Relational Query Optimization

Chapters 13 and 14

Overview of Query Optimization

Plan: Tree of R.A. 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 space of left-deep plans is considered.
  - Left-deep plans allow output of each operator to be materialized into a temporary relation.
  - Cartesian products are avoided.

Schema for Examples

Sailors (id integer, name string, rating integer, age: real)
Reserves (bid integer, bid integer, day: dates, name: string)

- Similar to old schema, name added for variations.
- Reserves:
  - Each tuple is 40 bytes long, 100 tuples per page, 1000 pages.
- Sailors:
  - Each tuple is 30 bytes long, 80 tuples per page, 500 pages.

Motivating Example

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

- Cost: 300+5001+1001/Os
- By no means the worst plan!
- Plan: `πname(On-the-fly)`
- Misses several opportunities: selections could have been 'pushed' earlier, no use is made of any available indexes, etc.
- Goal of optimization: To find more efficient plans that compute the same answer.

Alternative Plans 1

(No Indexes)

- Main difference: `push select`
- With 5 buffers, cost of plan:
  - `Scan Reserves (1000) + write temp T1` (10 pages, if we have 100 boats, uniform distribution).
  - `Scan Sailors (800) + write temp T2` (250 pages, if we have 10 ratings).
  - Sort T1 (2*2^10), sort T2 (2*2^20), merge (10*250)
  - Total: 3500 page I/Os.
- If we used BNL joins, join cost = 10*4*250, total cost = 2270.
- If we 'push' projections, T1 has only sid, T2 only sid and `name`:
  - T1 fits in 3 pages, cost of BNL drops to under 250 pages, total < 2000.
**Alternative Plans 2: With Indexes**

- With clustered index on bid of Reserves, we get 100 000/100 = 1000 tuples on 100/100 = 10 pages.
- INL with pipelining (outer is not materialized):
  - Projecting out unnecessary fields from outer doesn’t help.
  - Join column sid is a key for Sailors.
  - It is true in matching tuples, unchanged index on sid OK.
  - Decision not to push joining before the join is based on availability of sid index on Sailors.
- Cost: Selection of Reserve tuples (101/O): for each, must get matching Sailors tuple (100!1/2); total 12101/Os.

**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 over-simplification, but serves for now.)
- For each block, the plans considered are:
  - All available access methods, for each reln in FROM clause.
  - All left-join 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.
  - Depends 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 inelegant, 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 (NKeys) 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 inconsistency 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.
- 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 col=value has RF 1/NKeys(i), given index on col.
  - Term col1=col2 has RF 1/NKeys(i2), NKeys(i1).
  - Term col=value has RF (High(i)-Low(i))/High(i)-Low(i)

**Relational Algebra Equivalences**

- Allow us to choose different join orders and to ‘push’ selections and projections ahead of joins.
  - **Selections:** $\sigma_{C_{1},...,C_{n}}(R) = \sigma_{C_{1}}(\ldots \sigma_{C_{n}}(R))$ (Cascading)
  - $\sigma_{C_{1}}(\sigma_{C_{2}}(R)) = \sigma_{C_{2}}(\sigma_{C_{1}}(R))$ (Commuting)
  - **Projections:** $\pi_{A}(R) = \pi_{A}(\ldots (\pi_{A}(R)))$ (Cascading)
  - $\pi_{A}(\pi_{B}(R)) = \pi_{A}(\pi_{B}(R))$ (Commuting)
- Show that $R (S T) \equiv (R (S) T)$ (Associative)
  - $(R S) \equiv (S R)$ (Commuting)
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 a relation $R$ commutes with $R \bowtie S$. (i.e., $\sigma(R \bowtie S) \equiv \sigma(R) \bowtie \sigma(S)$)
- Similarly, if a join 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-relation plans
  - Multiple-relation plans
- For queries over a single relation, queries consist of a combination of selects, projects, 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 I on primary key matches selection:
  - Cost is Height($I$)+1 for a B+ tree, about 1.2 for hash index.
- Clustered index I matching one or more selects:
  - $(NPages(I)+NPages(\pi)) \cdot \text{product of RFs of matching selects}.$
- Non-clustered index I matching one or more selects:
  - $(NPages(I)+NPages(\pi)) \cdot \text{product of RFs of matching selects}.$
- Sequential scan of file:
  - $NPages(I)$.
- Note: Typically, no duplicate elimination on projections!
  (Exception: Done on answers if user says DISTINCT.)

Example

\[
\text{SELECT S.sid FROM Sailors S WHERE S.rating = 8}
\]

- If we have an index on rating:
  - $(1/\text{NKeys}(I)) \cdot \text{NRows}(\pi) = (1/10) \cdot 4000$ tuples retrieved.
  - Clustered index: $(1/\text{NKeys}(I)) \cdot (NPages(\pi)+NPages(\pi)) = (1/10) \cdot (50+300)$ pages are retrieved. (This is the cost.)
  - Unclustered index: $(1/\text{NKeys}(I)) \cdot (NPages(\pi)+NPages(\pi)) = (1/10) \cdot (30+4000)$ pages are retrieved.
- If we have an index on sid:
  - Would have to retrieve all tuples/pages. With a clustered index, the cost is $50+500$, with unclustered index, $30+4000$.
- 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 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 sort 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

- Sailors: B+ tree matches rating > 5, and is probably cheaper. However, if this selection is expected to retrieve a lot of tuples, and index is uncluttered, file scan may be cheaper.
- Reserves: B+ tree on bid matches bid = 500, cheapest.
- Pass 2:
  - We consider each plan returned from Pass 1 as the outer, and consider how to join it with the (only) other relation.
  - e.g., Reserves is outer. Hash indices can be used to get Sailors tuples that satisfy old = outer tuple's old value.

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 n-th nested version of the query is typically optimized better.

<table>
<thead>
<tr>
<th>Nested block to optimize:</th>
<th>SELECT * FROM Reserves R WHERE R.bid=103 AND S.bid=S.id</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Nested block to optimize:</th>
<th>SELECT S.sname FROM Sailors S WHERE EXISTS (SELECT * FROM Reserves R WHERE R.bid=103 AND S.bid=R.sid)</th>
</tr>
</thead>
</table>

Example plan enumeration query:

- SELECT S.sname FROM Sailors S, Reserves R WHERE S.sid=R.sid AND R.bid=103

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 such space pricilly, left-deep plans only.
- Must estimate cost of each plan that is considered.
- Must estimate time of result and cost of each plan node.
- Key issues: Statistics, indexes, operator implementations.

Summary (Contd.)

- Single relation queries:
  - All access paths considered, cheapest is chosen.
  - Issues: Selection that match index, whether index key has all needed fields and/or provides tuples in a desired order.
- 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'.

Database Management Systems, R. Ramakrishnan and J. Gehrke