Overview of Query Evaluation

Chapter 12
Sina Meraji
Query Evaluation

• How could we evaluate the following query?

\[- \pi_{\text{Date}} \left( \sigma_{R.\text{SID}=S.\text{SID} \text{ and Rating} = 10} (R \times S) \right)\]

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

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

- **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.
Overview of Query Evaluation

- **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 in query optimization:
  - 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.
Some Common Techniques

- Algorithms for evaluating relational operators 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 is 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!*
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.
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 it does not match \(b=3\), or \(a=5 \text{ AND } b=3\), or \(a>5 \text{ AND } b=3 \text{ AND } c=5\).
A Note on Complex Selections

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

- Selection conditions are first converted to **conjunctive normal form (CNF):**
  
  \[(\text{day}<8/9/94 \text{ OR } \text{bid}=5 \text{ OR } \text{sid}=3) \text{ AND } (\text{rname}='\text{Paul}' \text{ OR } \text{bid}=5 \text{ OR } \text{sid}=3)\]

- 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 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 are used to discard some retrieved tuples, but do not affect number of tuples/pages fetched.
  – 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 on rname, cost is little more than 100 I/Os; if unclustered, upto 10000 I/Os!

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

• The expensive part is removing duplicates.
  – SQL systems don’t remove duplicates unless the keyword DISTINCT is specified in a query.
• Sorting Approach: Sort on \(<sid, bid>\) and remove duplicates. (Can optimize this 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.
• Using Indexes: If there is an index with both R.sid and R.bid in the search key, may be cheaper to sort data entries!

```
SELECT DISTINCT R.sid, R.bid
FROM Reserves R
```
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: \( M + ( (M \cdot p_R) \cdot \text{cost of finding matching S tuples}) \)
  - \( M = \# \text{pages of R, } p_R = \# \text{ R tuples per page} \)

- 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 depends on clustering.
  - Clustered index: 1 I/O (typical), unclustered: upto 1 I/O per matching S tuple.
Examples of Index Nested Loops

• Hash-index on \textit{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: \((1+1.2)*100000=220,000\) I/Os.

• Hash-index on \textit{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.
Join: Sort-Merge ($R \Join_{i=j} S$)

- Sort R and S on the join column, then scan them to do a "merge" (on join col.), and output result tuples.
  - Advance scan of R until current R-tuple $\geq$ current S tuple, then advance scan of S until current S-tuple $\geq$ current R tuple; do this until current R tuple = current S tuple.
  - At this point, all R tuples with same value in $R_i$ (current $R$ group) and all S tuples with same value in $S_j$ (current $S$ group) match; output <r, s> for all pairs of such tuples.
  - Then resume scanning R and S.

- R is scanned once; each S group is scanned once per matching R tuple. (Multiple scans of an S group are likely to find needed pages in buffer.)
Example of Sort-Merge Join

- Cost: $M \log M + N \log N + (M+N)$
  - The cost of scanning, $M+N$
  - With 35, 100 or 300 buffer pages, both Reserves and Sailors can be sorted in 2 passes; total join cost: 7500, how?