} = 200\)
  - Total COST = 460.
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!!

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
Query Optimization Objectives

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

(c)Oszu & Valduriez
### Complexity of Relational Operators

<table>
<thead>
<tr>
<th>Operation</th>
<th>Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Select, Project (without duplicate elimination)</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>Project (w/ duplicate elimination)</td>
<td>$O(n \log n)$</td>
</tr>
<tr>
<td>Group</td>
<td>$O(n \log n)$</td>
</tr>
<tr>
<td>Join</td>
<td>$O(n \log n)$</td>
</tr>
<tr>
<td>Semijoin</td>
<td>$O(n \log n)$</td>
</tr>
<tr>
<td>Division</td>
<td>$O(n \log n)$</td>
</tr>
<tr>
<td>Set Operators</td>
<td>$O(n \log n)$</td>
</tr>
<tr>
<td>Cartesian Product</td>
<td>$O(n^2)$</td>
</tr>
</tbody>
</table>

Assume Relations of cardinality $n$
Sequential scan

(c)Oszu & Valduriez

### 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

(c)Oszu & Valduriez
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

(c)Oszu & Valduriez 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

(c)Oszu & Valduriez 14
Issues: Optimization Timing

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

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
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 or reln. Calculus
- Simplification
  - Eliminate redundant predicates
- Restructuring
  - Calculus query → algebra 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 quantification
- Put into normal form
  - Conjunctive normal form
  - Disjunctive normal form
  - ORs mapped into union
  - ANDs 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 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
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"

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

• Why simplify?
  – Remember the example

• How? Use transformation rules
  – Elimination of redundancy
    • Idempotency rules
  – Application of transitivity
  – Use of integrity rules

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"
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.

\[
\begin{align*}
\text{SELECT} & \quad ENAME \\
\text{FROM} & \quad \text{EMP, ASG, PROJ} \\
\text{WHERE} & \quad \text{EMP.ENO = ASG.ENO} \\
\text{AND} & \quad \text{ASG.PNO = PROJ.PNO} \\
\text{AND} & \quad \text{EMP.ENAME} \neq \text{"J. DOE"} \\
\text{AND} & \quad \text{PNAME} = \text{"CAD/CAM"} \\
\text{AND} & \quad (\text{DUR} = 12 \text{ OR DUR} = 24)
\end{align*}
\]

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
  \[
  (\prod_{A'} (\prod_A (R))) \Leftrightarrow \prod_A (R) \\
  (\sigma_{p_1(A_1)}(\sigma_{p_2(A_2)}(R))) \Leftrightarrow \sigma_{p_1(A_1)\bowtie p_2(A_2)}(R)
  \]
  - Where \(R[A]\) and \(A' \subseteq A\)

- Commuting selection with projection

\[\text{(c)Oszu & Valduriez}\]

Transformation Rules

- Commuting selection with binary operators

\[ \sigma_{p(A)}(R \times S) \Leftrightarrow (\sigma_{p(A)}(R)) \times S \]
\[ \sigma_{p(A)}(R \bowtie_{A_i, B_k} S) \Leftrightarrow (\sigma_{p(A)}(R)) \bowtie_{A_i, B_k} S \]
\[ \sigma_{p(A)}(R \cup S) \Leftrightarrow (\sigma_{p(A)}(R)) \cup S \]

- Where \( R_i \) belongs to \( R \) and \( T \)

- Commuting projection with binary operators

\[ \Pi_C(R \times S) \Leftrightarrow \Pi_{A'}(R) \times \Pi_{B'}(S) \]
\[ \Pi_C(R \bowtie_{A_i, B_k} S) \Leftrightarrow \Pi_{A'}(R) \bowtie_{A_i, B_k} \Pi_{B'}(S) \]
\[ \Pi_C(R \cup S) \Leftrightarrow \Pi_C(R) \cup \Pi_C(S) \]

- Where \( R[A] \) and \( S[B] \); where \( C = A' \cup B' \), \( A' \subseteq A \), \( B' \subseteq B \)

Example

- Same query as before

```sql
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 & Valdurie 28 29
Equivalent Query

\[ \Pi_{\text{ENAME}} \]
\[ \sigma_{\text{PNAME} = "CAD / CAM" \land (\text{DUR} = 2 \lor \text{DUR} = 24) \land \text{ENAME} \neq "J.DOE"} \]

Restructuring

\[ \Pi_{\text{ENAME}} \]
\[ \sigma_{\text{PNAME} = "CAD / CAM"} \]
\[ \sigma_{(\text{DUR} = 12 \lor \text{DUR} = 24)} \]
\[ \sigma_{\text{ENAME} = "J.DOE"} \]
Data Localization

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

\[ EMP_1 = \sigma_{ENO \leq E3} (EMP) \]
\[ EMP_2 = \sigma_{ENO > E3 \land ENO \leq E6} (EMP) \]
\[ EMP_3 = \sigma_{ENO > E6} (EMP) \]
- ASG is fragmented as
  \[ ASG_1 = \sigma_{ENO \leq E3} (ASG) \]
  \[ ASG_2 = \sigma_{ENO > E3} (ASG) \]

(c)Osuzu & Valduriez

(c)Osuzu & Valduriez
Provides Parallelism

Eliminates Unnecessary Work
Reduction for PHF

- Reduction with selection
  - Relation R and \( F_R = \{R_1, \ldots, R_w\} \), where \( R_j = \sigma_{p_j}(R) \)
  - Example:
    - \( \text{SELECT * FROM EMP WHERE ENO="E5"} \)

Reduction for PHF

- Reduction with join
  - Possible if fragmentation is done on join attribute
  - Distribute join over unions
    \( (R_1 \cup R_2) \bowtie S \iff (R_1 \bowtie S) \cup (R_2 \bowtie S) \)
  - Given \( R_i = \sigma_{p_i}(R) \) and \( R_j = \sigma_{p_j}(R) \)
    \( R_i \bowtie R_j = \phi \) if \( \forall x \) in \( R_i \) \( \forall y \) in \( R_j : \neg(p_i(x) \land p_j(y)) \)
Reduction for PHF

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

  \[ \text{ASG}_1 = \sigma_{\text{ENO} \leq 3^x} (\text{ASG}) \]
  \[ \text{ASG}_2 = \sigma_{\text{ENO} > 3^x} (\text{ASG}) \]

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

Reduction for PHF

• Reduction with join -- Example
  – Distribute join over unions
  – Apply the reduction rule
Reduction for VF

- Find useless (not empty) intermediate relations
  - Relation R defined over attributes A={A1, ..., An} vertically fragmented as Ri= ΠA′(R) where A′ is a subset of A
  - PD,K(Ri) is useless if D is not in A′
- Example EMP1=ΠENO,ENAME(EMP), EMP2=ΠENO,TITLE(EMP)
- SELECT ENAME FROM EMP

Reduction for DHF

- Rule:
  - Distribute join over unions
  - Apply the join reduction for horizontal fragmentation
- Example
  \[ \begin{align*}
  ASG_1 &= ASG \bowtie_{ENO} (EMP_1) \\
  ASG_2 &= ASG \bowtie_{ENO} (EMP_2) \\
  EMP_1 &= \sigma_{TITLE="Programmer"} (EMP) \\
  EMP_2 &= \sigma_{TITLE="#Programmer"} (EMP)
  \end{align*} \]
- Query:
  SELECT * 
  FROM EMP, ASG 
  WHERE ASG.ENO=EMP.ENO 
  AND EMP.TITLE="Mech. Engg"
Reduction for DHF

• Generic Query

\[ \sigma_{\text{TITLE}="\text{Mech. Engg."}} (\text{ASG}_1 \cup \text{ASG}_2 \cup \text{EMP}_1 \cup \text{EMP}_2) \]

\[ \sigma_{\text{ENQ}} (\text{ASG}_1 \cup \text{ASG}_2 \cup \text{EMP}_1 \cup \text{EMP}_2) \]

• Selections first

\[ \sigma_{\text{TITLE}="\text{Mech. Engg."}} (\text{ENQ}) \]

\[ \sigma_{\text{ENQ}} (\text{ASG}_1 \cup \text{ASG}_2 \cup \text{EMP}_1 \cup \text{EMP}_2) \]

• Joins over union

\[ \sigma_{\text{TITLE}="\text{Mech. Engg."}} (\text{ASG}_1 \cup \text{ASG}_2 \cup \text{EMP}_1 \cup \text{EMP}_2) \]

\[ \sigma_{\text{ENQ}} (\text{ASG}_1 \cup \text{ASG}_2 \cup \text{EMP}_1 \cup \text{EMP}_2) \]

• Elimination of empty intermediate relations

\[ \sigma_{\text{TITLE}="\text{Mech. Engg."}} (\text{ENQ}) \]

\[ \sigma_{\text{ENQ}} (\text{ASG}_1 \cup \text{ASG}_2 \cup \text{EMP}_1 \cup \text{EMP}_2) \]
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

Example

\[
\begin{align*}
EMP_1 &= \sigma_{\text{ENO}=\text{E}4^*}(\Pi_{\text{ENO}, \text{ENAME}}(EMP)) \\
EMP_2 &= \sigma_{\text{ENO}=\text{E}4^*}(\Pi_{\text{ENO}, \text{ENAME}}(EMP)) \\
EMP_3 &= \Pi_{\text{ENO}, \text{TITLE}}(EMP)
\end{align*}
\]

Query

```
SELECT \text{ENAME} \\
FROM EMP \\
WHERE \text{ENO}=\text{E}5^*
```
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)

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, …)
Optimization Process

Input Query → Search Space Generation → Equivalent QEP → Search Strategy → Best QEP

Transformation Rules → Cost Model

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
```
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

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
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 \rightarrow increases system throughput

- **Response Time**
  - Do as many things as possible in parallel
  - May increase total time because of increased total activity

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
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
Optimization Statistics

- Primary cost factor: size of intermediate relations
- Make them precise \( \rightarrow \) more costly to maintain
  - For each relation
    - Length of each attribute: \( \text{length}(A_i) \)
    - The number of distinct values for each attribute in each fragment: \( \text{card}(I_{A_p}R_j) \)
    - Max and min values in each domain or each attribute
    - Cardinalities of each domain: \( \text{card}(\text{dom}[A_i]) \)
    - Cardinalities of each fragment: \( \text{card}(R_j) \)
  - Selectivity factor of each operation for relations
    - For joins:
      \[
      SF_{\sigma}(R, S) = \frac{\text{card}(R \bowtie S)}{\text{card}(R) \cdot \text{card}(S)}
      \]

Intermediate Relation Size

- Selection
  - \( \text{Size}(R) = \text{card}(SF_\sigma(R)) \cdot \text{length}(R) \)
  - \( \text{Card}(\sigma_{F}(R)) = SF_\sigma(R) \cdot \text{card}(R) \)
  - Where
    - \( SF_\sigma(A=\text{value}) = \frac{1}{\text{card}(P_A(R))} \)
    - \( SF_\sigma(A>\text{value}) = (\max(A)-\text{value})/(\max(A)-\min(A)) \)
    - \( SF_\sigma(A<\text{value}) = (\text{value} - \min(A))/(\max(A)-\min(A)) \)
    - \( SF_\sigma(p(A_i)^p(A_j)) = SF_\sigma(p(A_i)) \cdot SF_\sigma(p(A_j)) \)
    - \( SF_\sigma(p(A_i) \lor p(A_j)) = SF_\sigma(p(A_i)) \cdot SF_\sigma(p(A_j)) - SF_\sigma(p(A_i))^p(A_i) \cdot SF_\sigma(p(A_j)) \)
    - \( SF_\sigma(A \in \text{value}) = SF_\sigma(A=\text{value}) \cdot \text{card}\{\text{values}\} \)
Intermediate Relation Size

• Projection
  – Card($P_A(R)$) = Card($R$)

• Cartesian Product
  – Card($R \times S$) = Card($R$) * Card($S$)

• Union
  – Upper bound: Card($R \cup S$) = Card($R$) + Card($S$)
  – Lower bound: Card($R \cup S$) = max{Card($R$), Card($S$)}

• Set difference
  – Upper bounds: Card($R - S$) = Card($R$)
  – Lower bounds: 0

(c) Oszu & Valduriez

Intermediate Relation Size

• Join
  – Special case: $A$ is a key of $R$ and $B$ is a foreign key of $S$: Card($R \bowtie_{A,B} S$) = Card($S$)
  – More general: Card($R \bowtie S$) = Card($R$) * Card($S$)

• Semijoin
  \[
  \text{Card}(R \bowtie_{A} < S) = SF_{<}(S.A) \times \text{Card}(R)
  \]
  – where
  \[
  SF_{<}(R \bowtie_{A} S) = SF_{<}(S.A) = \frac{\text{Card}(\Pi_{A}(S))}{\text{Card(dom}[A])}
  \]

(c) Oszu & Valduriez
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

   FROM R1 V1, ..., Rn Vn
   WHERE P1(V1.A1') AND P2(V1.A1, ... Vn.An)

Q': SELECT V1.A1 INTO R1'
    FROM R1 V1
    WHERE P1(V1.A1')

Q'': SELECT V2.A2, V3.A3, ..., Vn.An
     FROM R1' 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

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 \( q_{13} \) becomes

\[
\begin{align*}
Q_{131}: & \quad \text{SELECT EMP.ENAME} \\
& \quad \text{FROM EMP} \\
& \quad \text{WHERE EMP.ENO=} \text{“E1”}
\end{align*}
\]

\[
\begin{align*}
Q_{132}: & \quad \text{SELECT EMP.ENAME} \\
& \quad \text{FROM EMP} \\
& \quad \text{WHERE EMP.ENO=} \text{“E2”}
\end{align*}
\]

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 equijoin.

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

\[
\begin{align*}
& \text{EMP} \bowtie \text{ASG} \bowtie \text{PROJ} \\
& \text{ASG} \bowtie \text{PROJ} \bowtie \text{EMP} \\
& \text{PROJ} \bowtie \text{ASG} \bowtie \text{EMP} \\
& \text{ASG} \bowtie \text{EMP} \bowtie \text{PROJ} \\
& \text{EMP} \bowtie \text{PROJ} \bowtie \text{ASG} \\
& \text{PROJ} \bowtie \text{EMP} \bowtie \text{ASG}
\end{align*}
\]

- Select the best based on join costs

\[
\begin{align*}
& \text{ASG} \bowtie \text{EMP} \bowtie \text{PROJ} \\
& \text{PROJ} \bowtie \text{ASG} \bowtie \text{EMP} \\
& \text{EMP} \bowtie \text{ASG} \bowtie \text{PROJ}
\end{align*}
\]

System R Algorithm

Best total order is one of

\[
\begin{align*}
& ((\text{ASG} \bowtie \text{EMP}) \bowtie \text{PROJ}) \\
& ((\text{PROJ} \bowtie \text{ASG}) \bowtie \text{EMP})
\end{align*}
\]
System R Algorithm

• \(((\text{PROJ} \quad \text{ASG}) \quad \text{EMP})\) 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

Join Ordering in Fragment Queries

• Ordering joins
  – Distributed INGRES
  – System R*

• Semijoin ordering
  – SDD-1
Join Ordering

• Consider two relation only

  If size(R) < size(S)

  If size(S) < size(R)

• 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

Join Ordering – Example

Consider

\[ \text{PROJ} \bowtie \text{ASG} \bowtie \text{EMP} \]

(c)Oszu & Valduriez

77

(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
    
    $R \bowtie A S \iff (R \bowtie A S) \bowtie A S$
    
    $R \bowtie A S \iff R \bowtie A (S < A R)$
    
    $R \bowtie A S \iff (R < A S) \bowtie A (S < A R)$

(c)Oszu & Valduriez 79

Semijoin Algorithms

- Perform the join
  - Send $R$ to site 2
  - Site 2 computes the join

- Consider semijoin $R \bowtie A S \iff (R \bowtie A S) \bowtie A S$
  - $S' \leftarrow \Pi_A(S)$
  - $S' \rightarrow$ Site 1
  - Site 1 computes $R' = R \bowtie A S'$
  - $R' \rightarrow$ Site 2
  - Site 2 computes $R' \bowtie A S$

Semijoin is better if

\[
\text{size}(\Pi_A(S)) + \text{size}(R \bowtie A S) < \text{size}(R)
\]

(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 – $O(\text{card of external relation})$
  – Data transfer per message is minimal
  – Better if relations are large and selectivity is good.

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
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

(c)Oszu & Valduriez 86

---

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)

(c)Oszu & Valduriez 87
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)Osuzu & 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)Osuzu & Valduriez 90
Hill Climbing Algorithm

• Assume join in between three relations
• Step 1: do initial processing
• Step 2: select initial feasible solution (ES₀)
  – 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
  – ES₀ = candidate site with minimum cost
• Step 3: determine candidate splits of ES₀ into {ES₁, ES₂}
  – ES₁ consists of sending one of the relations to the other relations site
  – ES₂ consists of sending the join of the relations to the final result site.

(c)Oszu & Valduriez 91

Hill Climbing algorithm

• Step 4: Replace ES₀ with the split schedule which gives
  \[ \text{cost}(ES₁) + \text{cost}(\text{local join}) + \text{cost}(ES₂) < \text{cost}(ES₀) \]
• Step 5: Recursively apply steps 3-4 on ES₁ and ES₂ 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?

\[ \Pi_{\text{ML}} (PAY \bowtie \sigma_{\text{TITLE}} (EMP \bowtie \sigma_{\text{ENO}} (ASG \bowtie \sigma_{\text{PNO}} (\sigma_{\text{CAD/CAM}} (PROJ)))))) \]

<table>
<thead>
<tr>
<th>Relation</th>
<th>Size</th>
<th>Site</th>
</tr>
</thead>
<tbody>
<tr>
<td>EMP</td>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td>PAY</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>PROJ</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>ASG</td>
<td>10</td>
<td>4</td>
</tr>
</tbody>
</table>

- 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

<table>
<thead>
<tr>
<th>Relation</th>
<th>Size</th>
<th>Site</th>
</tr>
</thead>
<tbody>
<tr>
<td>EMP</td>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td>PAY</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>PROJ</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>ASG</td>
<td>10</td>
<td>4</td>
</tr>
</tbody>
</table>

(c)Oszu & Valduriez 94
Hill Climbing example

• Step 2: initial feasible solution
  – Alt 1: resulting site is site 1
    • Total cost = cost(PAY $\rightarrow$ site1) + cost(ASG $\rightarrow$ site1) + cost(PROJ $\rightarrow$ 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 $\rightarrow$ Site 4; S $\rightarrow$ Site 4; PROJ $\rightarrow$ Site 4}

Hill Climbing example

• Step 3: Determine candidate splits
  – Alternative 1: {ES1, ES2, ES3} where
    • ES1: $\text{EMP} \rightarrow \text{Site} 2$
    • ES2: $(\text{EMP} \bowtie \text{PAY}) \rightarrow \text{Site} 4$
    • ES3: $\text{PROJ} \rightarrow \text{Site} 4$
  – Alternative 2: {ES1, ES2, ES3} where
    • ES1: $\text{PAY} \rightarrow \text{Site} 1$
    • ES2: $(\text{PAY} \bowtie \text{EMP}) \rightarrow \text{Site} 4$
    • ES3: $\text{PROJ} \rightarrow \text{Site} 4$
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: ES₀ is the best
• Step 6: No redundant transmissions.

(c)Oszu & Valduriez

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