Introduction

- Alternative ways of evaluating a given query
  - Equivalent expressions
  - Different algorithms for each operation

(a) Initial expression tree

(b) Transformed expression tree
Relational algebra optimization

• Many ways to get the same result
  – Equivalent relational algebra expressions
  – Different algorithms for processing expressions

• Questions:
  – What are equivalent?
  – How do we determine what is best?

• Transformation rules
  – (preserve equivalence)
  – What are good transformations?

Rules: Natural joins & cross products & union

• \( R \bowtie S = S \bowtie R \)
• \( (R \bowtie S) \bowtie T = R \bowtie (S \bowtie T) \)

• \( R \times S = S \times R \)
• \( (R \times S) \times T = R \times (S \times T) \)

• \( R \cup S = S \cup R \)
• \( R \cup (S \cup T) = (R \cup S) \cup T \)
Note:

- Carry attribute names in results, so order is not important
- Can also write as trees, e.g.:

```
T ≡ R
S
```

Introduction

- Alternative ways of evaluating a given query
  - Equivalent expressions
  - Different algorithms for each operation
Introduction (Cont.)

- An **evaluation plan** defines exactly what algorithm is used for each operation, and how the execution of the operations is coordinated.

  ![Diagram of query evaluation plan]

  - Find out how to view query execution plans on your favorite database

  - Most database support `explain <query>`
    - Displays plan chosen by query optimizer, along with cost estimates
    - Some syntax variations between databases
      - Oracle: `explain plan for <query>` followed by `select * from table (dbms_xplan.display)`
      - SQL Server: `set showplan_text on`

  - Some databases (e.g. PostgreSQL) support `explain analyse <query>`
    - Shows actual runtime statistics found by running the query, in addition to showing the plan

  - Some databases (e.g. PostgreSQL) show cost as `f..l`
    - `f` is the cost of delivering first tuple and `l` is cost of delivering all results
Optimization: Transform Query Plan

• Find the “tree” that gives the fastest response
  – All must give the same answer
  – Fewest IOs
• Rule-based Query Optimization
  – Transformations we know will always help
  – Independent of data values
• Cost-based Query Optimization
  – Estimate cost based on data

Equivalent Query Plans

• Give the same set of tuples on EVERY legal database instance
  – Looking only at the schema
  – In practice, ignore integrity constraints
  – Note: since dealing with SQL, consider multiset semantics
• Equivalence Rule
  – Transformation that can be applied to a small set of operations as part of the larger tree
  – Algebra…
Rules: Selects

- $\sigma_{p_1 \land p_2}(R) = \sigma_{p_1}[\sigma_{p_2}(R)]$
- $\sigma_{p_1 \lor p_2}(R) = [\sigma_{p_1}(R)] \cup [\sigma_{p_2}(R)]$

Let: $X = \text{set of attributes}$
$Y = \text{set of attributes}$
$XY = X \cup Y$

$\pi_{xy}(R) = \pi_x[\pi_y(R)]$
Multisets vs. Sets

- \( R = \{a,a,b,b,b,c\} \)
- \( S = \{b,b,c,c,d\} \)
- \( RUS = ? \)
  - **Option 1** SUM
    \[ RUS = \{a,a,b,b,b,b,b,c,c,c,d\} \]
  - **Option 2** MAX
    \[ RUS = \{a,a,b,b,b,c,c,d\} \]

Option 2 (MAX) makes this rule work:

\[ \sigma_{p_1 \vee p_2} (R) = \sigma_{p_1} (R) \cup \sigma_{p_2} (R) \]

**Example:** \( R=\{a,a,b,b,b,c\} \)
- \( P1 \) satisfied by \( a,b \); \( P2 \) satisfied by \( b,c \)
- \( \sigma_{p_1 \vee p_2} (R) = \{a,a,b,b,b,c\} \)
- \( \sigma_{p_1} (R) = \{a,a,b,b,b\} \)
- \( \sigma_{p_2} (R) = \{b,b,b,c\} \)
- \( \sigma_{p_1} (R) \cup \sigma_{p_2} (R) = \{a,a,b,b,b,c\} \)
“Sum” option makes more sense:

<table>
<thead>
<tr>
<th>T1</th>
<th>Yr</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>97</td>
<td>CA</td>
<td></td>
</tr>
<tr>
<td>99</td>
<td>CA</td>
<td></td>
</tr>
<tr>
<td>98</td>
<td>AZ</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>T2</th>
<th>Yr</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>99</td>
<td>CA</td>
<td></td>
</tr>
<tr>
<td>98</td>
<td>CA</td>
<td></td>
</tr>
</tbody>
</table>

Union?

Rules: $\sigma + \bowtie$ combined

- Let $p = \text{predicate with only R attribs}$
  $q = \text{predicate with only S attribs}$
  $m = \text{predicate with only R,S attribs}$

- $\sigma_p (R \bowtie S) = [\sigma_p (R)] \bowtie S$
- $\sigma_q (R \bowtie S) = R \bowtie [\sigma_q (S)]$
Some Rules can be Derived:
• $\sigma_{p \land q} (R \Join S) =$
• $\sigma_{p \land q \land m} (R \Join S) =$
• $\sigma_{p \lor q} (R \Join S) =$

Derivation for first one
• $\sigma_{p \land q} (R \Join S) =$
  $- \sigma_{p} [\sigma_{q} (R \Join S)] =$
  $- \sigma_{p} [R \Join \sigma_{q} (S)] =$
• $[\sigma_{p} (R) \Join [\sigma_{q} (S)]]$
Rules: $\Pi, \sigma$ combined

Let

$- x = \text{subset of } R \text{ attributes}$

$- z = \text{attributes in predicate } P \text{ (subset of } R \text{ attributes)}$

$\Pi_x[\sigma_p (R) ] = \Pi_x\{\sigma_p (\Pi_{xz}(R))\}$

Rules: $\pi, \bowtie$ combined

Let

$x = \text{subset of } R \text{ attributes}$

$y = \text{subset of } S \text{ attributes}$

$z = \text{intersection of } R,S \text{ attributes}$

$\Pi_{xy} (R \bowtie S) = \Pi_{xy}\{[\Pi_{xz}(R)] \bowtie [\Pi_{yz}(S)]\}$
\[ \Pi_{xy} \{ \sigma_p (R \bowtie S) \} = \]
\[ \Pi_{xy} \{ \sigma_p [\Pi_{xz'} (R) \bowtie \Pi_{yz'} (S)] \} \]
\[ z' = z \cup \{ \text{attributes used in } P \} \]

Rules for \( \sigma, \pi \) combined with \( X \)

• similar...
• e.g., \( \sigma_p (R \times S) = ? \)
Rules: $\sigma$, $U$ combined:

\[
\sigma_p(R \cup S) = \sigma_p(R) \cup \sigma_p(S)
\]

\[
\sigma_p(R - S) = \sigma_p(R) - S = \sigma_p(R) - \sigma_p(S)
\]

Which are “good” transformations?

\[
\sigma_{p1 \land p2} (R) \rightarrow \sigma_{p1} [\sigma_{p2} (R)]
\]

\[
\sigma_p (R \Join S) \rightarrow [\sigma_p (R)] \Join S
\]

\[
R \Join S \rightarrow S \Join R
\]

\[
\Pi_x [\sigma_p (R)] \rightarrow \Pi_x \{\sigma_p [\Pi_{xz} (R)]\}
\]
Conventional wisdom: do projects early

- Example: $R(A,B,C,D,E) \quad x = \{E\}$
  $P: (A=3) \land (B=\text{“cat”})$

- $\pi_x \{\sigma_p (R)\}$ vs. $\pi_E \{\sigma_p \{\pi_{ABE}(R)\}\}$

What if we have A, B indexes?

B = “cat”  \[\text{Intersect pointers to get}\]

A=3  \[\text{pointers to matching tuples}\]
Bottom line:

• No transformation is always good
• Usually good: early selections
• More transformations:
  – Eliminate common sub-expressions
  – Other operations: duplicate elimination