Overview of Query Processing

Asst. Prof. Lipyeow Lim
Information & Computer Science Department
University of Hawaii at Manoa
SELECT * FROM Reserves WHERE sid=101

Optimizer

Evaluate Plan A

Pick B

Evaluate Plan A

Reserves

IDXSCAN (sid=101) Reserves

Index(sid)

fetch 25.0

Reserves

SCAN (sid=101)

Reserves

32.0

Choose Best Plan

Estimate Cost

Enumerate Plans

Parse Query

Result

Evaluate Query Plan

Choose Best Plan

Estimate Cost

Enumerate Plans

Parse Query

Result

4/9/2012

Lippyew Lim -- University of Hawaii at Manoa
Query Processing

• **Query Execution Plan (QEP):** tree of database operators.
  – At high-level, relational algebra operators are used
  – At low-level, RA operators with particular implementation algorithm.

• **Plan enumeration:** find equivalent plans
  – Different QEPs that return the same results
  – Query rewriting: transformation of one QEP to another equivalent QEP.

• **Cost estimation:** a mapping of a QEP to a cost
  – **Cost Model:** a model of what counts in the cost estimate. Eg. Disk accesses, CPU cost ...

• **Query Optimizer:**
  – Explores the space of equivalent plan for a query
  – Chooses the best plan according to a cost model
Access Paths

• An **access path** is a method of retrieving tuples. Eg. Given a query with a selection condition:
  – File or table scan
  – Index scan

• **Index matching problem:** given a selection condition, which indexes can be used for the selection, i.e., matches the selection?
  – Selection condition normalized to conjunctive normal form (CNF), where each term is a *conjunct*
  – Eg. (day<8/9/94 **AND** rname=‘Paul’) **OR** bid=5 **OR** sid=3
  – **CNF:** (day<8/9/94 **OR** bid=5 **OR** sid=3 ) **AND** (rname=‘Paul’ **OR** bid=5 **OR** sid=3)
Index Matching

- A tree index matches a selection condition if the selection condition is a prefix of the index search key.
- A hash index matches a selection condition if the selection condition has a term `attribute=value` for every attribute in the index search key.
One Approach to Selections

1. Find the *most selective access path*, retrieve tuples using it
2. Apply remaining terms in selection not matched by the chosen access path

- The **selectivity** of an access path is the size of the result set (in terms of tuples or pages).
  - Sometimes selectivity is also used to mean **reduction factor**: fraction of tuples in a table retrieved by the access path or selection condition.

- Eg. Consider the selection:
  
  \[
  \text{day}<8/9/94 \ \text{AND} \ \text{bid}=5 \ \text{AND} \ \text{sid}=3
  \]
  - Tree Index(day)
  - Hash index (bid,sid)
Query Execution Plans

• A tree of database operators: each operator is a RA operator with specific implementation
• Selection $\sigma$: Index Scan or Table Scan
• Projection $\pi$:
  – Without DISTINCT : Table Scan
  – With DISTINCT : requires sorting or index scan
• Join $\Join$:
  – Nested loop joins (naïve)
  – Index nested loop joins
  – Sort merge joins
Nested Loop Join

For each data page $P_{S1}$ of $S1$
  For each tuple $s$ in $P_{S1}$
    For each data page $P_{R1}$ of $R1$
      For each tuple $r$ in $P_{R1}$
        if ($s$.sid == $r$.sid) 
          then output $s,r$

• Worst case number of disk reads 
  = $N_{pages}(S1) + |S1| \times N_{pages}(R1)$
Index Nested Loop Join

For each data page \( P_{S1} \) of \( S1 \)
For each tuple \( s \) in \( P_{S1} \)
if \( (s.sid \in \text{Index}(R1.cid)) \)
then fetch \( r \) & output \( <s, r> \)

- Worst case number of disk reads with tree index
  \[ = N_{\text{pages}}(S1) + |S1| \times (1 + \log_F N_{\text{pages}}(R1)) \]
- Worst case number of disk reads with hash index
  \[ = N_{\text{pages}}(S1) + |S1| \times 2 \]
Sort Merge Join

1. Sort $S_1$ on SID
2. Sort $R_1$ on SID
3. Compute join on SID using Merging algorithm

• If join attributes are relatively unique, the number of disk pages
  
  $= N_{\text{pages}}(S_1) \log N_{\text{pages}}(S_1)$
  $+ N_{\text{pages}}(R_1) \log N_{\text{pages}}(R_1)$
  $+ N_{\text{pages}}(S_1) + N_{\text{pages}}(R_1)$

• What if the number of duplicates is large?
  – the number of disk pages approaches that of nested loop join.
Example

**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>Bytes</td>
<td>40 bytes/tuple</td>
<td>50 bytes/tuple</td>
</tr>
<tr>
<td>Tuples</td>
<td>100 tuples/page</td>
<td>80 tuples/page</td>
</tr>
<tr>
<td>Pages</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)
Example: Predicate Pushdown

Select: S.sname
From: Reserves R, Sailors S
Where: R.sid=S.sid AND R.bid=100 AND S.rating>5

- 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

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

- With 10% selectivity, selection on R has 10K tuples
- Disk accesses for scan = 1000
- Index Nested Loop Join = $10K \times (1 + \log_{10} 500) = 37K$
- Total disk access = 38 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>
Statistics & Cost Estimation

• 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