Relational Query Optimization
Chapters 13 and 14

Overview of Query Optimization
- Plan: Tree of RA ops, with choice of alg for each op.
  - Each operator typically implemented using a 'pull' interface: when an operator is 'pulled' for the next output tuples, it 'pulls' on its inputs and computes them.
  - Two main issues:
    - For a given query, what plans are considered?
      - Algorithm to search plan space for cheapest (estimated) plan.
    - How is the cost of a plan estimated?
  - Ideally: Want to find best plan. Practically: Avoid worst plans!
  - We will study the System R approach.

Highlights of System R Optimizer
- Impact:
  - Most widely used currently; works well for < 10 joins.
- 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.
  - Only the space of leaf-leaf plans is considered.
  - Leaf plans allow output of one operator to be merged into the next operator without shuffling if in a temporary relation.
  - Cartesian products avoided.

Schema for Examples
Sailors (s_id integer, surname string, rating integer, age real)
Reserves (r_id integer, rid integer, age dates, rname string)
- Similar to old schema; name added for variations.
- Reserves:
  - Each tuple is 40 bytes long, 100 tuples per page, 100 pages.
- Sailors:
  - Each tuple is 50 bytes long, 80 tuples per page, 500 pages.

Motivating Example
SELECT Surname
FROM Reserves R, Sailors S
WHERE R.id=S.id AND R.flag=100 AND Rating>5
- Cost: 1000+2000 I/Os
- By no means the worst plan!
- Misses several opportunities, selections could have been 'pushed' earlier, no use made of any available indexes, etc.
- Cost of optimization: To find more efficient plans that compute the same answer.

Alternative Plans 1 (No Indexes)
- Main difference: push selects.
  - With 5 buffers, cost of plan:
    - Scan Reserves (1000) + write temp T1 (10 pages, if we have 100 boats, uniform distribution).
    - Scan Sailors (1000) + write temp T2 (250 pages, if we have 10 ratings).
    - Sort T1 (cost 210), sort T2 (cost 230), merge (10-250).
    - Total: 2650 pages, 140.
  - If we used BNL join, join cost = 10+4*250, total cost = 2770.
  - If we 'push' projections, T1 has only 45, T2 only 45 and surname:
    - T1 fits in 3 pages, cost of BNL drops to under 250 pages, total < 200.
Alternative Plans 2 With Indexes

- With clustered index on bid of Reserves, we get 1000 tuples on 100/100 = 10 pages.
- INL with 
  \[ \text{Indexing (outer is not materialized).} \]
  - Projecting unnecessary fields from outer doesn't help.
- Join column sid is a key for Sailors.
- Almost one matching tuple, understand index on sid OK.
- Decision not to push \( \text{n} \) before the join is based on availability of sid index on Sailors.
- Cost: Selection of Reserves tuples \( (0.1/0.01) \); for each, must get matching Sailors tuple \( (0.01/0.01) \); total \( 120/01/0.1 \).

Query Blocks: Units of Optimization

- An SQL query is parsed into a collection of query blocks, and these are optimized one block at a time.
- Nested blocks are usually treated as calls to a subroutine, made once per outer tuple. (This is an oversimplification, but serves for now.)
- For each block, plans considered are:
  - All available access methods for each reln in FROM clause.
  - All left join trees (i.e., all ways to join the relations one-at-a-time, with the inner reln in the FROM clause, considering all reln permutations and join methods)

Cost Estimation

- For each plan considered, must estimate cost:
  - Must estimate cost of each operation in plan tree.
  - Depend on input cardinalities.
  - We've already discussed how to estimate the cost of operations (sequential scan, index scan, joins, etc.)
  - Must estimate size of result for each operation in tree.
  - Use information about the input relations.
  - For selections and joins, assume independence of predicates.
  - We'll discuss the System R cost estimation approach.
  - Very inexact, but works ok in practice.
  - More sophisticated techniques known now.

Statistics and Catalogs

- Need information about the relations and indexes involved. Catalogs typically contain at least:
  - \# tuples (NTuples) and \# pages (NPages) for each relation.
  - \# distinct key values (NKays) and NPages for each index.
  - Index height, low/high key values (Low/High) for each tree index.
- Catalogs updated periodically.
  - Updating whenever data changes is too expensive, lots of approximation anyway, so slight inexactness ok.
  - More detailed information (e.g., histograms of the values in some field) are sometimes stored.

Size Estimation and Reduction Factors

- Consider a query block \( \text{SELECT attribute list FROM relation list} \).
- Maximum \# tuples in result is the product of the cardinalities of relations in the FROM clause.
- Reduction factor (RF) associated with each term reflects the impact of the term in reducing result size. Result cardinality = Max \# tuples * product of all RF's.
  - Implicit assumption that terms are independent.
  - Term \( \text{of} \) value has RF \( (\text{range size}) \) given index. I on col.
  - Term \( \text{of} \) value has RF \( (\text{index size}) \).
  - Term \( \text{of} \) value has RF \( (\text{height} * \text{width}) \).

Relational Algebra Equivalences

- Allow us to choose different join orders and to 'push' selections and projections ahead of joins.
- \( \text{Selections:} \) \( \sigma_{c_1, \ldots, c_n}(R) = \sigma_{c_1}(\ldots \sigma_{c_n}(R)) \) (Cascade)
  \( \sigma_{c_1}(\sigma_{c_2}(R)) = \sigma_{c_2}(\sigma_{c_1}(R)) \) (Commute)
- \( \text{Projections:} \) \( \pi_{a_1}(R) = \pi_{a_1}(\ldots (\pi_{a_n}(R))) \) (Cascade)
  \( \text{Associative:} \) \( (R S) T = (R (S T)) \) (Associative)
  \( (R S) = (S R) \) (Commute)

Show that \( R (S T) = (T R) S \)
More Equivalences

- A projection commutes with a selection that only uses attributes retained by the projection.
- Selection between attributes of the two arguments of a cross-product converts cross-product to a join.
- A selection on just attributes of R commutes with R \( \bowtie \) S (i.e., \( \sigma (R \bowtie S) = \sigma (R) \bowtie S \))
- Similarly, if a projection follows a join R \( \bowtie \) S, we can "push it" by retaining only attributes of R (and S) that are needed for the join or are kept by the projection.

Enumeration of Alternative Plans

- There are two main cases:
  - Single-table plans
  - Multiple-table plans
- For queries over a single relation, queries consist of a combination of select, project, and aggregate ops:
  - Each available access path (file scan / index) is considered, and the one with the least estimated cost is chosen.
  - The different operations are essentially carried out together (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 Estimates for Single-Relation Plans

- Index: on primary key matches selection:
  - Cost is Height(D) + 1 for a B+-tree, about 1.2 for hash index.
- Clustered index: matching one or more selects:
  - \( N\text{Page}(D) \times N\text{Page}(R) \) * product of RFs of matching selects.
- Non-clustered index: matching one or more selects:
  - \( N\text{Page}(D) \times N\text{Page}(R) \) * product of RFs of matching selects.
- Sequential scan of file:
  - N\text{Page}(R).
- Note: Typically, no duplicate elimination on projections!
  (Exception: Done on result if user says DISTINCT.)

Example

```
SELECT Said
FROM Sailors S
WHERE Rating=8
```

- If we have an index on rating:
  - \( 1/\text{N\text{Keys}(R)} \times \text{N\text{Tuples}(R)} = 1/10 \times 4000 \) tuples retrieved.
  - Clustered index: \( 1/\text{N\text{Keys}(R)} \times \text{N\text{Page}(R)} \times \text{N\text{Tuples}(R)} = 1/10 \times 50 \times 50 \) pages are retrieved. (This is the cost.)
  - Unclustered index: \( 1/\text{N\text{Keys}(R)} \times \text{N\text{Page}(R)} \times \text{N\text{Tuples}(R)} = 1/10 \times 50 \times 4000 \) pages are retrieved.
- If we have an index on sid:
  - Would have to retrieve all tuples/pages. With a clustered index, the cost is 30\*50, with unclustered index, 50\*4000.
- Doing a file scan:
  - We retrieve all file pages (500).

Queries Over Multiple Relations

- Fundamental decision in System R: **only left-deep join trees** are considered:
  - As the number of joins increases, the number of alternative plans grows rapidly; we need to restrict the search space.
  - Left-deep trees allow us to generate fully pipelined plans.
  - Intermediate results are not written to temporary files.
  - Not all left-deep trees are fully pipelined (e.g., SM 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.
- Enumerated 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 a (N-1)-relation plan (as outer) to the N^th relation. (**All N-relation plans**)
- For each subset of relations, retain only:
  - Cheapest plan overall, plus
  - Cheapest plan for each interesting order of the tuples.
**Enumeration of Plans (Contd.)**

- ORDER BY, GROUP BY, aggregates etc handled as a final step, using either an ‘interestingly ordered’ plan or an additional sorting operator.
- An N-1 way plan is not combined with an additional relation unless there is a join condition between them, unless all predicates in WHERE have been used up.
  - i.e., avoid Cartesian products if possible.
- In spite of pruning plan space, this approach is still exponential in the # of tables.

**Example**

- **Pass 1**
  - **Sailors**: B: true on rating
  - **Reserves**: B: true on bid

**Nested Queries**

- Nested block is optimized independently, with the outer tuple considered as providing a selection condition.
- Outer block is optimized with the cost of ‘calling’ nested block computation taken into account.
- Implicit ordering of these blocks means that some good strategies are not considered. The non-nested version of the query is typically optimal better.

```
SELECT Surname
FROM Sailors S
WHERE EXISTS
(SELECT *
FROM Reserves R
WHERE R.bid=Rid AND S.bid=Said)
```

**Summary**

- Query optimization is an important task in a relational DBMS.
  - Must understand optimization in order to understand the performance impact of a given database design (relations, indexes) on a workload (set of queries).
  - Two parts to optimizing a query:
    - Consider a set of alternative plans.
    - Must prune search space, typically, left-deep plans only.
    - Must estimate cost of each plan that is considered.
    - Must estimate cost of result and cost for each plan node.
    - Key Issues: Statistics, indexes, operator implementations.

**Summary (Contd.)**

- Single relation queries:
  - All access paths considered, cheapest is chosen.
- Multiple relation queries:
  - All single relation plans are first enumerated.
  - Selections/projections considered as early as possible.
  - Next, for each 1-relation plan, all ways of joining another relation (as inner) are considered.
  - Next, for each 2-relation plan that is ‘retained’, all ways of joining another relation (as inner) are considered, etc.
  - At each level, for each subset of relations, only best plan for each interesting order of tuples is ‘retained’.