CPSC 421
Database Management Systems

Lecture 18:
Query Optimization

* Some material adapted from R. Ramakrishnan, L. Delcambre, and B. Ludaescher

Agenda

• No quiz!
• More on Query Optimization
“On the fly”

- In “on they fly” evaluation, each operator is performed in memory while the tuple is available (streaming!)

- On-the-fly evaluation *induces no I/O cost!*

- Note: Relation on left is assumed to be the outer relation

One Possible Plan:

\[
\begin{align*}
\pi_{\text{sname}} & \quad \text{(on-the-fly)} \\
\sigma_{\text{bid}=100 \land \text{rating}>5} & \quad \text{(on-the-fly)} \\
\preceq & \quad \text{(Page NL Join)} \\
\text{Reserves} & \quad \text{Sailors}
\end{align*}
\]

Limitations of “On the fly”

- Can only happen if:
  - Computation can be done entirely on tuples in memory
  - Results do not need to be materialized

- Cannot be used for first table scan …
  - Someone has to read the table

- Cannot be used with all operations

One Possible Plan:

\[
\begin{align*}
\pi_{\text{sname}} & \quad \text{(on-the-fly)} \\
\sigma_{\text{bid}=100 \land \text{rating}>5} & \quad \text{(on-the-fly)} \\
\preceq & \quad \text{(Page NL Join)} \\
\text{Reserves} & \quad \text{Sailors}
\end{align*}
\]
Cost of Plan 1

- Assume:
  - No index
  - Sailors inner relation

- Cost of page-oriented nested loops join is:
  - \( M + M \times N \)
  - \( 1000 + 1000 \times 500 = 501,000 \)

- And on-the-fly ops have no I/O – so plan cost is 501,000!

### Generating More Plans

How can we generate more plans?

- Use equivalences to produce new query trees
  - Then find a plan for each of these

- Or assign different algorithms to operators
  - That is, generate more plans from one query tree

- Example:
  - Apply commutativity of join to the current tree …
**Cost of Plan 2**

- **Assume:**
  - No index
  - Reserves inner relation

- **Cost of page-oriented nested loops join is:**
  - \( N + N \times M \)
  - \( 500 + 500 \times 1000 = 500,500 \)

- And on the fly ops have no I/O – so plan cost is 500,500!

**Select**

\[
\begin{align*}
\pi_{\text{sname}} & \quad \text{(on-the-fly)} \\
\sigma_{\text{bid}=100 \land \text{rating}>5} & \quad \text{(on-the-fly)} \\
\bowtie & \quad \text{(Page NL Join)} \\
\text{Sailors} & \quad \text{Reserves}
\end{align*}
\]

**Cost of Plan 3**

- **Push down selects:**
  \( \sigma_e(R \bowtie S) = \sigma_e(R) \bowtie S \)

- **What is the cost now?**
  - Scan Sailors and Reserves costs \( M + N \) I/Os
  - What about the join?
  - Depends on how many reservations for boat 100 and how many sailors have a rating > 5

- **How do we find this information?**
  - Guess …
  - Better: *Use the statistics kept by the DBMS!*

Select

\[
\begin{align*}
\pi_{\text{sname}} & \quad \text{(on-the-fly)} \\
\sigma_{\text{bid}=100} & \quad \text{(Table Scan)} \\
\sigma_{\text{rating}>5} & \\
\text{Sailors} & \quad \text{Reserves}
\end{align*}
\]

**SELECT**

\[
\begin{align*}
\text{S.sname} \\
\text{FROM} & \quad \text{Reserves R, Sailors S} \\
\text{WHERE} & \quad \text{R.sid = S.sid AND} \\
& \quad \text{R.bid = 100 AND} \\
& \quad \text{S.rating > 5;}
\end{align*}
\]
Estimating costs

- For all operators (except leaf-level of the query plan)
  - The input tables are the result of some earlier query
  - Thus, we need to estimate the size of intermediate results!

  - This can be difficult …
  - This is a reason why cost estimates may not be very good
  - Estimation errors tend to compound

  - For example, what information would you need to estimate the number of reservations for bid = 100?
  - Or how many sailors have a rating higher than 5?

The DBMS Catalog and Statistics

- A Catalog typically contains at least:
  - The # of tuples for each table
  - The # of pages for each table
  - The # of distinct key values and # of pages for each index
  - The index height, low/high key values for each tree index
  - A distribution of data values …

- Catalogs are updated “periodically”
  - This could be after each insert, daily, weekly, monthly, after each backup …

- Simplest case: Assume values uniformly distributed
  - Thus, for a gender attribute …
  - Assume half the rows have the male value and the other half have the female value
  - This may be grossly inaccurate!
Calculating Selectivities

Assume that “rating” values range from 1 to 10
And that “bid” values range from 1 to 100

• What percentage of the incoming tuples to the operator $\sigma_{\text{bid}=100}$ will be output?

• What about $\sigma_{\text{rating}>5}$?

• What about $\sigma_{\text{bid}=100 \land \text{rating}>5}$?
  – More later …

Improving Uniform Distributions

• The DBMS might gather more detailed information about how the values of attributes distribute
  – Often using “histograms” of the values in a field
  – The histograms are stored in the catalog

• Suppose there was an attribute “degree-program” with 3 possible values “BS/CS”, “MS/CS”, “PhD/CS”
  – The DBMS might count the values and know there are 428 BS/CS values, 98 MS/CS values, and 25 PhD/CS values
  – This allows for a much better estimate of the “reduction factor” …
Reduction Factor (RF)

A “reduction factor” (RF) is associated with each term …

- The fraction of tuples in the table that satisfy a given conjunct
- Reflects the impact of the term in reducing the result size
- A value between 0 and 1 … often expressed as a percentage (e.g., a reduction of 30%)
  - Confusingly, more is less …
- For a term

\[ \text{Result Cardinality} = \text{Cardinality} \times \text{Reduction Factor} \]

Independence of Reduction Factors

- How do we use distributions for multiple conditions?
  - That is, a select operator with terms separated by AND …
  - E.g., \( \sigma_{\text{bid}=100 \land \text{rating}>5} \)
- We assume that all terms are \textit{independent}!
- If one attribute is “class” and the other is “num_hours”
  - The optimizer might assume that “class” is uniformly distributed over {Freshman, Sophomore, Junior, Senior}
  - And that “num_hours” uniformly distributes over \{0, 1, …, 144\}
  - But we know that class correlates with credit hours! (E.g., it could be that num_hours \( \xrightarrow{} \) class)
Independence of Reduction Factors

• What percentage of the input tuples, to the operator $\sigma_{\text{bid}=100 \land \text{rating}>5}$ will be output?
  – What is the reduction factor of this complex term?

• Given multiple terms (ANDed together), we simply multiply all the reduction factors
  – Total $RF = RF_{T_1} \times RF_{T_2} \times \ldots \times RF_{T_n}$
  – So: Result Cardinality = Cardinality * product of all RFs
  – How accurate is this?

Enumerating Plans for Multiple Joins

• What if the query has multiple joins?
• Should we generate as many plans as possible?
  – No … it typically takes too long / there are too many
  – It is best to generate a few, if we can find some cheap ones
• In “System R” only left-deep join trees are generated
Enumerating Plans for Multiple Joins

• Left-deep trees allow us to generate all of the “fully pipelined plans”
  – That is, intermediate results not written to temporary files
  – Not all left-deep tree can be fully pipelined (e.g., under Sort-Merge Join)

• Because picking only left-deep plans restricts the search space (of plans)

• … the optimizer may not find the optimal plan!

Enumerating Left-Deep Plans

• Considering all possible left-deep plans
  – For SPJ queries, these are enumerated by ordering tables

• For each ordering, consider the access method for each relation and the join method for each join

• (Greedy) Enumeration using N passes (if 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 (All 2-relation plans)
  – Pass N: Find best way to join result of (N – 1)-relation plan (as outer) to the N'th relation (All N-relation plans)

• At the end of each pass only retain
  – Cheapest overall plan + cheapest plan for each “interesting order” of tuples (e.g., for further joins, ORDER-BY, GROUP-BY, etc.)
**Nested Queries**

- Nested block is optimized independently
  - Outer tuple considered as providing a selection condition
- Outer block is optimized with the cost of “calling” the nested block
- Implicit ordering of these blocks means that some good strategies are not considered
- The non-nested version of the query is traditionally more optimized … so you may need to explicitly unnest the query

```sql
SELECT S.sname
FROM Sailors S
WHERE EXISTS
  (SELECT *
   FROM Reserves R
   WHERE R.bid = 103 AND R.sid = S.sid);
```

**Finding the Best Plan**

- Query optimizers don’t (always) find the best plan
  - There are usually more plans than you can consider (even if only left-deep plans are considered)
  - The optimizer might not even try to generate all of them
  - Sometimes the optimizer will compare the optimization cost to the estimated execution cost and quit early
  - The optimizer chooses the plan with the lowest ESTIMATED cost … actual costs may differ

Optimizers and optimization techniques are fairly sophisticated

- We have only covered the “traditional” approaches (that stemmed from System R)
Next time

• Physical database design
• Then transactions and recovery