Relational Query Optimization

Chapter 15
Overview of Query Optimization

- **Goal of optimization:** To find more efficient plans that compute the same answer for a given query.
- **Ideally:** Want to find best plan.  
  **Practically:** Avoid worst plans!
Schema for Examples

Sailors (sid: integer, sname: string, rating: integer, age: real)
Reserves (sid: integer, bid: integer, day: dates, rname: string)

- Reserves:
  - Each tuple is 40 bytes long, 100 tuples per page, 1000 pages.
- Sailors:
  - Each tuple is 50 bytes long, 80 tuples per page, 500 pages.
**Motivating Example**

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

\[ \pi_{sname}(\sigma_{bid=100 \land rating>5}(\text{Reserves} \bowtie_{sid=sid} \text{Sailors})) \]

Query Expressed as a Relational Algebra Tree:
Motivating Example

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

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

Plan 3:
Relational Algebra Equivalences

- Allow us to choose different join orders and to `push' selections and projections ahead of joins.

- **Selections:** \( \sigma_{c_1 \land \ldots \land c_n}(R) \equiv \sigma_{c_1}(\ldots \sigma_{c_n}(R)) \) (Cascade)
  \[
  \sigma_{c_1}(\sigma_{c_2}(R)) \equiv \sigma_{c_2}(\sigma_{c_1}(R)) \] (Commutative)

- **Projections:** \( \pi_{a_1}(R) \equiv \pi_{a_1}(\ldots(\pi_{a_n}(R))) \) (Cascade)

- **Joins:** \( R \bowtie (S \bowtie T) \equiv (R \bowtie S) \bowtie T \) (Associative)
  \[
  (R \bowtie S) \equiv (S \bowtie R) \] (Commutative)
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 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 also estimate size of result for each operation in tree!
    • Use information about the input relations.
    • For selections and joins, assume independence of predicates.
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.
Cost Estimates for Single-Relation Plans

- Sequential scan of file:
  - \( N\text{Pages}(R). \)

- \textbf{Reduction factor (RF)} associated with each \textit{term} reflects the impact of the \textit{term} in reducing result size.

- Index I on primary key matches selection:
  - Cost is \( \text{Height}(I)+1 \) for a B+ tree, about 1.2 for hash index.

- Clustered index I matching one or more selects:
  - \( (N\text{Pages}(I)+N\text{Pages}(R)) \times \text{product of RF’s of matching selects}. \)

- Non-clustered index I matching one or more selects:
  - \( (N\text{Pages}(I)+N\text{Tuples}(R)) \times \text{product of RF’s of matching selects}. \)
Example

- Doing a file scan:
  - We retrieve all file pages (500).

- If we have an index on rating:
  - \( \left(\frac{1}{N\text{Keys}(I)}\right) \times \text{NTuples}(R) = \left(\frac{1}{10}\right) \times 40000 \) tuples retrieved.
  - Clustered index: \( \left(\frac{1}{N\text{Keys}(I)}\right) \times (\text{NPages}(I) + \text{NPages}(R)) = \left(\frac{1}{10}\right) \times (50+500) \) pages are retrieved. (This is the cost.)
  - Unclustered index: \( \left(\frac{1}{N\text{Keys}(I)}\right) \times (\text{NPages}(I) + \text{NTuples}(R)) = \left(\frac{1}{10}\right) \times (50+40000) \) 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, 50+40000.

```
SELECT S.sid
FROM Sailors S
WHERE S.rating=8
```
A Query Tree Illustrating Pipelining

- When a query is composed of several operators, the result of one operator is sometimes pipelined to another operator without creating a temporary relation to hold the intermediate result.

Result tuples of first join pipelined into join with C
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 all 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.
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.
Cost Estimation for Multirelation Plans

- Consider a query block:

  ```sql
  SELECT attribute list
  FROM relation list
  WHERE term1 AND ... AND termk
  ```

- 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.

- Multirelation plans are built up by joining one new relation at a time.
  - Cost of join method, plus estimation of join cardinality gives us both cost estimate and result size estimate
Example

- **Pass 1:**
  - **Sailors:** B+ tree matches *rating > 5*, and is probably cheapest. However, if this selection is expected to retrieve a lot of tuples, and index is unclustered, file scan may be cheaper.
    - Still, B+ tree plan kept (because tuples are in *rating* order).
  - **Reserves:** B+ tree on *bid* matches *bid = 100*; cheapest.

- **Pass 2:**
  - We consider each plan retained from Pass 1 as the outer, and consider how to join it with the (only) other relation.
    - e.g., **Reserves as outer:** Hash index can be used to get Sailors tuples that satisfy *sid = outer tuple’s sid* 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 non-nested version of the query is typically optimized better.

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

Nested block to optimize:
```
SELECT *
FROM Reserves R
WHERE R.bid=103
AND S.sid= outer value
```

Equivalent non-nested 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 search space; typically, left-deep plans only.
  - Must estimate cost of each plan that is considered.
    - Must estimate size 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.
  - Issues: Selections 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’.