Introduction to Query Optimization

Chapter 13

Overview of Query Optimization

- **Query Plan**: Tree of R.A. operations, with choice of algorithm for each operation.
  - `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 subspace of *left-deep plans* is considered.
    - Left-deep plans allow output of each operator to be *pipelined* into the next operator without storing it in a temporary relation.
    - Cartesian products avoided.

---

**Left-Deep Query Plans**

- In a *linear query plan*, at least one child of a join node is a base relation.

- A *left-deep query plan* is a linear plan where the right child of a join node is a base relation.

- Left-deep trees can generate all *fully pipelined* plans.
  - Intermediate results not written to temporary files.

![Diagram of Query Plans](image)
Schema for Examples

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

- Similar to old schema; \(rname\) added for variations.
- 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

\[
\text{SELECT } S.\text{sname} \\
\text{FROM Reserves R, Sailors S} \\
\text{WHERE R.sid=S.sid AND} \\
\text{R.bid=100 AND S.rating}>5
\]

RA Tree: \(\text{Join} \quad \text{Join} \quad \text{Project} \quad \text{Project} \quad \text{Join} \quad \text{Project} \)

Query Plan: \(\text{Join} \quad \text{Join} \quad \text{Project} \quad \text{Project} \quad \text{Join} \quad \text{Project} \)
Motivating Example (contd.)

- Cost: $500+500 \times 1000$ I/Os
- By no means the worst plan!
- 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

- 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 (500) + write temp T2 (250 pages, if we have 10 ratings).
  - Sort T1 ($2 \times 2 \times 10$), sort T2 ($2 \times 3 \times 250$), merge ($10+250$)
  - Total: 3560 page I/Os.
- If we used BNL join, join cost = 10 + 4 \times 250, total cost = 2770.
- If we `push’ projections, T1 has only sid, T2 only sid and sname:
  - T1 fits in 3 pages, cost of BNL drops to under 250 pages, total < 2000.
Alternative Plans 2

- With clustered index on bid of Reserves, we get $100,000 / 100 = 1000$ tuples on $1000 / 100 = 10$ pages.
- INL with pipelining (outer is not materialized).
- Join column sid is a key for Sailors.
  - At most one matching tuple, unclustered index on sid is o.k.
- Decision not to push rating>5 before the join is based on availability of sid index on Sailors.
- Cost: Selection of Reserves tuples (10 I/Os); for each tuple, must get matching Sailors tuple (1000*1.2); total 1210 I/Os.

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 the relational operations.
  - Must estimate size of result for each operation in tree!
    - Often, assume uniform distribution of attribute values.
    - For selections and joins, assume independence of predicates.
- We’ll discuss the System R cost estimation approach.
  - Very inexact, but works o.k. 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 o.k.

- More detailed information (e.g., histograms of the values in some field) are sometimes stored.

Size Estimation and Reduction Factors

- Consider a query block:

  \[
  \text{SELECT attribute list FROM relation list WHERE term1 AND ... AND termk}
  \]

- 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.
  - Implicit assumption that terms are independent!
  - Term \( col=value \) has RF \( 1/NKeys(I) \), for index I on \( col \).
  - Term \( col1=col2 \) has RF \( 1/\text{MAX}(NKeys(I1), NKeys(I2)) \).
  - Term \( col>value \) has RF \( (\text{High}(I)-value)/(\text{High}(I)-\text{Low}(I)) \).
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.