Query Optimization

• Two main issues:
  – For a given query, what plans are considered?
  – How is the cost of a plan estimated?

• Ideally: Want to find best plan.
  Practically: Avoid worst plans!

• System R Optimizer:
  – Most widely used currently; works well for < 10 joins.
  – Statistics, maintained in system catalogs, used to estimate cost of operations and result sizes.
  – Only the space of left-deep plans is considered.
  – Cartesian products avoided.
Example

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

<table>
<thead>
<tr>
<th></th>
<th>Reserves</th>
<th>Sailors</th>
</tr>
</thead>
<tbody>
<tr>
<td>Size</td>
<td>40 bytes/tuple</td>
<td>50 bytes/tuple</td>
</tr>
<tr>
<td>Pages</td>
<td>100 tuples/page</td>
<td>80 tuples/page</td>
</tr>
<tr>
<td>Total</td>
<td>1000 pages</td>
<td>500 pages</td>
</tr>
</tbody>
</table>

- Nested Loop Join cost 1K+ 100K*500
- On the fly selection and project does not incur any disk access.
- Total disk access = 500001K (worst case)
What about complex queries?

- For each block, the plans considered are:
  - All available access methods, for each reln in FROM clause.
  - All left-deep join trees (i.e., all ways to join the relations one-at-a-time, with the inner reln in the FROM clause, considering all reln permutations and join methods.)

```
SELECT S.sid, MIN(R.day)
FROM Sailors S, Reserves R, Boats B
WHERE S.sid=R.sid AND R.bid=B.bid AND B.color='red' AND S.rating = (SELECT MAX(S2.rating)
FROM Sailors S2)
GROUP BY S.sid
HAVING COUNT(*) > 1
```
RA Equivalences

• Selections
  – Cascade: \( \sigma_{c_1 \land \ldots \land c_n}(R) \equiv \sigma_{c_1}(\ldots \sigma_{c_n}(R)\ldots) \)
  – Commute: \( \sigma_{c_1}(\sigma_{c_n}(R)) \equiv \sigma_{c_n}(\sigma_{c_1}(R)) \)

• Projections
  – Cascade: \( \pi_{c_1}(R) \equiv \pi_{c_1}(\ldots \pi_{c_n}(R)\ldots) \), \( c_1 \subset c_i \), \( i > 1 \)

• Joins
  – Associativity: \( R \text{ join } (S \text{ join } T) \equiv (R \text{ join } S) \text{ join } T \)
  – Commutative: \( R \text{ join } S \equiv S \text{ join } R \)
  – Definition: \( R \text{ join } S \equiv \sigma_{R.\text{col}=S.\text{col}}(R \times S) \)
More equivalences

• Commutability between projection & selection

\[- \pi_{c1, ..., cn} (\sigma_{\text{predicate}} (S)) \equiv \sigma_{\text{predicate}} (\pi_{c1, ..., cn} (S)) \text{ iff predicate only uses } c1, ..., cn\]

• Commutability between selection & join (predicate pushdown)

\[- \sigma_{\text{predicate}} (R \text{ join } S) \equiv (\sigma_{\text{predicate}} (R)) \text{ join } S \text{ iff predicate only uses attributes from } R\]

• Commutability between projection & join

\[- \pi_{c1, ..., cn} (R \text{ join}_{cr=cs} S) \equiv (\pi_{c1, ..., cn, cr}(R)) \text{ join}_{cr=cs} S\]
Example: Using Equivalences

\[ \pi_{S.sname} \]

\[ \sigma_{S.rating>5 \land R.bid=100} \]

\[ R.sid=S.sid \]

\[ R.sid=S.sid \]

\[ \pi_{S.sname} \]

\[ \sigma_{R.bid=100} \]

\[ \sigma_{S.rating>5} \]

\[ \pi_{S.sid, S.sname} \]

\[ \sigma_{R.bid=100} \]

\[ \sigma_{S.rating>5} \]
Cost Estimation

• Obvious inefficient plans are pruned during enumeration. Eg. Predicate pushdown etc.

• For each plan considered,
  – 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.
Example: Predicate Pushdown

**SELECT**  
S.sname

**FROM**  
Reserves R, Sailors S

**WHERE**  
R.sid=S.sid AND R.bid=100 AND S.rating>5

Reserves | 40 bytes/tuple | 100 tuples/page | 1000 pages
Sailors | 50 bytes/tuple | 80 tuples/page | 500 pages

- Nested Loop Join requires materializing the inner table as T1.
- With 50% selectivity, T1 has 250 pages
- With 10% selectivity, outer “table” in join has 10K tuples
- Disk accesses for scans = 1000 + 500
- Writing T1 = 250
- NLJoin = 10K * 250
- Total disk access = 2500.175 K (worst case)

What happens if we make the left leg the inner table of the join?
Example: Sort Merge Join

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

<table>
<thead>
<tr>
<th>Table</th>
<th>Bytes/tuple</th>
<th>Tuples/page</th>
<th>Pages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reserves</td>
<td>40</td>
<td>100</td>
<td>1000</td>
</tr>
<tr>
<td>Sailors</td>
<td>50</td>
<td>80</td>
<td>500</td>
</tr>
</tbody>
</table>

- Sort Merge Join requires materializing both legs for sorting.
- With 10% selectivity, T1 has 100 pages
- With 50% selectivity, T2 has 250 pages
- Disk accesses for scans = 1000 + 500
- Writing T1 & T2 = 100 + 250
- Sort Merge Join = 100 log 100 + 250 log 250 + 100+250 (assume 10 way merge sort)
- Total disk access = 52.8 K

What happens if we make the left leg the inner table of the join?
Example: Index Nested Loop Join

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

- With 10% selectivity, selection on R has 10K tuples
- Disk accesses for scan = 1000
- Index Nested Loop Join = 10K*(1 + log_{10} 200) = 33K
- Total disk access = 34 K
Join Ordering

- Independent of what join algorithm is chosen, the order in which joins are performed affects the performance.
- Rule of thumb: do the most “selective” join first.
- In practice, left deep trees (e.g. the right one above) are preferred --- why?

<table>
<thead>
<tr>
<th>Relations</th>
<th>Tuples</th>
<th>Pages</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>10K</td>
<td>1000</td>
</tr>
<tr>
<td>B</td>
<td>20K</td>
<td>2000</td>
</tr>
<tr>
<td>C</td>
<td>30K</td>
<td>3000</td>
</tr>
<tr>
<td>A join B</td>
<td>10K</td>
<td>1000</td>
</tr>
<tr>
<td>B join C</td>
<td>1K</td>
<td>100</td>
</tr>
</tbody>
</table>

![Diagram showing join operations and associated tuple counts and page numbers.](image-url)
How to estimate the selectivity & cardinality?

\[ \sigma_{col=value} \]
- Arbitrary constant 10%
- \( \frac{1}{\text{Number of distinct values in the column}} \)
- \( \frac{1}{\text{Number of keys in Index(col)}} \)

\[ \sigma_{col>value} \]
- Arbitrary constant of 50% if non numeric
- \( \frac{(\text{High Key} - \text{value})}{(\text{High Key} - \text{Low Key})} \)

\[ \sigma_{R.col=S.col} \]
- Join result size
- Arbitrary constant 10%
- \( \frac{1}{\text{MAX( Nkeys(Index(R.col), Nkeys(Index(S.col)) )} \)} \)

Can we do better?
Histograms

Age

<table>
<thead>
<tr>
<th>18</th>
<th>18</th>
<th>19</th>
<th>24</th>
<th>25</th>
<th>29</th>
<th>30</th>
<th>34</th>
<th>40</th>
<th>50</th>
<th>58</th>
<th>61</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Bar chart" /></td>
<td><img src="image" alt="Bar chart" /></td>
<td><img src="image" alt="Bar chart" /></td>
<td><img src="image" alt="Bar chart" /></td>
<td><img src="image" alt="Bar chart" /></td>
<td><img src="image" alt="Bar chart" /></td>
<td><img src="image" alt="Bar chart" /></td>
<td><img src="image" alt="Bar chart" /></td>
<td><img src="image" alt="Bar chart" /></td>
<td><img src="image" alt="Bar chart" /></td>
<td><img src="image" alt="Bar chart" /></td>
<td><img src="image" alt="Bar chart" /></td>
</tr>
</tbody>
</table>

\[ \sigma_{\text{age}>45} : 0.5 \times f(D) + f(E) + f(G) \]

Equi-width buckets

\[ \sigma_{\text{age}>30} : \frac{(34-30)}{(34-25)} \times B + C \]

Equi-depth buckets

Each Bucket Counts 4 entries
Statistics Collection in DBMS

• Page size

• Data Statistics:
  – Record size -> number of records per data page
  – Cardinality of relations (including temporary tables)
  – Selectivity of selection operator on different columns of a relation

• (Tree) Index Statistics
  – number of leaf pages, index entries
  – Height

• Statistics collection is user triggered
  – DB2: RUNSTATS ON TABLE mytable AND INDEXES ALL
What about the parallel/distributed case?

- QEP enumeration/rewrite
  - Main “trick” is expressing a horizontally fragmented table as a union of fragments in RA
  - Push the union up. Conversely push the $\sigma, \pi, \times$ down.
  - Eliminate sub-trees that return empty results.
- Cost estimation takes into account communication costs.

```sql
SELECT S.sname
FROM Sailors S
WHERE S.rating > 5
```

Sailors partitioned by ranges [0,5),[5,10] on rating

\[
\pi_{S.sname} (\sigma_{S.rating} > 5 \ S) \\
= \pi_{S.sname} (\sigma_{S.rating} > 5 \ (S1 \cup S2)) \\
= \pi_{S.sname} ((\sigma_{S.rating} > 5 \ S1) \cup (\sigma_{S.rating} > 5 \ S2)) \\
= (\pi_{S.sname} \sigma_{S.rating} > 5 \ S1) \cup (\pi_{S.sname} \sigma_{S.rating} > 5 \ S2)
\]

S1’s range is [0,5), so selection is empty!
Distributed Multi-table Query

\[ R \Join S = \sigma_{R.sid=S.sid} (R \times S) \]
\[ = \sigma_{R.sid=S.sid} ((R1 \cup R2) \times (S1 \cup S2)) \]
\[ = \sigma_{R.sid=S.sid} ((R1 \times S1) \cup (R1 \times S2) \cup (R2 \times S1) \cup (R2 \times S2)) \]
\[ = \sigma_{R.sid=S.sid} (R1 \times S1) \cup \sigma_{R.sid=S.sid} (R1 \times S2) \]
\[ \quad \cup \sigma_{R.sid=S.sid} (R2 \times S1) \cup \sigma_{R.sid=S.sid} (R2 \times S2) \]
\[ = (R1 \Join S1) \cup (R1 \Join S2) \cup (R2 \Join S1) \cup (R2 \Join S2) \]

Equivalent to a union of joins over each pair of fragments