Query Processing

From SQL to Runtime
Processing Steps

Query in High Level Language
Processing Steps

Query in High Level Language → QUERY Processor
Processing Steps

Query in High Level Language

QUERY Processor

Runtime Code
Processing Steps

Query in High Level Language

QUERY Processor

Runtime Code

Runtime Processor
Processing Steps

1. Query in High Level Language
2. Query Processor
3. Runtime Code
4. Runtime Processor
5. Results
Processing Steps

Query in High Level Language

Compiler
Processing Steps

Query in High Level Language → Scan, Parse, Validate, Compile
Processing Steps

1. Query in High Level Language
2. Scan, Parse, Validate, Compile
3. Intermediate Form of Query

Diagram:
- Query in High Level Language → Scan, Parse, Validate, Compile → Intermediate Form of Query
Processing Steps

- Query in High Level Language
- Scan, Parse, Validate, Compile
- Intermediate Form of Query
- Optimizer
Processing Steps

1. Query in High Level Language
2. Scan, Parse, Validate, Compile
3. Intermediate Form of Query
4. Execution Plan
5. Optimizer
Processing Steps

- Query in High Level Language
- Scan, Parse, Validate, Compile
- Intermediate Form of Query
- Execution Plan
- Optimizer

Compiler
Processing Steps

1. *Query in High Level Language*
2. *Scan, Parse, Validate, Compile*
3. *Intermediate Form of Query*
4. *Optimizer*
5. *Execution Plan*
6. *Code Generator*
Processing Steps

1. **Query in High Level Language**
2. **Scan, Parse, Validate, Compile**
3. **Intermediate Form of Query**
4. **Optimizer**
5. **Execution Plan**
6. **Code Generator**
7. **Runtime Code**
Processing Steps

- Query in High Level Language
- Scan, Parse, Validate, Compile
- Intermediate Form of Query
- Optimizer
- Execution Plan
- Runtime Processor
- Runtime Code
- Code Generator
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'

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

• $R_1 \leftarrow \text{project \times department}$
• $R_2 \leftarrow R_1 \times \text{employee}$
• $R_3 \leftarrow \sigma_{\text{dnum}=\text{dnumber}}(R_2)$
• $R_4 \leftarrow \sigma_{\text{mgrssn}=\text{e.ssn}}(R_3)$
• $R_5 \leftarrow \sigma_{\text{plocation}=\text{Stafford}}(R_4)$
• $\pi_{\text{pnumber}, \text{dnum}, \text{lname}, \text{address}}(R_5)$
Rel Algebra Query Tree

\[ \text{R1} \leftarrow \text{project} \times \text{department} \]
\[ \text{R2} \leftarrow \text{R1} \times \text{employee} \]
\[ \text{R3} \leftarrow \sigma_{\text{dnum} = \text{dnumber}}(\text{R2}) \]
\[ \text{R4} \leftarrow \sigma_{\text{mgrssn} = \text{e.ssn}}(\text{R3}) \]
\[ \text{R5} \leftarrow \sigma_{\text{plocation} = 'Stafford'}(\text{R4}) \]
\[ \pi_{\text{pnumber}, \text{dnum}, \text{lname}, \text{address}}(\text{R5}) \]

project  
department
R1 ← project × department
R2 ← R1 × employee
R3 ← \sigma_{dnum=dnumber}(R2)
R4 ← \sigma_{mgrssn=e.ssn}(R3)
R5 ← \sigma_{plocation='Stafford'}(R4)
π_{pnumber, dnum, lname, address}(R5)
Rel Algebra Query Tree

• 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

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

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

- \( R_1 \leftarrow \text{project} \times \text{department} \)
- \( R_2 \leftarrow R_1 \times \text{employee} \)
- \( R_3 \leftarrow \sigma_{\text{dnum}=\text{dnumber}}(R_2) \)
- \( R_4 \leftarrow \sigma_{\text{mgrssn}=\text{e.ssn}}(R_3) \)
- \( R_5 \leftarrow \sigma_{\text{plocation}=\text{‘Stafford’}}(R_4) \)
- \( \pi_{\text{pnumber}, \text{dnum}, \text{lname}, \text{address}}(R_5) \)
Rel Algebra Query Tree

• R1 ← project × department
• R2 ← R1 × employee
• R3 ← σ_{dnum=dnumber}(R2)
• R4 ← σ_{mgrssn=e.ssn}(R3)
• R5 ← σ_{plocation='Stafford'}(R4)
• \pi_{pnumber,dnum,lname,address}(R5)
Rel Algebra Query Tree

- \( R1 \leftarrow \text{project} \times \text{department} \)
- \( R2 \leftarrow R1 \times \text{employee} \)
- \( R3 \leftarrow \sigma_{dnum=dnumber}(R2) \)
- \( R4 \leftarrow \sigma_{mgrssn=e.ssn}(R3) \)
- \( R5 \leftarrow \sigma_{plocation='Stafford'}(R4) \)
- \( \pi_{pnumber,dnum,lname,address}(R5) \)
Move Selects Down

\[
\begin{align*}
\text{R1} & \leftarrow \text{project} \times \text{department} \\
\text{R2} & \leftarrow \text{R1} \times \text{employee} \\
\text{R3} & \leftarrow \sigma_{dnum=dnumber}(\text{R2}) \\
\text{R4} & \leftarrow \sigma_{mgrssn=e.ssn}(\text{R3}) \\
\text{R5} & \leftarrow \sigma_{plocation='Stafford'}(\text{R4}) \\
\pi_{pnumber,dnum,lname,address}(\text{R5})
\end{align*}
\]
Introduce Joins

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

- \( R_1 \leftarrow \text{project} \times \text{department} \)
- \( R_2 \leftarrow R_1 \times \text{employee} \)
- \( R_3 \leftarrow \sigma_{dnum=dnrumber}(R_2) \)
- \( R_4 \leftarrow \sigma_{mgrssn=e.ssn}(R_3) \)
- \( R_5 \leftarrow \sigma_{plocation='Stafford'}(R_4) \)
- \( \pi_{\text{pnumber}, \text{dnum}, \text{lname}, \text{address}}(R_5) \)
Original vs Optimized

• $R1 \leftarrow \text{project} \times \text{department}$
• $R2 \leftarrow R1 \times \text{employee}$
• $R3 \leftarrow \sigma_{\text{dnum}=\text{dnumber}}(R2)$
• $R4 \leftarrow \sigma_{\text{mgrssn}=\text{e.ssn}}(R3)$
• $R5 \leftarrow \sigma_{\text{plocation}=\text{‘Stafford’}}(R4)$
• $\pi_{\text{pnumber, dnum, lname, address}}(R5)$

• $R1a \leftarrow \sigma_{\text{plocation}=\text{‘Stafford’}}(\text{project})$
• $R1b \leftarrow \pi_{\text{pnumber, dnum}}(R0a)$
• $R1c \leftarrow \pi_{\text{dnumber, mgrssn}}(\text{department})$
• $R3 \leftarrow R1b \bowtie_{\text{dnum}=\text{dnumber}} R1c$
• $R4a \leftarrow \pi_{\text{ssn, lname, address}}(\text{employee})$
• $R4 \leftarrow R3 \bowtie_{\text{mgrssn}=\text{e.ssn}} R4a$
• $\pi_{\text{pnumber, dnum, lname, address}}(R4)$
Implementing Operations

\[ \pi_{\text{pnumber,dnum}} \]

\[ \sigma_{\text{plocation='Stafford'} } \]

project
Implementing Operations

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

Use Full Table Scan
Implementing Operations

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

- Use Full Table Scan
- Use Index on Plocation
Implementing Operations

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

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>11</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>
```

```
<table>
<thead>
<tr>
<th></th>
<th>9</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</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>12</td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```
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>9</td>
</tr>
<tr>
<td>11</td>
<td></td>
<td>12</td>
</tr>
<tr>
<td>12</td>
<td></td>
<td>12</td>
</tr>
<tr>
<td>13</td>
<td></td>
<td>12</td>
</tr>
<tr>
<td>15</td>
<td></td>
<td>15</td>
</tr>
</tbody>
</table>
Sort Merge Join

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

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
<td>12</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
<td>12</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td></td>
<td>12</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td></td>
<td>15</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Sort Merge Join

- Sort A and B on join fields
- Scan each once to effect join
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>9</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>15</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>11</th>
<th>7</th>
<th>12</th>
<th>8</th>
<th>16</th>
<th>9</th>
<th>13</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table B</td>
<td>16</td>
<td>9</td>
<td>12</td>
<td>8</td>
<td>12</td>
<td>7</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Hash Join

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

Table A

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

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

Bucket 0
- 9
- 12
- 12

Bucket 1
- 16
- 7

Bucket 2
- 8
Hash Join

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

Table A

Table B copied into buckets 0, 1 and 2 based on \( H() \)
Hash Join

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

Table A

| 11 | 7 | 12 | 8 | 16 | 9 | 13 | 15 |

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

Bucket 0

| 9 | 12 | 12 |

Bucket 1

| 16 | 7 |

Bucket 2

| 8 |
Hash Join

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

Table A

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

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  |

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

No match in bucket
Hash Join

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

Table A

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</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>

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

Bucket 0

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

Bucket 1

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

Bucket 2

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td></td>
</tr>
</tbody>
</table>
**Hash Join**

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

Table A

| 11 |
| 7 |
| 12 |
| 8 |
| 16 |
| 9 |
| 13 |
| 15 |

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

- **Bucket 0**
  - 9
  - 12
  - 12

- **Bucket 1**
  - 16
  - 7

- **Bucket 2**
  - 8

\( H(7) = 1 \)

Found match
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