Overview of Query Evaluation

Chapter 12
Set Operations

- Intersection and cross-product special cases of join.
- Union (Distinct) and Except similar

Sorting based approach to union:
- Sort both relations (on combination of all attributes).
- Scan sorted relations and merge them.
- Alternative: Merge runs from Pass 1 for both relations.

Hash based approach to union:
- Partition R and S using hash function $h$.
- For each S-partition, build in-memory hash table (using $h2$), scan corr. R-partition and add tuples to table while discarding duplicates.
Aggregate Operations (AVG, MIN, etc.)

- **Without grouping:**
  - In general, requires scanning the relation.
  - Given index whose search key includes all attributes in the SELECT or WHERE clauses, can do index-only scan.

- **With grouping:**
  - Sort on group-by attributes, then scan relation and compute aggregate for each group. (Can improve upon this by combining sorting and aggregate computation.)
  - Similar approach based on hashing on group-by attributes.
  - Given tree index whose search key includes all attributes in SELECT, WHERE and GROUP BY clauses, can do index-only scan; if group-by attributes form prefix of search key, can retrieve data entries/tuples in group-by order.
Highlights of System R Optimizer

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

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

Query Parser

Query Optimizer

Plan Generator
Plan Cost Estimator

Catalog Manager

Query Plan Evaluator
Cost Estimation

- For each plan considered, must estimate cost:
  - Must estimate cost of each operation in plan tree.
    - Depends on input cardinalities.
    - We’ve already discussed how to estimate the cost of operations (sequential scan, index scan, joins, etc.)
  - Must also 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 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!
  - Term \( col=value \) has RF \( 1/N\text{Keys}(I) \), given index I on \( col \)
  - Term \( col1=col2 \) has RF \( 1/\max(N\text{Keys}(I1), N\text{Keys}(I2)) \)
  - Term \( col>value \) has RF \( (\text{High}(I)-value)/(\text{High}(I)-\text{Low}(I)) \)
Schema for Examples

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

<table>
<thead>
<tr>
<th></th>
<th>S</th>
<th>R</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pages</td>
<td>N=500</td>
<td>M=1,000</td>
</tr>
<tr>
<td>Tuples/page</td>
<td>p_S = 80</td>
<td>p_R = 100</td>
</tr>
</tbody>
</table>
## Motivating Example

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

\[
\begin{align*}
&\text{SELECT S.sname} \\
&\text{FROM Reserves R, Sailors S} \\
&\text{WHERE R.sid=S.sid AND R.bid=100 AND S.rating>5}
\end{align*}
\]

<table>
<thead>
<tr>
<th></th>
<th>S</th>
<th>R</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pages</td>
<td>N=500</td>
<td>M=1,000</td>
</tr>
<tr>
<td>Tuples/page</td>
<td>p_S = 80</td>
<td>p_R = 100</td>
</tr>
</tbody>
</table>
Motivating Example

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

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

- **Cost:** $500+500*1000 = 500,500$ I/Os
- **Misses several opportunities:** selections could have been `pushed` earlier, no use is made of any available indexes, etc.
- **Goal of optimization:** To find more efficient plans that compute the same answer.

<table>
<thead>
<tr>
<th></th>
<th>$S$</th>
<th>$R$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pages</td>
<td>$N=500$</td>
<td>$M=1,000$</td>
</tr>
<tr>
<td>Tuples/page</td>
<td>$p_S = 80$</td>
<td>$p_R = 100$</td>
</tr>
</tbody>
</table>
Alternative Plans 1
(No Indexes)

(Scan; write to temp T1)

(On-the-fly)

(Sort-Merge Join)

(sid=sid)

(Scan; write to temp T2)

Reserves

Sailors

(sid=sid)

(On-the-fly)

rating > 5

bid=100

σ

S

R

<table>
<thead>
<tr>
<th>Pages</th>
<th>S</th>
<th>R</th>
</tr>
</thead>
<tbody>
<tr>
<td>N=500</td>
<td>M=1,000</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Tuples/page</th>
<th>S</th>
<th>R</th>
</tr>
</thead>
<tbody>
<tr>
<td>ps = 80</td>
<td>pR = 100</td>
<td></td>
</tr>
</tbody>
</table>
Alternative Plans 1 (No Indexes)

- **Main difference:** *push selects.*

- **Cost of plan**
  - Scan Reserves (1000) + write temp T1 (10 pages, if we have 100 boats, uniform distribution).
  - Scan Sailors (500) + write temp T2 (250 pages, if we have 10 ratings).
  - Sort T1 (2*2*10), sort T2 (2*2*250), merge (10+250)
  - Total: 3060 page I/Os.

- **If we `push' projections, T1 has only sid, T2 only sid and sname:**
  - T1 fits in 3 pages, cost of join drops to under 250 pages, total < 2000.
**Alternative Plans 2**

**With Indexes**

\[
\begin{align*}
\exists \text{ sid=SID} & \quad \text{(Index Nested Loops, with pipelining)} \\
\sigma \text{ rating > 5} & \quad \text{(On-the-fly)} \\
\pi \text{ sname} & \quad \text{(On-the-fly)}
\end{align*}
\]

---

<table>
<thead>
<tr>
<th></th>
<th><strong>S</strong></th>
<th><strong>R</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Pages</td>
<td>N=500</td>
<td>M=1,000</td>
</tr>
<tr>
<td>Tuples/page</td>
<td>(p_S = 80)</td>
<td>(p_R = 100)</td>
</tr>
</tbody>
</table>
**Alternative Plans 2**

- With clustered index on *bid* of Reserves, we get 100,000/100 = 1000 tuples on 10 pages.
- INL with **pipelining** (outer is not materialized).

  - Projecting out unnecessary fields from outer doesn’t help.
  - Join column *sid* is a key for Sailors.
    - At most one matching tuple, unclustered index on *sid* OK.
  - Decision not to push *rating*>5 before the join is based on availability of *sid* index on Sailors.

- **Cost:** Selection of Reserves tuples (10 I/Os); for each, must get matching Sailors tuple:
  - $10 + 1000 \times 1.2 = 1,210$ (Alt. 1)
  - $10 + 1000 \times (1.2+1) = 2,210$ (Alt. 2)

<table>
<thead>
<tr>
<th></th>
<th>S</th>
<th>R</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pages</td>
<td>N=500</td>
<td>M=1,000</td>
</tr>
<tr>
<td>Tuples/page</td>
<td>$p_S = 80$</td>
<td>$p_R = 100$</td>
</tr>
</tbody>
</table>