Class Agenda

• Last time:
  – More physical operators: selection, projection, duplicate elimination, aggregates
  – Quiz review

• Today:
  – Quiz #1
  – Query optimization

• Reading
  – Chapter 15 of Ramakrishnan and Gehrke (or Chapter 14 of Silberschatz et al)
Announcements

Reminder: DavisDB Part 2 due Sunday @11:59pm

Statistics re DavisDB Part 1:

<table>
<thead>
<tr>
<th>Statistic</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>AVG</td>
<td>76/100</td>
</tr>
<tr>
<td>MEDIAN</td>
<td>82/100</td>
</tr>
<tr>
<td>STDEV</td>
<td>25</td>
</tr>
<tr>
<td>MIN</td>
<td>26/100</td>
</tr>
<tr>
<td>MAX</td>
<td>111/100</td>
</tr>
</tbody>
</table>
Relational Query Optimization
Query Optimization

• Given a SQL query:
  – Build a *logical query plan*: tree of algebraic operations
  – Transform into "better" logical plan
  – Convert into a *physical query plan*, using implementations of operators we've seen in the previous lectures

• Goal: find the physical query plan that has minimum cost
  – In practice: avoid the plans with the highest costs
  – Sources of cost: Interactions with other concurrent tasks; sizes of intermediate results; choices of algorithms, access methods; I/O and CPU; properties of data such as skew, order, placement; ...
Optimization Strategies

• Many possible strategies, all boil down to a search over the space of possible plans
  – Super-exponential complexity in the # of operators
  – Hence, exhaustive search generally not feasible

• What can you do?
  – Heuristics only: INGRES, Oracle until the mid-90s
  – Randomized, simulated annealing, ... : many efforts in the mid-90s
  – **Heuristics plus cost-based join enumeration: System R**
  – Stratified search (heuristics plus cost-based enumeration of joins and a few other operators): Starbust
  – Unified search (full cost-based search): EXODUS, Volcano, Cascades
Highlights of System R Optimizer

• Historically, the most influential optimizer design

• Cost estimation: approximate art at best
  – Statistics, maintained in system catalogs, used to estimate cost of operations and result sizes
  – Considers combination of CPU and I/O costs

• Plan space: too large, must be pruned using heuristics
  – Only the space of left-deep plans is considered
  – *Pipelined execution model*: output of each operator is pipelined into the next operator, without storing it in a temporary relation
  – Cartesian products avoided

• Dynamic programming approach
Query Blocks: Units of Optimization in System R

- SQL query parsed into a collection of *query blocks*, to be optimized one block at a time

- Nested blocks treated as calls to a subroutine, made once per outer tuple

- For each block, the plans considered are
  - All available access methods, for each relation in `from` clause
  - All *left-deep join trees*: i.e., all ways to join the relations one-at-a-time, with the inner relation in the `from` clause, considering all join order permutations and join methods

```sql
select S.name
from Sailors S
where S.age in (select max(S2.age)
    from Sailors S2
    group by S2.rating)
```
Left-Deep Join Trees

• Left-deep join tree:

\[ \text{R \& U} \]
\[ \text{R \& T} \]
\[ \text{R \& S} \]

• "Bushy" join tree:

\[ \text{R \& S \& T \& U} \]
Relational Algebra Equivalences

- Allow us to choose different join orders; to "push" selections and projections ahead of joins; etc

1. \( \sigma_{F_1}(\sigma_{F_2}(E)) \equiv \sigma_{F_1 \land F_2}(E) \)

2. \( \sigma_F(E_1 [\cup, \cap, -] E_2) \equiv \sigma_F(E_1) [\cup, \cap, -] \sigma_F(E_2) \)

3. \( \sigma_F(E_1 \times E_2) \equiv \sigma_{F_0}(\sigma_{F_1}(E_1) \times \sigma_{F_2}(E_2)); \)
   \( F \equiv F_0 \land F_1 \land F_2, F_i \) contains only attributes of \( E_i, i = 1, 2 \).

4. \( \sigma_{A=B}(E_1 \times E_2) \equiv E_1 \bowtie_{A=B} E_2 \)

5. \( \pi_A(E_1 [\cup, \cap, -] E_2) \equiv \pi_A(E_1) [\cup, \cap, -] \pi_A(E_2) \)
Relational Algebra Equivalences (2)

6. \( \pi_A(E_1 \times E_2) \equiv \pi_{A1}(E_1) \times \pi_{A2}(E_2) \),
   with \( Ai = A \cap \{ \text{attributes in } E_i \} \), \( i = 1, 2 \).

7. \( E_1 [\cup, \cap] E_2 \equiv E_2 [\cup, \cap] E_1 \)
   \( (E_1 \cup E_2) \cup E_3 \equiv E_1 \cup (E_2 \cup E_3) \) (the analogous holds for \( \cap \))

8. \( E_1 \times E_2 \equiv \pi_{A1,A2}(E_2 \times E_1) \)
   \( (E_1 \times E_2) \times E_3 \equiv E_1 \times (E_2 \times E_3) \)
   \( (E_1 \times E_2) \times E_3 \equiv (E_1 \times E_3) \times E_2 \)

9. \( E_1 \bowtie E_2 \equiv E_2 \bowtie E_1 \) \( (E_1 \bowtie E_2) \bowtie E_3 \equiv E_1 \bowtie (E_2 \bowtie E_3) \)

(Theoretical aside: is this set of equivalences complete?)
Enumeration of Alternative Plans

• There are two main cases:
  – Single-relation plans
  – Multiple-relation plans

• Single-relation plans: queries consist of a combination of selections, projections, and aggregates (no joins)
  – Each available access path (file or index scan) is considered, and the one with the least estimated cost is chosen
  – The different operations are carried out together in a pipeline (e.g., if an index is used for a selection, projection is done for each retrieved tuple, and the resulting tuples are pipelined into the aggregate computation)
Cost Estimation

• Must estimate cost of each plan considered

• To do this, must estimate cost of each operation in plan tree
  – Depends on input cardinalities, statistical properties, etc

• Must also estimate size of result for each operation in tree!
  – Use information about the input relations
  – For selections and joins, assume independence of predicates

• Dirty little secret of DBMS world: estimation works well for simple plans, but poorly for complex plans
Queries Over Multiple Relations

• Fundamental heuristic in System R: *only left-deep join trees are considered*

• As the # of joins increases, the # of alternative plans grows very rapidly; we need to restrict the search space

• Left-deep join trees allow us to generate all *fully pipelined* plans
  – i.e., intermediate results not written to temporary files (not "materialized")
  – not all left-deep physical plans are fully pipelined

• Bushy join trees: can't have fully pipelined plans
  – Inner table must always be materialized for each tuple of the outer table
  – So, a plan in which the inner table is the result of a join forces us to materialize the result of that join
Enumeration of Left-Deep Plans

• Left-deep plans differ only in the order of relations, the access method for each relation, and the join method for each join

• Enumeration via dynamic programming strategy: \( n \) passes, where \( n = \# \) relations joined
  
  – Pass 1: find best 1-relation plan for each relation
  
  – Pass 2: find best way to join result of each 1-relation plan (as outer) to another relation
  
  – Pass \( n \): find best way to join result of each \((n-1)\)-relation plan (as outer) to the \( n \)th relation

• For each subset of relations, retain only:
  
  – Cheapest plan overall, plus
  
  – Cheapest plan for each "interesting order" of the tuples