Principles of Data Management

Lecture #9
Query Processing Overview

Instructor: Chen Li
DBMS Structure (from Lecture 1)

- Query Parser
- Query Optimizer
- Plan Executor
- Relational Operators (+ Utilities)
- Files of Records
- Access Methods (Indices)
- Buffer Manager
- Disk Space and I/O Manager
- Transaction Manager
- Lock Manager
- Log Manager
- WAL

SQL

You are now here...
Component Roles

- **Query Parser**
  - Parse and analyze SQL query
  - Produce data structure capturing SQL statement and the “objects” that it refers to in the system catalogs

- **Query optimizer (often w/2 steps)**
  - Rewrite query logically (e.g., views/unnesting)
  - Perform cost-based optimization (*a la System R*)
  - Goal is to pick a “good” query plan considering
    - Physical table structures
    - Available access paths (indexes)
    - Data statistics (if known)
    - Cost model (for relational operations)

(Cost differences can be orders of magnitude!)
Component Roles (cont.)

- Plan Executor + Relational Operators
  - Runtime side of query processing
  - Usually based on a pull-based “tree of iterators” model, e.g.:

  - Nodes are relational operators (actually physical implementations of relational operators/combos)
Some Common Techniques

- Algorithms for evaluating relational operators all use some simple ideas extensively*:
  - **Indexing:** Can use WHERE conditions to retrieve small set of tuples (selections, joins)
  - **Iteration:** Sometimes faster to scan all tuples even if there’s an index. (And sometimes, we can scan the data entries in an index instead of the table itself.)
  - **Partitioning:** By using sorting or hashing, we can partition the input tuples and replace an expensive operation by similar operations on smaller inputs.

*Watch for these techniques as we discuss query evaluation!
(Schema and Data for Examples)

Sailors $(sid: \text{integer}, sname: \text{string}, rating: \text{integer}, age: \text{real})$

Reserves $(sid: \text{integer}, bid: \text{integer}, day: \text{date}, rname: \text{string})$

- **Sailors:**
  - Members of the UW-Madison Hoofers sailing club.
  - 40,000 tuples, each one 50 bytes long, with 80 tuples/page (4000 byte pages), so 500 pages total.

- **Reserves:**
  - Members’ reservations for the club’s sailboats.
  - 100,000 tuples, each one 40 bytes long, so 100 tuples/page (4000 byte pages), yielding 1000 pages total.
Access Paths

- An **access path** is a method of retrieving tuples:
  - File scan, or index that matches a selection (in the query)
- A tree index **matches** (a conjunction of) terms that involve only attributes in a **prefix** of the search key.
  - E.g., Tree index on \(<a, b, c>\) matches the selection \(a=5 \text{ AND } b=3\), and \(a=5 \text{ AND } b>6\), but not \(b=3\).
- A hash index **matches** (a conjunction of) terms that has a term **attribute = value** for every attribute in the search key of the index.
  - E.g., Hash index on \(<a, b, c>\) matches \(a=5 \text{ AND } b=3 \text{ AND } c=5\); but not \(b=3\), or \(a=5 \text{ AND } b=3\), or \(a>5 \text{ AND } b=3 \text{ AND } c=5\).
A Note on Complex Selections

Selection conditions are first converted to be in conjunctive normal form (CNF):

\[(\text{day}<8/9/94 \text{ AND } rname=\text{'Paul'}) \text{ OR } bid=5 \text{ OR } sid=3) \text{ AND } (rname=\text{'Paul'} \text{ OR } bid=5 \text{ OR } sid=3)\]

- We’ll mostly discuss cases with no ORs; see the text if you’re curious about the general case.
One Approach to Selections

- Find the *most selective access path*, retrieve tuples using it, then apply any remaining terms that don’t match the index:
  - *Most selective access path*: An index or file scan that we estimate will require the fewest page I/Os.
  - Terms that match this index reduce the number of tuples retrieved; other terms used to filter the retrieved tuples on the fly, but don’t prevent retrieval of the tuples/pages.
  - Consider \( \text{day} < 8/9/94 \ AND \ \text{bid} = 5 \ AND \ \text{sid} = 3 \). A B+ tree index on \( \text{day} \) can be used; \( \text{bid} = 5 \) and \( \text{sid} = 3 \) must be checked for each retrieved tuple. Similarly, a hash index on \( \langle \text{bid}, \ \text{sid} \rangle \) could be used; \( \text{day} < 8/9/94 \) must then be checked.
Using an Index for Selections

- Cost depends on #qualifying tuples and clustering.
  - Cost of finding qualifying data entries (typically small) plus cost of retrieving actual records themselves (can be large w/o clustering).
  - E.g., assuming uniform distribution of names, about 10% of tuples qualify (100 pages, 10,000 tuples). With a clustered index, cost will be just over 100 I/Os; if index is unclustered, can cost up to 10,000 I/Os!

```
SELECT * 
FROM Reserves R 
WHERE R.rname < 'C'
```
Duplicate Elimination

- Relational algebra “projection” removes duplicates.
  - SQL systems don’t remove duplicates unless the keyword DISTINCT is specified in a query. (Bags vs. sets.)
- Sorting Approach: Sort on <sid, bid> and remove duplicates. (Can optimize by dropping unwanted information while sorting.)
- Hashing Approach: Hash on <sid, bid> to create partitions. Load partitions into memory one at a time, build an in-memory hash structure, and eliminate duplicates within it. (Q: See why this works?)
- If there is an index with both R.sid and R.bid in the search key, may be cheaper to sort data entries!

```sql
SELECT DISTINCT R.sid, R.bid 
FROM Reserves R
```
Highlights of System R Optimizer

- **Impact:**
  - Most widely used approach; works well for < 10 joins.

- **Cost estimation:** Approximate cost 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, so must be pruned.
  - Only the space of *left-deep plans* is considered.
    - Left-deep plans allow the output of each operator to be pipelined into the next operator without storing it in a temporary relation.
  - Cartesian products always avoided.
Statistics and Catalogs

- Need information about the relations and indexes involved. **Catalogs** typically contain at least:
  - # tuples and # pages for each relation.
  - # distinct key values and # pages for each index.
  - Index height, low/high key values for each tree index.

- Catalogs updated periodically.
  - Updating each time data changes too expensive; lots of approximation anyway, so slight inconsistencies okay.

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

- For each plan considered, must estimate cost:
  - **Estimate cost** of each operation in plan tree.
    - Depends on input cardinalities.
    - We’ll cover how to estimate the cost of operations (sequential scan, index scan, joins, etc.)
  - **Estimate size of result** for each operation in tree!
    - Use information about the input relations.
    - For selections and joins, assume independence of predicates.
Size Estimation and Reduction Factors

- Consider a query block:

- Maximum # tuples in result is the product of the cardinalities of the 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!*
  - More on this later in the quarter.
(Example Schema and Data, Again)

Sailors \((sid: \text{integer}, \ sname: \text{string}, \ rating: \text{integer}, \ age: \text{real})\)

Reserves \((sid: \text{integer}, \ bid: \text{integer}, \ day: \text{date}, \ rname: \text{string})\)

- **Sailors:**
  - Members of the UW-Madison Hoofers sailing club.
  - 40,000 tuples, each one 50 bytes long, with 80 tuples/page (4000 byte pages), so 500 pages total.

- **Reserves:**
  - Members’ reservations for the club’s sailboats.
  - 100,000 tuples, each one 40 bytes long, so 100 tuples/page (4000 byte pages), yielding 1000 pages total.
A Motivating Example

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

- **Cost:** $500 + 500 \times 1000 = \textbf{500,500} \text{ I/Os}$
- *(Not the worst possible plan! 😊)*
- Misses several opportunities: selections could have been `pushed’ earlier, no use is made of any available indexes, etc.
- **Goal of optimization:** To consider more efficient plans that compute the same logical answer.
Alternative Plan 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), with cost = 1010.
  - Scan Sailors (500) + write temp T2 (250 pages, if 10 ratings), cost = 750.
  - Join by sort T1 (2*2*10 = 40), sort T2 (2*3*250 = 1500), merge (10+250).
  - Total: 3560 page I/Os.
- If we use block NL join, join cost = 10+10*250, total cost = 4270.
- If we “push” projections, T1 has only sid, T2 only sid and sname:
  - T1 now fits in 3 pages, so block NL cost drops lower, … (etc.)
Alternative Plan 2
(Using Indexes)

- With clustered index on bid of Reserves, we get 100,000/100 = 1000 tuples on 1000/100 = 10 pages.
- INL join with pipelining (outer not materialized).
  - (Projecting out unnecessary fields from outer doesn’t help.)
- Join column sid is a key for Sailors.
  - At most one matching Sailors tuple, so unclustered index on sid is OK.
- Decision not to push rating>5 before the join based on the availability of the sid index on Sailors.
- Cost: Selection of Reserves tuples (10 I/Os); then, for each one, must get matching Sailors tuple (1000*1.2); total = 1210 I/Os.
QP Summary

- There are several alternative evaluation algorithms for each relational operator.
- A query is evaluated by converting it to a tree of operators and then evaluating the operators in the tree.
- DBA must understand query optimization to fully understand the performance impact of a given database design (relations, indexes) on a workload (set of queries).
- Two parts to optimizing a query (coming later):
  - Enumerate a set of alternative plans.
    - Must prune search space; typically, left-deep plans only.
  - Estimate the cost of each plan that is considered.
    - Must estimate size of result and cost for each plan node.
    - Key issues: Statistics, indexes, operator implementations.