Evaluating Relational Operators: Part II

From Chapter 14

Relational Operators

- Select
- Project
- Join
- Set operations (union, intersect, except)
- Aggregation

Example

```
SELECT *
FROM Reserves R, Sailor S,
WHERE R.sid = S.sid
```

- No indices on Sailor or Reserves
Tuple Nested Loop Join

foreach tuple r in R do
    foreach tuple s in S do
        if r.sid == s.sid then add <r, s> to result

- R is “outer” relation
- S is “inner” relation

Analysis

- Assume
  - M pages in R, $p_R$ tuples per page
    - $M = 1000, p_R = 100$
  - N pages in S, $p_S$ tuples per page
    - $N = 500, p_S = 80$
- Total cost = ________________
  - Ignore cost of writing out result
  - Same for all join methods
- Main problem: depends on # tuples per page

Page Nested Loops Join

foreach page p1 in R do
    foreach page p2 in S do
        foreach r in p1 do
            foreach s in p2 do
                if r.sid == s.sid then add <r, s> to result

- R is “outer” relation
- S is “inner” relation
Analysis

- Assume
  - M pages in R, $p_R$ tuples per page
    - $M = 1000, p_R = 100$
  - N pages in S, $p_S$ tuples per page
    - $N = 500, p_S = 80$
- Total cost = __________
  - Note: _______ relation should be “outer”
  - Better for __ to be “outer” in this case!
- Main problem: does not use all buffer pages

Block Nested Loops Join

- Use one page as an input buffer for scanning the inner S, one page as the output buffer, and use all remaining pages to hold “block” of outer R.
  - For each matching tuple $r$ in R-block, $s$ in S-page, add <r, s> to result. Then read next R-block, scan S, etc.

Analysis

- Assume
  - M pages in R, $p_R$ tuples per page
    - $M = 1000, p_R = 100$
  - N pages in S, $p_S$ tuples per page
    - $N = 500, p_S = 80$
  - BS is the block size
- Total cost = __________
  - With sequential reads, analysis changes: may be best to divide buffers evenly between R and S.
Example

```sql
SELECT *
FROM Reserves R, Sailor S,
WHERE R.sid = S.sid
```

- No indices on Sailor or Reserves

Sort-Merge Join

- Sort R on the join attributes
- Sort S on the join attributes
- Merge sorted relations to produce join result
  - Advance r in R until r.sid >= s.sid
  - Advance s in S until s.sid >= r.sid
  - If r.sid = s.sid
    - All R tuples with same value as r.sid is current R group
    - All S tuples with same value as s.sid is current S group
    - Output all <rg, sg> pairs, where rg is in current R group, sg is in current S group
  - Repeat

Example of Sort-Merge Join

<table>
<thead>
<tr>
<th>sid</th>
<th>sid</th>
<th>bid</th>
<th>day</th>
<th>rname</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>28</td>
<td>103</td>
<td>12/4/96</td>
<td>puppy</td>
</tr>
<tr>
<td>28</td>
<td>28</td>
<td>103</td>
<td>11/3/96</td>
<td>yuppy</td>
</tr>
<tr>
<td>31</td>
<td>31</td>
<td>101</td>
<td>10/10/96</td>
<td>dustin</td>
</tr>
<tr>
<td>44</td>
<td>31</td>
<td>102</td>
<td>10/12/96</td>
<td>lubber</td>
</tr>
<tr>
<td>58</td>
<td>58</td>
<td>103</td>
<td>10/11/96</td>
<td>lubber</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>11/12/96</td>
<td>dustin</td>
</tr>
</tbody>
</table>
Analysis

- Assume
  - M pages in R, p_R tuples per page
    - M = 1000, p_R = 100
  - N pages in S, p_S tuples per page
    - N = 500, p_S = 80
- Total cost =
  - Note: (M + N) could be (M * N) in worst case

- With 300 buffer pages, R and S sorted in __ passes
  - Sort-merge join cost: __________
  - BNL join cost: __________

Refinement of Sort-Merge Join

- We can combine the merging phases in the sorting of R and S with the merging required for the join.

  ![Diagram of sort-merge join]

- Requirement: M/2B + N/2B <= B – 1

Analysis

- Assume
  - M pages in R, p_R tuples per page
    - M = 1000, p_R = 100
  - N pages in S, p_S tuples per page
    - N = 500, p_S = 80
- Total cost = __________
  - Phase 1: __________
  - Phase 2: __________
Example

```
SELECT *
FROM Reserves R, Sailor S,
WHERE R.sid = S.sid
```

- No indices on Sailor or Reserves

**Hash-Join**

- Partition both relations using hash fn $h$: $R$ tuples in partition $i$ will only match $S$ tuples in partition $i$.
- Read in a partition of $R$, hash it using $h_2 (\neq h_1)$. Scan matching partition of $S$, search for matches.

**Analysis (without recursive partitioning)**

- Assume
  - $M$ pages in $R$, $p_R$ tuples per page
    - $M = 1000$, $p_R = 100$
  - $N$ pages in $S$, $p_S$ tuples per page
    - $N = 500$, $p_S = 80$
- Total cost = __________
  - Phase 1: __________
  - Phase 2: __________
- Requirement: $M/(B - 1) \leq B - 2$
Hash-Join vs. Sort-Merge Join

- Given a minimum amount of memory, both have cost of 3 \((M + N)\)
- Benefits of hash join
  - Superior if relation sizes differ greatly
  - Highly parallelizable
- Sort merge join
  - Less sensitive to data skew
  - Result is sorted

Example

```sql
SELECT *
FROM Reserves R, Sailor S,
WHERE R.sid = S.sid
```

- Hash index on Sailor.sid

Example

```sql
SELECT *
FROM Reserves R, Sailor S,
WHERE R.sid > S.sid
```

- B+-tree index on Sailor.sid
Index Nested Loops Join

\[
\text{foreach tuple } r \text{ in } R \text{ do}
\]
\[
\text{foreach tuple } s \text{ in } S \text{ where } ri \text{ OP } sj \text{ do}
\]
\[
\text{add } <r, s> \text{ to result}
\]

- R is "outer" relation
- S is "inner" relation

Analysis

- Assume
  - M pages in R, \( p_R \) tuples per page
    - \( M = 1000, p_R = 100 \)
  - N pages in S, \( p_S \) tuples per page Select
    - \( N = 500, p_S = 80 \)
  - IC is cost of probing index
    - Different based on hash index or B+ tree index, predicate
- Total cost =

General Join Conditions

- Equalities over several attributes (e.g., \( R.sid = S.sid \) AND \( R.mame = S.sname \)):
  - For Index NL, index on \(<sid, sname>\) (if S is inner); or indices on \( sid \) or \( sname \).
  - For Sort-Merge and Hash Join, sort/partition on combination of the two join columns.
- Inequality conditions (e.g., \( R.mame < S.sname \)):
  - For Index NL, need (clustered!) B+ tree index.
    - Range probes on inner; # matches likely to be much higher than for equality joins.
  - Hash Join, Sort Merge Join not applicable.
  - Block NL quite likely to be the best join method here.
Relational Operators

- Select
- Project
- Join
- Set operations (union, intersect, except)
- Aggregation

Set Operations

- Intersection and cross-product special cases of join.
- Union (Distinct) and Except similar
- Sorting based approach to union:

- Hash based approach to union:
Example

SELECT MAX(S.age)
FROM Sailor S

GROUP BY S.rating