Query Optimization II

R&G, Chapters 12, 13, 14
Lecture 9

Administrivia

- **Homeworks 2 and 3 have been switched around.**
- **The new Hw2 will be due in ~2 weeks.**
- **Midterm in ~3 weeks.**

Review – the Big Picture

- **Data Modelling**
  - Relational
  - E-R
- **Storing Data**
  - File Indexes
  - Buffer Pool Management
- **Query Languages**
  - SQL
  - Relational Algebra
  - Relational Calculus
  - Formal Query Languages permit Query Optimization

Review – Query Optimization 1

- **External Sorting with Merge sort**
  - It is possible to sort most tables in 2 passes
- **Join Algorithms**
  - Nested Loops
  - Indexed Nested Loops
  - Sort-Merge Join
  - Hash Join

Review – Cost of Join Methods

- **Blocked Nested Loops**
  $M + \lceil M / B \rceil * N$
- **Indexed Nested Loops**
  $M + ((M * p_R) *$ cost to find matching tuples)
- **Sort-Merge Join**
  between 3(M+N) and M*N
- **Hash Join**
  3(M+N) and higher, especially with skewed data

Today: Finding a Better Query

- **Query Languages based on formal foundation**
- **Just as regular algebra expressions can be rewritten, so can relational algebra:**

  A*B = B*A, A(B + C) = AB + AC, etc.

  $A \cdot B = B \cdot C$, etc.

Relational Algebra Equivalences

- **Choose different join orders**
- **`push' selections and projections ahead of joins.**
- ***Selections:*** $\sigma_{a_1 \wedge \ldots \wedge a_n}(R) \equiv \sigma_{a_1}(\ldots \sigma_{a_n}(R))$ ***(Cascade)***

  $\sigma_{a_1}(\sigma_{a_2}(R)) \equiv \sigma_{a_2}(\sigma_{a_1}(R))$ *(Commute)*
- *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)*

  $(R \bowtie S) \equiv (S \bowtie R)$ *(Commute)*

Show that: $R \bowtie (S \bowtie T) \equiv (T \bowtie R) \bowtie S$

Optimization in a Nutshell

- **Consider access paths to source tables**
  - Tuple Scan
  - Indexes
  - Partitioning
- **Consider equivalent algebra formulas**
  - Estimate costs based on statistics in DBMS

Or, in more detail...

- ***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.**

Schema for Examples

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

- **Similar to old schema; *rname* added for variations.**
- **Reserves:**
  - Each tuple is 40 bytes long,
  - 100 tuples per page,
  - M = 1000 pages total.
- **Sailors:**
  - Each tuple is 50 bytes long,
  - 80 tuples per page,
  - N = 500 pages total.

Statistics and Catalogs

- **Need information about the relations and indexes involved.** *Catalogs* **typically contain at least:**
  - # tuples (NTuples), # 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 sometimes stored.**
  - e.g., histograms of the values in some fields

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 AND b=3*, and *a=5 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 AND b=3 AND c=5*; but it does not match *b=3*, or *a=5 AND b=3*, or *a>5 AND b=3 AND c=5*.

A Note on Complex Selections

(day<8/9/94 AND rname='Paul') OR bid=5 OR sid=3

- **Selection conditions are first converted to *conjunctive normal form* (CNF):**
  (day<8/9/94 OR bid=5 OR sid=3) AND (rname='Paul' OR bid=5 OR sid=3)
- **We only discuss case with no ORs; see text if you are curious about the general case.**

One Approach to Selections

- **Find the *most selective access path*,**
- **retrieve tuples using it, and**
- **apply any remaining terms that don't match index**
  - Index or file scan that estimate will need the fewest I/Os.
  - terms that match index reduce the number of tuples retrieved;
  - other terms discard already retrieved tuples, but do not affect I/Os
- **Consider *day<8/9/94 AND bid=5 AND sid=3* .**
  - A B+ tree index on *day* can be used; then, *bid=5* and *sid=3* must be checked for each retrieved tuple.
  - Similarly, a hash index on <*bid, sid*> could be used; *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 records (could be large w/o clustering).
  - In example, assuming uniform distribution of names, about 10% of tuples qualify (100 pages, 10000 tuples). With a clustered index, cost is little more than 100 I/Os; if unclustered, upto 10000 I/Os!

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

Projection

```
SELECT DISTINCT
       R.sid, R.bid
FROM   Reserves R
```

- The expensive part is removing duplicates.
  - SQL systems don't remove duplicates unless the DISTINCT is specified.
- 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 in-memory hash structure, and eliminate duplicates.
- If there is an index with both R.sid and R.bid in the search key, may be cheaper to sort data entries!

Join: Index Nested Loops

```
foreach tuple r in R do
    foreach tuple s in S where r_i == s_j do
        add <r, s> to result
```

- **If there is an index on the join column of one relation (say S), can make it the inner and exploit the index.**
  - Cost: M + ((M*p_R) * cost of finding matching S tuples)
- **For each R tuple, cost of probing S index is about 1.2 for hash index, 2-4 for B+ tree. Cost of then finding S tuples (assuming Alt. (2) or (3) for data entries) depends on clustering.**
  - Clustered index: 1 I/O (typical), unclustered: up to 1 I/O per matching S tuple.

Examples of Index Nested Loops

- **Hash-index (Alt. 2) on *sid* of Sailors (as inner):**
  - Scan Reserves: 1000 page I/Os, 100*1000 tuples.
  - For each Reserves tuple: 1.2 I/Os to get data entry in index, plus 1 I/O to get (the exactly one) matching Sailors tuple. Total: 220,000 I/Os.
- **Hash-index (Alt. 2) on *sid* of Reserves (as inner):**
  - Scan Sailors: 500 page I/Os, 80*500 tuples.
  - For each Sailors tuple: 1.2 I/Os to find index page with data entries, plus cost of retrieving matching Reserves tuples. Assuming uniform distribution, 2.5 reservations per sailor (100,000 / 40,000). Cost of retrieving them is 1 or 2.5 I/Os depending on whether the index is clustered.