QUERY PROCESSING & OPTIMIZATION

CHAPTER 19 (6/E)
CHAPTER 15 (5/E)
LECTURE OUTLINE

- Query Processing Methodology
- Basic Operations and Their Costs
- Generation of Execution Plans
QUERY PROCESSING IN A DDBMS

- high level user query
- query processor
- Low-level data manipulation commands for D-DBMS
SELECTING ALTERNATIVES

\[
\begin{align*}
\text{SELECT} & \quad \text{ENAME} \\
\text{FROM} & \quad \text{EMP, ASG} \\
\text{WHERE} & \quad \text{EMP.ENO} = \text{ASG.ENO} \\
\text{AND} & \quad \text{ASG.RESP} = "Manager"
\end{align*}
\]

Strategy 1

\[
\Pi_{\text{ENAME}}(\sigma_{\text{RESP}=\text{"Manager"} \land \text{EMP.ENO} = \text{ASG.ENO}}(\text{EMP} \times \text{ASG}))
\]

Strategy 2

\[
\Pi_{\text{ENAME}}(\text{EMP} \bowtie_{\text{ENO}} (\sigma_{\text{RESP}=\text{"Manager"}} (\text{ASG}))
\]

Strategy 2 avoids Cartesian product, so may be “better”
Strategy 1

\[ \text{Strategy 1} \]

\[ \Pi_{\text{ENAME}} \]

\[ \sigma_{\text{ASG.RESP}='Manager' \land \text{EMP.ENO} = \text{ASG.ENO}} \]

EMP \quad ASG

Strategy 2

\[ \text{Strategy 2} \]

\[ \Pi_{\text{ENAME}} \]

\[ \bowtie \]

\[ \sigma_{\text{ASG.RESP}='Manager'}} \]

EMP \quad ASG
**QUERY PROCESSING METHODOLOGY**

- Query in high-level language
  - Query Decomposition & Translation
    - Algebraic Query
  - Query Optimization
    - Query Execution Plan
  - Query Code Generator
    - Code to execute query
  - Runtime Processor
    - Query Result

**SQL**
- check SQL syntax
- check existence of relations and attributes
- replace views by their definitions
- transform query into an internal form

*generate alternative access plans, i.e., procedure, for processing the query*

*select an efficient access plan*
EXAMPLE

SELECT V.Vno, Vname, count(*), sum(Amount) 
FROM Vendor V, Transaction T 
WHERE V.Vno = T.Vno 
AND V.Vno between 1000 and 2000 
GROUP BY V.Vno, Vname 
HAVING sum(Amount) > 100

- Scan the Vendor table, select all tuples where Vno = [1000, 2000], eliminate attributes other than Vno and Vname, and place the result in a temporary relation $R_1$.

- Join the tables $R_1$ and Transaction, eliminate attributes other than Vno, Vname, and Amount, and place the result in a temporary relation $R_2$. This may involve:
  - sorting $R_1$ on Vno
  - sorting Transaction on Vno
  - merging the two sorted relations to produce $R_2$

- Perform grouping on $R_2$, and place the result in a temporary relation $R_3$. This may involve:
  - sorting $R_2$ on Vno and Vname
  - grouping tuples with identical values of Vno and Vname
  - counting the number of tuples in each group, and adding their Amounts

- Scan $R_3$, select all tuples with sum(Amount) > 100 to produce the result.
EXAMPLE

\[
\text{SELECT } V.Vno, Vname, \text{count}(*), \text{sum}(\text{Amount}) \\
\text{FROM } \text{Vendor } V, \text{Transaction } T \\
\text{WHERE } V.Vno = T.Vno \\
\text{AND } V.Vno \text{ between 1000 and 2000} \\
\text{GROUP BY } V.Vno, Vname \\
\text{HAVING } \text{sum}(\text{Amount}) > 100
\]
QUERY OPTIMIZATION ISSUES

- Determining the “shape” of the execution plan
  - Order of execution
- Determining which how each “node” in the plan should be executed
  - Operator implementations
- These are interdependent and an optimizer would do both in generating the execution plan
“SHAPE” OF THE EXECUTION PLAN

- Finding query trees that are “equivalent”
  - Produce the same result – provably
- These are based on the transformation (equivalence) rules
- Commutativity of selection
  - \( \sigma_{p_1(A_1)}(\sigma_{p_2(A_2)}(R)) \iff \sigma_{p_2(A_2)}(\sigma_{p_1(A_1)}(R)) \)
- Commutativity of binary operations
  - \( R \times S \iff S \times R \)
  - \( R \bowtie S \iff S \bowtie R \)
  - \( R \cup S \iff S \cup R \)
  - \( R \cap S \iff S \cap R \)
- Associativity of binary operations
  - \( (R \times S) \times T \iff R \times (S \times T) \)
  - \( (R \bowtie S) \bowtie T \iff R \bowtie (S \bowtie T) \)
  - \( (R \cup S) \cup T \iff (S \cup R) \cup T \)
- Cascading of unary operations
  - \( \Pi_{A'}(\Pi_{A''}(R)) \iff \Pi_{A''}(R) \) where \( R[A] \) and \( A' \subseteq A, A'' \subseteq A \) and \( A' \subseteq A'' \)
  - \( \sigma_{p_1(A_1)}(\sigma_{p_2(A_2)}(R)) \iff \sigma_{p_1(A_1) \cap p_2(A_2)}(R) \)
OTHER TRANSFORMATION RULES

- Commuting selection with projection
  \[ \Pi_B(\sigma_{p(A)} R) \Leftrightarrow \sigma_{p(A)}(\Pi_B R) \quad (\text{where } B \subseteq A) \]

- Commuting selection with binary operations
  \begin{align*}
  &\sigma_{p(A)}(R \times S) \Leftrightarrow (\sigma_{p(A)} (R)) \times S \quad (\text{where } A \text{ belongs to } R \text{ only}) \\
  &\sigma_{p(A_i)}(R \bowtie_{(A_j \cdot B_k)} S) \Leftrightarrow (\sigma_{p(A_i)} (R)) \bowtie_{(A_j \cdot B_k)} S \quad (\text{where } A_i \text{ belongs to } R \text{ only}) \\
  &\sigma_{p(A_i)}(R \cup S) \Leftrightarrow \sigma_{p(A_i)} (R) \cup \sigma_{p(A_i)} (S) \quad (\text{where } A_i \text{ belongs to } R \text{ and } S) \\
  &\sigma_{p(A_i)}(R \cap S) \Leftrightarrow \sigma_{p(A_i)} (R) \cap \sigma_{p(A_i)} (s) \quad (\text{where } A_i \text{ belongs to } R \text{ and } S)
  \end{align*}

- Commuting projection with binary operations
  \begin{align*}
  &\Pi_C(R \times S) \Leftrightarrow \Pi_{A'}(R) \times \Pi_B(S) \\
  &\Pi_C(R \bowtie_{(A_j \cdot B_k)} S) \Leftrightarrow \Pi_{A'}(R) \bowtie_{(A_j \cdot B_k)} \Pi_B(S) \\
  &\Pi_C(R \cup S) \Leftrightarrow \Pi_C(R) \cup \Pi_C(S) \\
  &\Pi_C(R \cap S) \Leftrightarrow \Pi_C(R) \cap \Pi_C(S)
  \end{align*}

where \( R[A] \) and \( S[B] \); \( C = A' \cup B' \) where \( A' \subseteq A \), \( B' \subseteq B \)
Find the names of employees other than J. Doe who worked on the CAD/CAM project for either one or two years.

\[
\text{SELECT ENAME}
\]
\[
\text{FROM PROJ P, ASG G, EMP E}
\]
\[
\text{WHERE G.ENO=E.ENO AND G.PNO=P.PNO AND E.ENAME <> 'J. Doe' AND P.PNAME='CAD/CAM' AND (G.DUR=12 OR G.DUR=24)}
\]
EQUIVALENT QUERY

\[ \Pi_{E.ENAME} \]
\[ \sigma_{P.PNAME='CAD/CAM' \land (G.DUR=12 \land G.DUR=24) \land E.ENAME<>'J. Doe'} \]
\[ \bowtie_{PNO,ENO} \]

EMP  PROJ  ASG
ANOTHER EQUIVALENT QUERY

\[ \pi_{\text{ENAME}} \left( \sigma_{\text{ENAME} \neq 'J. Doe'} \left( \pi_{\text{PNO, ENAME}} \left( \sigma_{\text{PNAME} = 'CAD/CAM'} \left( \pi_{\text{PNO}} \left( \sigma_{\text{DUR} = 12 \land \text{DUR} = 24} \left( \pi_{\text{PNO, ENO}} \left( \sigma_{\text{ENAME} \neq 'J. Doe'} \left( \pi_{\text{PNO, ENAME}} \left( \pi_{\text{PNO}} \right) \right) \right) \right) \right) \right) \right) \right) \right) \]

\[ \Join \text{PNO} \]

\[ \Join \text{EN} \]

\[ \Join \text{O} \]

\[ \Join \text{EN} \]

\[ \Join \text{PROJ} \]

\[ \Join \text{ASG} \]

\[ \Join \text{EMP} \]
CLICKER QUESTION #36

- Is the right query plan equivalent to the left query plan?

(a) Yes
(b) No
- Assume you have

\[ R \bowtie S \bowtie T \bowtie W \]

- Most systems implement linear join trees
  - Left-linear
JOIN ORDERING

- Even with left-linear, how do you know which order?
  - Assume natural join over common attributes
SOME OPERATOR IMPLEMENTATIONS

- Tuple Selection
  - without an index
  - with a clustered index
  - with an unclustered index
  - with multiple indices
- Projection
- Joining
  - nested loop join
  - sort-merge join
  - and others...
- Grouping and Duplicate Elimination
  - by sorting
  - by hashing
- Sorting
EXAMPLE – JOIN ALGORITHMS

```
SELECT C.Cnum, A.Balance
FROM Customer C, Accounts A
WHERE C.Cnum = A.Cnum
```

- Nested loop join:

  ```
  for each tuple c in Customer do
    for each tuple a in Accounts do
      if c.Cnum = a.Cnum then
        output c.Cnum, a.Balance
      end
    end
  end
  ```
EXAMPLE – JOIN ALGORITHMS (2)

SELECT  C.Cnum, A.Balance
FROM    Customer C, Accounts A
WHERE   C.Cnum = A.Cnum

- Index join:

  for each tuple c in Customer do
  use the index to find Accounts tuples a
  where a.Cnum matches c.Cnum
  if there are any such tuples a then
  output c.Cnum, a.Balance
  end
  end

- Sort-merge join:

  sort Customer and Accounts on Cnum
  merge the resulting sorted relations
### COMPLEXITY OF OPERATORS

- Assume
  - Relations of cardinality $n$
  - Sequential scan

<table>
<thead>
<tr>
<th>Operation</th>
<th>Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Select</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>Project (without duplicate elimination)</td>
<td></td>
</tr>
<tr>
<td>Project (with duplicate elimination)</td>
<td>$O(n \times \log n)$</td>
</tr>
<tr>
<td>Group</td>
<td></td>
</tr>
<tr>
<td>Join</td>
<td>$O(n \times \log n)$</td>
</tr>
<tr>
<td>Semi-join</td>
<td></td>
</tr>
<tr>
<td>Division</td>
<td></td>
</tr>
<tr>
<td>Set Operators</td>
<td></td>
</tr>
<tr>
<td>Cartesian Product</td>
<td>$O(n^2)$</td>
</tr>
</tbody>
</table>
COST OF PLANS

- Alternative access plans may be compared according to cost.
- The cost of an access plan is the sum of the costs of its component operations.
- There are many possible cost metrics. However, most metrics reflect the amounts of system resources consumed by the access plan. System resources may include:
  - disk block I/O’s
  - processing time
  - network bandwidth
LECTURE SUMMARY

- Query processing methodology
- Basic query operations and their costs
- Generation of execution plans