Evaluation of Relational Operations

Yanlei Diao

Slides Courtesy of R. Ramakrishnan and J. Gehrke
Overview of Query Processing

- Query Parser
- Query Rewriter
- Query Optimizer
- Query Executor

- Lock Manager
- Access Methods
- Buffer Manager
- Log Manager

- Disk Space Manager
- Disk Manager

- Query Processor
  
  Transactional Storage Manager
Relational Operations

- We will consider how to implement
  - **Selection** $(\sigma)$ Selects a subset of rows from relation.
  - **Join** $(\bowtie)$ Allows us to combine two relations.
  - **Projection** $(\pi)$ Deletes unwanted columns from relation.
  - **Union** $(\cup)$ Tuples in either reln. 1 or reln. 2.
  - **Intersection** $(\cap)$ Tuples in both reln. 1 and reln. 2.
  - **Set-difference** $(\setminus)$ Tuples in reln. 1, but not in reln. 2.
  - **GROUP BY** and **Aggregation** (SUM, MIN, etc.)

with cost estimation, which leads to **cost-based optimization**
Outline

- Selection
  - Sorting routine
  - Join
  - Projection
  - Set operators
  - Group By aggregation
Schema for Examples

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

- **Sailors:**
  - Each tuple is 50 bytes long,
  - 80 tuples per page,
  - 500 pages.
- **Reserves:**
  - Each tuple is 40 bytes long,
  - 100 tuples per page,
  - 1000 pages.
- **Cost metric:** \# I/Os
Using an Index for Selections

```
SELECT * 
FROM   Sailors S 
WHERE  S.rating > 8
```

- Cost of selection includes:
  1) top down search in the index
  2) scan the relevant leaf nodes
  3) retrieve records from file (could be large w/o clustering)
Using an Index for Selections

Cost of selection includes:
1) top down search in the index
2) scan the relevant leaf nodes
3) retrieve records from file (could be large w/o clustering)

- Step 1) top down search: $\leq 3-4$ I/Os, depending on the height of the tree and buffer management

```
SELECT * 
FROM   Sailors S 
WHERE  S.rating > 8 
```
Cost Factors of Steps 2 and 3

- Cost of selection includes:
  1) top down search in the index
  2) scan the relevant leaf nodes
  3) retrieve records from file (could be large w/o clustering)

- **Step 2 scanning leaf nodes**: cost factors include % of qualifying tuples and num. of leaf nodes
  - **rating > 8**: 20% of tuples qualify
  - If a data entry is 1/5 of a tuple, need $500/5/5 = 20$ leaf nodes, so 20 I/Os.

```
SELECT * 
FROM    Sailors S 
WHERE   S.rating > 8 
```
Step 3 retrieving records from file: cost factors include *num. of qualifying tuples and clustering*

- **rating > 8**: 20% of tuples, 500/5=100 data pages, 80*100=8,000 tuples
- **Retrieving records from file ~**
  - **Clustered index**: 100 I/Os.
  - **Unclustered index**: worst case 1 I/O per tuple; 8,000 I/Os here!
  - **Unclustered index + Sorting based on rid**: ≤ 500 I/Os.
    (Bitmap Index Scan + Bitmap Heap Scan in PostgreSQL)
General Selections

- Boolean combination of predicates using AND and OR.
  - Conjunctive Normal Form (CNF), e.g.,
    \[ \text{pred1 AND (pred3 OR pred4)} \]
    \[ (\text{pred1 OR pred2}) \text{ AND (pred3 OR pred4)} \]

- *File scan* always works for general selections.

- *Index scan* works when it matches a predicate that is a conjunct of CNF.
  - E.g., an index matching \( \text{pred1} \) can be used for
    \[ \text{pred1 AND (pred3 OR pred4)} \]
Conjunctive Predicates Only

- **CNF without OR:** e.g. \( \text{pred 1 AND pred 2 AND pred 3} \)
  - Find the *most selective access path*, retrieve tuples using it
    - File scan or index scan that gives the smallest I/O cost.
  - Apply remaining terms that don’t match index *on the fly*.
    - Other terms do not affect I/O cost.

\[ \text{day<8/9/94 AND bid=5 AND sid=3} \]

- Hash index on \( \text{<bid, sid>} \): check \( \text{day<8/9/94} \) on the fly.
- B+ tree index on \( \text{day} \): apply \( \text{bid=5 and sid=3} \) on the fly.
**Improvement: Intersection of Rids**

- 2+ matching indexes (Alternatives 2 or 3):
  1. Get sets of rids of data records using each index.
  2. \textit{Intersect} these sets of rids.
  3. Retrieve the records and apply any remaining terms.

\[
day<8/9/94 \text{ AND bid}=5 \text{ AND sid}=3
\]

B+ tree index on \textit{day}, a hash index on \textit{sid}, both using Alt 2:

1. retrieve rids of records satisfying \textit{day}<8/9/94 using first index,
   rids of records satisfying \textit{sid}=3 using second index,
2. intersect these rids,
3. retrieve records, check \textit{bid}=5.