Query Processing

From SQL to Runtime

Processing Steps

Query in High Level Language -> QUERY Processor

Runtime Code -> Runtime Processor -> Results
Processing Steps

Query in High Level Language → Scan, Parse Validate, Compile → Intermediate Form of Query

Optimizer → Execution Plan → Runtime Processor → Results

Code Generator

Runtime Code

High Level Language

• SQL
• OOQL
• QUEL
• QBE
• Not record-by-record language
Intermediate Representation

- Relational Algebra Query Tree
- (others also used)
- Tree can be reordered by rules to reduce cost.
- Each operation has several implementations.

Example Query

- For each Stafford project, retrieve project number, controlling dept number, and manager’s name and address
- SELECT p.pnumber, p.dnum, e.lname, e.address
  FROM project p, department d, employee e
  WHERE p.dnum=d.dnumber
  AND d.mgrssn=e.ssn
  AND p.plocation='Stafford'
SQL to Rel Algebra

- SELECT p.pnumber, p.dnum, e.lname, e.address
  FROM project p, department d, employee e
  WHERE p.dnum = d.dnumber
  AND d.mgrssn = e.ssn
  AND p.plocation = 'Stafford'
- R1 ← project × department
- R2 ← R1 × employee
- R3 ← σ_{dnum=dnumber}(R2)
- R4 ← σ_{mgrssn=e.ssn}(R3)
- R5 ← σ_{plocation='Stafford'}(R4)
- π_{pnumber, dnum, lname, address}(R5)

Rel Algebra Query Tree
Move Selects Down

\[ R_1 \leftarrow \text{project} \times \text{department} \]
\[ R_2 \leftarrow R_1 \times \text{employee} \]
\[ R_3 \leftarrow \sigma_{dnum=dnumber}(R_2) \]
\[ R_4 \leftarrow \sigma_{mgrssn=e.ssn}(R_3) \]
\[ R_5 \leftarrow \sigma_{plocation='Stafford'}(R_4) \]
\[ \pi_{pnumber,dnum,lname,address}(R_5) \]

Introduce Joins

\[ R_1 \leftarrow \text{project} \times \text{department} \]
\[ R_2 \leftarrow R_1 \times \text{employee} \]
\[ R_3 \leftarrow \sigma_{dnum=dnumber}(R_2) \]
\[ R_4 \leftarrow \sigma_{mgrssn=e.ssn}(R_3) \]
\[ R_5 \leftarrow \sigma_{plocation='Stafford'}(R_4) \]
\[ \pi_{pnumber,dnum,lname,address}(R_5) \]
Remove Unneeded Fields

R1 ← project × department
R2 ← R1 × employee
R3 ← σ_{dnum=dnumber} (R2)
R4 ← σ_{mgrssn=e.ssn} (R3)
R5 ← σ_{plocation='Stafford'} (R4)
p_{number,dnum,lname,address}(R5)

π_{pnumber,dnum,lname,address}

Original vs Optimized

R1 ← project × department
R2 ← R1 × employee
R3 ← σ_{dnum=dnumber} (R2)
R4 ← σ_{mgrssn=e.ssn} (R3)
R5 ← σ_{plocation='Stafford'} (R4)
p_{number,dnum,lname,address}(R5)

• R1a ← σ_{plocation='Stafford'} (project)
• R1b ← π_{pnumber,dnum}(R0a)
• R1c ← π_{dnum,mgrssn}(department)
• R3 ← R1b □_{dnum=dnumber} R1c
• R4a ← π_{ssn,lname,address}(employee)
• R4 ← R3 □_{mgrssn=e.ssn} R4a
• π_{pnumber,dnum,lname,address}(R4)
Implementing Operations

\[ \pi_{\text{pnumber}, \text{dnum}} \]
\[ \sigma_{\text{plocation} = 'Stafford'} \]
\[ \text{project} \]

- Pipeline with \( \sigma \) results
- Use Full Table Scan
- Use Index on Plocation

Implementing Join

- Nested Loop
  - For each tuple in A
    - Loop thru B and test join condition
- Single Loop with Index on Join Field of B
  - For each tuple in A
    - Look in index for matching B tuples and retrieve them
Sort Merge Join

• Sort A and B on join fields
• Scan each once to effect join

<table>
<thead>
<tr>
<th></th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Hash Join

• Use hash function $H$ on join attributes of smaller table to copy tuples into buckets
  – Result: table partitioned into buckets
• Use $H$ on join attributes of other table, record by record, to see which bucket to search for matches
**Hash Join**

**Hash Function:** \( H(x) : x \mod 3 \)

<table>
<thead>
<tr>
<th>Table A</th>
<th>Table B</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>16</td>
</tr>
<tr>
<td>7</td>
<td>9</td>
</tr>
<tr>
<td>12</td>
<td>12</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>16</td>
<td>7</td>
</tr>
<tr>
<td>9</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td></td>
</tr>
</tbody>
</table>

*Table B copied into buckets 0, 1 and 2 based on \( H() \)*
Hash Function: $H(x) : x \mod 3$

Table A

<table>
<thead>
<tr>
<th>9</th>
<th>11</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>15</td>
<td></td>
</tr>
</tbody>
</table>

$H(11) = 2$

$H(7) = 1$

Bucket 0

<table>
<thead>
<tr>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
</tr>
<tr>
<td>12</td>
</tr>
</tbody>
</table>

Bucket 1

<table>
<thead>
<tr>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
</tr>
</tbody>
</table>

Bucket 2

| 8 |

Hash Join

Table B copied into buckets 0, 1 and 2 based on $H()$

Optimization

- Apply heuristic rules to optimize query tree
  - Minimize size of intermediate result sets
- Use statistics to determine:
  - Which leaves of tree to start with
  - Which implementation of each operation to use