administrivia

- midterm this Thursday 10/17

- assignments
  - lab 2 ends tomorrow night, don’t forget write up!
  - lab 3 starts after fall break
  - no problem set out this week

goals for today

- reason about the stages of query optimization

- understand how to estimate the cost of a full query plan
  - pipelining vs. materialization
  - intermediate result sizes

Cost-based query sub-system

Ideally: find the best query plan
Reality: avoid the worst plans!
Query Optimization Overview

- Query converted to relational algebra expression
- Relational algebra converted to tree, joins as branches
- ** Operators can also be applied in different order! **

```
SELECT S.sname
FROM Reserves R, Sailors S
WHERE R.sid=S.sid
AND R.bid=100
AND S.rating>5
```

Each operator has implementation choices → Choosing forms physical plan

Query Optimizer algorithm

- **Goal: given a a query**, the optimizer wants to
  - Decide which query plans to consider
  - Compare plans and choose the “best” one
    (best = shortest time to run)

- **How about this algorithm?**
  - Step 1: enumerate the space of all possible plans
  - Step 2: run each query plan, measure its runtime
  - Step 3: choose the plan that ran the fastest!

Query Optimizer algorithm

- **Goal: given a a query**, the optimizer wants to
  - Decide which query plans to consider
  - Compare plans and choose the “best” one
    (best = shortest time to run)

- **Actual algorithm**
  - Step 1: consider a set of possible plans
  - Step 2: estimate cost for each plan
  - Step 3: choose the plan with lowest cost

Estimating Cost

- Don’t want to execute a plan to figure out its run-time!
  - Instead estimate cost of the plan
  - Use cost as a proxy for run-time

- Cost of a plan = sum of costs for each operator in plan
Exercise 2: Reasoning about cost

- Assume:
  - Each relation is 5 pages and stored as a heap file, no indexes
  - Buffer pool has 4 frames
  - Join algorithm is page-nested-loop-join (PNLJ)
  - Order by operator uses general external merge-sort

1. (Review) What is the cost in I/Os for this plan, ignoring cost of final output?

   \[ 5 + 5 \times 5 \]

2. Now what about the cost of this plan? What information are you missing?

   ORDER BY(A.foo)

   Need Input size to ORDER BY, determined by output size of JOIN

Pipelined vs. Materialized

- Each query plan operator’s output could be generated in either materialized or pipelined fashion

  - Materialized
    - Complete output of an operator saved (typically written back to disk) as a temporary relation before its parent reads it in

  - Pipelining (“on-the-fly”)
    - Parts of output of operator immediately given to parent as input

Pipelining

- Parent and child operators executing concurrently
  - Iterator model
  - Parent calls next() on child/children
  - (As needed) child calls next() on its child/children

- Savings compared to materialization to disk
  - No write I/O cost for child’s output
  - No read I/O cost for parent’s input

- Operator algorithm(s) must support pipelining for this to work!

Exercise 3: Pipelining

- Use Page-Nested-Loop joins for the join algorithm

- Some examples:
  - (A join B) join C
    - Pipelined
  - C join (A join B)
    - Since (A join B) is the inner relation for the second join, need to materialize it
Motivating Example

- Suppose there are
  - 100 boats (uniformly distributed)
  - 10 ratings (uniformly distributed 1-10)
- Cost: 500 + 500*1000 I/Os
- Misses several opportunities:
  - Selections could have been “pushed” earlier
  - No use is made of any available indexes...
- Goal of optimization: find more efficient plans that compute the same answer.

SELECT S.sname
FROM Reserves R, Sailors S
WHERE R.sid=S.sid
  AND R.bid=100
  AND S.rating>5

Alternative Plans – Push SELECTs
(No Indexes)

Sailors
(500 pages)
Reserves
(1000 pages)

Alternative Plans – Push SELECTs
(No Indexes)

Sailors
(500 pages)
Reserves
(1000 pages)

Exercise 4-5: Estimate I/O cost

- Sailors
  - 250,500 I/Os
  - 500 + 1000 * 10 + (250 * 10)

- Reserves
  - 6000 I/Os
  - 1000 + 500 * 250 + (10 * 250)

- Sailors
  - 4250 I/Os
  - 500 + 1000 + 100 + 10000 + (250 * 10)
Exercise 5: Estimate I/O cost

Alternative Plans: Indexes

Suppose these indexes exist:
- Clustered Alt 1 hash index on bid of Reserves
- Unclustered Alt 2 hash index on sid of Sailors

Accessing Reserves, bid=100:
- Get 100,000/100 boats = 1000 records → 1000/100=10 pages

Cost: Selection on Reserves (10 I/Os); then, for each tuple, get [one] matching Sailors tuple:
1000 tuples * (1.2+1) = 2210 I/Os

Join column sid is a key for Sailors!

Query Blocks: Units of Optimization

- An SQL query is parsed into a set of query blocks, and these are optimized one block at a time
- Inner blocks are usually treated as subroutines
- Computed:
  - once per query (for uncorrelated sub-queries)
  - or once per outer tuple (for correlated sub-queries)

The System R aka “Selinger-style” Query Optimizer

Impact:
- Inspired most optimizers in use today
- Works well for small-medium complexity queries (< 10 joins)

Cost estimation:
- Very inexact, but works ok in practice.
- Statistics, maintained in system catalogs, used to estimate cost of operations and result sizes.
- Considers a simple combination of CPU and I/O costs.

Plan Space: Too large, must be pruned!
Statistics and cardinality estimation

- **Catalogs** typically contain at least:
  - Number of tuples ($N_{\text{tuples}}$) and number of pages ($N_{\text{pages}}$) per relation
  and for each index:
  - Number of distinct key values ($N_{\text{keys}}$)
  - Low/high key values ($\text{Low/High}$)
  - Index height ($\text{Height}$) for each tree index.
  - Index size ($N_{\text{pages}}$) (e.g., # leaf pages for tree)

- Statistics in catalogs updated periodically.
  - Updating whenever data changes is too expensive; lots of approximation anyway, so slight inconsistency ok.

Size Estimation and Reduction Factors

- Consider a query block: `SELECT attribute list FROM relation list WHERE term1 AND ... AND termk`

  - Reduction factor (RF) associated with each term reflects the impact of the term in reducing result size
  - RF is also called “selectivity”

  - How to predict size of output?
    - Need to know/estimate input size
    - Need to know/estimate RFs
    - Need to know/assume how terms are related

Result Size Estimation for Selections

- Result cardinality (for conjunctive terms) = 
  \[ \text{# input tuples} \times \text{product of all RF's} \]

Assumptions:
1. Values are uniformly distributed
   and terms are independent!
2. In System R, stats only tracked for indexed attributes
   (modern systems have removed this restriction)

<table>
<thead>
<tr>
<th>Term</th>
<th>Reduction Factor</th>
</tr>
</thead>
<tbody>
<tr>
<td>col = value</td>
<td>$1 / N_{\text{keys}(l)}$</td>
</tr>
<tr>
<td>col &gt; value</td>
<td>$(\text{High}(l)-\text{value}) / (\text{High}(l)-\text{Low}(l))$</td>
</tr>
</tbody>
</table>

Note: in System R, if missing indexes, assume RF = 1/10

Exercise 6

- RF = $16/40 \times 1/10 = 1/25$
  - Result size: 20 pages or 1600 tuples