Query Processing

• $Q \rightarrow$ Query Plan

Example:

Select B,D
From R,S
Where R.A = “c” ∧ S.E = 2 ∧ R.C=S.C
How do we execute query?

One idea

- Do Cartesian product
- Select tuples
- Do projection
Relational Algebra - can be used to describe plans...

Plan I

\[ \Pi_{B,D} \left( \sigma_{R.A=\text{"c"}} \land S.E=2 \land R.C=S.C} \right) \times \left( R \times S \right) \]

OR: \[ \Pi_{B,D} \left[ \sigma_{R.A=\text{"c"}} \land S.E=2 \land R.C=S.C } \right] (R \times S) \]
Another idea:

Plan II

\[ \pi_{B,D} \sigma_{R.A = c} \sigma_{S.E = 2} \]

R \quad S

natural join
### R

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>b</td>
<td>1</td>
<td>20</td>
</tr>
<tr>
<td>c</td>
<td>2</td>
<td>10</td>
</tr>
<tr>
<td>d</td>
<td>2</td>
<td>35</td>
</tr>
<tr>
<td>e</td>
<td>3</td>
<td>45</td>
</tr>
</tbody>
</table>

### \(\sigma(R)\)

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>c</td>
<td>2</td>
<td>10</td>
</tr>
</tbody>
</table>

### \(\sigma(S)\)

<table>
<thead>
<tr>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>x</td>
<td>2</td>
</tr>
<tr>
<td>20</td>
<td>y</td>
<td>2</td>
</tr>
<tr>
<td>30</td>
<td>z</td>
<td>2</td>
</tr>
<tr>
<td>40</td>
<td>x</td>
<td>1</td>
</tr>
<tr>
<td>50</td>
<td>y</td>
<td>3</td>
</tr>
</tbody>
</table>

### S

<table>
<thead>
<tr>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>x</td>
<td>2</td>
</tr>
<tr>
<td>20</td>
<td>y</td>
<td>2</td>
</tr>
<tr>
<td>30</td>
<td>z</td>
<td>2</td>
</tr>
<tr>
<td>40</td>
<td>x</td>
<td>1</td>
</tr>
<tr>
<td>50</td>
<td>y</td>
<td>3</td>
</tr>
</tbody>
</table>
Plan III

Use R.A and S.C Indexes

(1) Use R.A index to select R tuples with R.A = “c”

(2) For each R.C value found, use S.C index to find matching tuples

(3) Eliminate S tuples S.E ≠ 2

(4) Join matching R,S tuples, project B,D attributes and place in result
### R

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>b</td>
<td>1</td>
<td>20</td>
</tr>
<tr>
<td>c</td>
<td>2</td>
<td>10</td>
</tr>
<tr>
<td>d</td>
<td>2</td>
<td>35</td>
</tr>
<tr>
<td>e</td>
<td>3</td>
<td>45</td>
</tr>
</tbody>
</table>

### S

<table>
<thead>
<tr>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>x</td>
<td>2</td>
</tr>
<tr>
<td>20</td>
<td>y</td>
<td>2</td>
</tr>
<tr>
<td>30</td>
<td>z</td>
<td>2</td>
</tr>
<tr>
<td>40</td>
<td>x</td>
<td>1</td>
</tr>
<tr>
<td>50</td>
<td>y</td>
<td>3</td>
</tr>
</tbody>
</table>

The diagram shows a query processing operation with conditions and outputs as follows:

- **A** = “c”
- **C**
  - `<c,2,10>`
  - `<10,x,2>`
- Check if `check = 2`?
  - Output: `<2,x>`
- Next tuple: `<c,7,15>`
<p>Query Processing</p>

SQL query → parse → parse tree → convert → logical query plan → apply laws → "improved" l.q.p

estimate result sizes → l.q.p. + sizes

More-Improved l.q.p → Enumerate physical plans

{(P1,C1),(P2,C2)…} → estimate costs → pick best → Pi → answer
Key Steps in Query Processing

Logical query plan(s)
→ Physical query plans
→ “Best” Physical plan.

- Improving logical/physical plans requires cost estimation – which in turn requires size estimation of intermediate results.
- Size estimation techniques: Keep data statistics, and use certain models (later slides).
Example

SELECT title
FROM StarsIn
WHERE starName IN (SELECT name
    FROM MovieStar
    WHERE birthdate LIKE '%%1960');

(Find the movies with stars born in 1960)
Example: Parse Tree

\(<\text{Query}>)
\(<\text{SFW}>\)

SELECT  \(<\text{SelList}>\)  FROM  \(<\text{FromList}>\)  WHERE  \(<\text{Condition}>\)

\(<\text{Attribute}>\)  \(<\text{RelName}>\)  \(<\text{Tuple}>\)  IN  \(<\text{Query}>\)

\text{title}  \text{StarsIn}  \text{starName}

\select\text{name}  \text{MovieStar}  \text{birthDate}\text{\%1960}'
Example: Generating Relational Algebra

\[ \Pi_{\text{title}} \]

\[ \sigma \]

\[ \text{StarsIn} \quad \text{<condition>} \]

\[ \text{<tuple>} \quad \text{IN} \quad \Pi_{\text{name}} \]

\[ \text{<attribute>} \quad \sigma_{\text{birthdate LIKE \text{‘%1960’}}} \]

\[ \text{starName} \quad \text{MovieStar} \]
Example: Logical Query Plan

\[ \Pi_{\text{title}} \sigma_{\text{starName}=\text{name}} \times \Pi_{\text{name}} \sigma_{\text{birthdate} \text{ LIKE} \ '%1960'} \times \Pi_{\text{name}} \sigma_{\text{birthdate} \text{ LIKE} \ '%1960'} \times \Pi_{\text{MovieStar}} \]
Example: Improved Logical Query Plan

\[ \Pi_{\text{title}} \]
\[ \star \text{starName}=\text{name} \]
\[ \Pi_{\text{name}} \]
\[ \sigma \text{birthdate LIKE `1960`} \]
\[ \text{MovieStar} \]

Question: Push project to StarsIn?
Example: Estimate Result Sizes

StarsIn

Need expected size

\[ \Pi \sigma \]

MovieStar
Example: One Physical Plan

Hash join

SEQ scan
StarsIn

Parameters: join order, memory size, ...

index scan
MovieStar

Parameters: Select Condition,...
Example: Estimate costs

\[
\begin{align*}
& \text{L.Q.P} \\
& P_1 \quad P_2 \quad \ldots \quad P_n \\
& \quad C_1 \quad C_2 \quad \ldots \quad C_n
\end{align*}
\]

Pick best!
Query Processing

SQL query

<table>
<thead>
<tr>
<th>parse</th>
<th>parse tree</th>
</tr>
</thead>
<tbody>
<tr>
<td>convert</td>
<td></td>
</tr>
<tr>
<td>logical query plan</td>
<td></td>
</tr>
<tr>
<td>apply laws</td>
<td></td>
</tr>
<tr>
<td>&quot;improved&quot; l.q.p</td>
<td></td>
</tr>
<tr>
<td>estimate result sizes</td>
<td></td>
</tr>
<tr>
<td>l.q.p. + sizes</td>
<td></td>
</tr>
<tr>
<td>More-Improved l.q.p</td>
<td></td>
</tr>
</tbody>
</table>

More-Improved l.q.p

Enumerate physical plans

{(P1,C1),(P2,C2),...} → estimate costs

pick best

execute

Pi

answer

statistics
Rules: Joins, products, union

\[ R \bowtie S = S \bowtie R \]
\[(R \bowtie S) \bowtie T = R \bowtie (S \bowtie T) \]

\[ R \times S = S \times R \]
\[(R \times S) \times T = R \times (S \times T) \]

\[ R \cup S = S \cup R \]
\[ R \cup (S \cup T) = (R \cup S) \cup T \]
**Rules: Selects**

\[ \sigma_{p_1 \land p_2}(R) = \sigma_{p_1} \left[ \sigma_{p_2}(R) \right] \]

\[ \sigma_{p_1 \lor p_2}(R) = \left[ \sigma_{p_1}(R) \right] U \left[ \sigma_{p_2}(R) \right] \]
Rules: \( \sigma + \bowtie \) combined

Let \( p = \) predicate with only R attribs
\( q = \) predicate with only S attribs
\( m = \) predicate with only R, S attribs

\[
\begin{align*}
\sigma_p (R \bowtie S) &= [\sigma_p (R)] \bowtie S \\
\sigma_q (R \bowtie S) &= R \bowtie [\sigma_q (S)]
\end{align*}
\]
Derived Rules: $\sigma + \bowtie$ combined

\[ \sigma_{p \land q} (R \bowtie S) = [\sigma_p (R)] \bowtie [\sigma_q (S)] \]

\[ \sigma_{p \land q \land m} (R \bowtie S) = \]

\[ \sigma_{m} \left[ (\sigma_p R) \bowtie (\sigma_q S) \right] \]

\[ \sigma_{p v q} (R \bowtie S) = \]

\[ \left[ (\sigma_p R) \bowtie S \right] \cup \left[ R \bowtie (\sigma_q S) \right] \]
Rules: $\pi, \sigma$ combined

Let $x =$ subset of $R$ attributes
$z =$ attributes in predicate $P$
(subset of $R$ attributes)

$$\pi_x[\sigma_p(R)] = \pi_x\{\sigma_p[\pi_x(R)]\}$$
Rules: $\sigma$, $U$ combined

$\sigma_p(R \cup S) = \sigma_p(R) \cup \sigma_p(S)$

$\sigma_p(R - S) = \sigma_p(R) - S = \sigma_p(R) - \sigma_p(S)$
Which are “good” transformations?

- \( \sigma_{p1 \land p2} (R) \rightarrow \sigma_{p1} [\sigma_{p2} (R)] \)
- \( \sigma_p (R \bowtie S) \rightarrow [\sigma_p (R)] \bowtie S \)
- \( R \bowtie S \rightarrow S \bowtie R \)
- \( \pi_x [\sigma_p (R)] \rightarrow \pi_x \{ \sigma_p [\pi_{xz} (R)] \} \)
Bottom line:

- No transformation is *always* good
- Usually good: early selections
**Query Processing**

1. **SQL query**
   - **Parse**
     - Parse tree
   - **Convert**
     - Logical query plan
   - **Apply laws**
     - "Improved" l.q.p
   - **Estimate result sizes**
     - l.q.p. + sizes
     - More-Improved l.q.p
   - **Enumerate physical plans**
     - \( \{P_1, P_2, \ldots\} \)

2. **Estimate costs**
   - \( \{(P_1, C_1), (P_2, C_2), \ldots\} \)
   - **Pick best**
   - **Execute**
   - **Answer**
Estimating result size

- Keep statistics for relation R
  - $T(R)$: # tuples in R
  - $S(R)$: # bytes in each R tuple
  - $B(R)$: # blocks to hold all R tuples
  - $V(R, A)$: # distinct values in R for attribute A
**Example**

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>cat</td>
<td>1</td>
<td>10</td>
<td>a</td>
</tr>
<tr>
<td>cat</td>
<td>1</td>
<td>20</td>
<td>b</td>
</tr>
<tr>
<td>dog</td>
<td>1</td>
<td>30</td>
<td>a</td>
</tr>
<tr>
<td>dog</td>
<td>1</td>
<td>40</td>
<td>c</td>
</tr>
<tr>
<td>bat</td>
<td>1</td>
<td>50</td>
<td>d</td>
</tr>
</tbody>
</table>

A: 20 byte string  
B: 4 byte integer  
C: 8 byte date  
D: 5 byte string

\[ T(R) = 5 \quad S(R) = 37 \]

\[ V(R,A) = 3 \quad V(R,C) = 5 \]

\[ V(R,B) = 1 \quad V(R,D) = 4 \]
Size estimates for $W = R_1 \times R_2$

\[ T(W) = T(R_1) \times T(R_2) \]

\[ S(W) = S(R_1) + S(R_2) \]
Size estimate for $W = \sigma_{A=a}(R)$

$S(W) = S(R)$

$T(W) = ?$
Example

\[ W = \sigma_{z=\text{val}(R)}; \quad T(W) = \frac{T(R)}{V(R,Z)} \]

(under assumption of uniform distribution of values of \( Z \) over \( V(R,Z) \))
\[ W = \sigma_{z \geq \text{val}} (R) \]. \quad T(W) = f \times T(R)

<table>
<thead>
<tr>
<th>R</th>
<th>Z</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Min = 1 \quad V(R, Z) = 10
Max = 20

\[ f = \frac{20 - 15 + 1}{20} = \frac{6}{20} \] (fraction of range)
Size estimate for $W = R_1 \times \square R_2$

Let $x =$ attributes of $R_1$
$y =$ attributes of $R_2$

Case 1

$X \cap Y = \emptyset$

Same as $R_1 \times R_2$
Case 2

\[ W = R_1 \Join R_2 \quad X \cap Y = A \]

<table>
<thead>
<tr>
<th>R1</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>R2</th>
<th>A</th>
<th>D</th>
</tr>
</thead>
</table>

Assumption (containment of values):

\[ V(R_1, A) \leq V(R_2, A) \implies \text{Every A value in R1 is in R2} \]
\[ V(R_2, A) \leq V(R_1, A) \implies \text{Every A value in R2 is in R1} \]
### Computing $T(W)$ when $V(R1,A) \leq V(R2,A)$

<table>
<thead>
<tr>
<th>R1</th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>R2</th>
<th>A</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Take 1 tuple**

1 tuple matches with $\frac{T(R2)}{V(R2,A)}$ tuples...

so $T(W) = \frac{T(R2) \times T(R1)}{V(R2, A)}$
• \( V(R_1,A) \leq V(R_2,A) \)  \( T(W) = T(R_2) T(R_1) \frac{V(R_2,A)}{V(R_1,A)} \)

• \( V(R_2,A) \leq V(R_1,A) \)  \( T(W) = T(R_2) T(R_1) \frac{V(R_2,A)}{V(R_1,A)} \)

[A is common attribute]
In general, \( W = R_1 \bowtie R_2 \)

\[
T(W) = \frac{T(R_2) T(R_1)}{\max \{ V(R_1, A), V(R_2, A) \}}
\]
In all cases:

$$S(W) = S(R1) + S(R2) - S(A)$$

size of attribute A
Similarly, we can estimate sizes of:

\[ \Pi_{AB} (R); \quad \sigma_{A=a \land B=b} (R); \]

\[ R \bowtie S \] with common attributes. A,B,C;

Union, intersection, diff, .... Sec. 16.4.7
Note: for complex expressions, need intermediate T,S,V results.

E.g. \( W = \left[ \sigma_{A=a} (R1) \right] \bowtie R2 \)

Treat as relation \( U \)

\[
\begin{align*}
T(U) & = T(R1)/V(R1,A) \\
S(U) & = S(R1) \\
V(U,\ast) & = ?? \quad \text{(Needed for } T(W)\text{)}
\end{align*}
\]
To estimate $V_s$

E.g., $U = \sigma_{A=a}(R1)$

Say $R1$ has attributes $A,B,C,D$

$V(U, A) = 1$  \hspace{1cm} (exact)

$V(U, B) = V(R1, B)$  \hspace{1cm} (guess)

$V(U, C) = V(R1,C)$  \hspace{1cm} (guess)

$V(U, D) = V(R1,D)$  \hspace{1cm} (guess)
For Joins \[ U = R1(A,B) \bowtie R2(A,C) \]

\[
V(U,A) = \min \{ V(R1, A), V(R2, A) \}
\]
\[
V(U,B) = V(R1, B)
\]
\[
V(U,C) = V(R2, C)
\]

[called “preservation of value sets” assumption]
Example:

$$Z = R_1(A,B) \bowtie R_2(B,C) \bowtie R_3(C,D)$$

<table>
<thead>
<tr>
<th>R1</th>
<th>T(R1) = 1000</th>
<th>V(R1,A)=50</th>
<th>V(R1,B)=100</th>
</tr>
</thead>
<tbody>
<tr>
<td>R2</td>
<td>T(R2) = 2000</td>
<td>V(R2,B)=200</td>
<td>V(R2,C)=300</td>
</tr>
<tr>
<td>R3</td>
<td>T(R3) = 3000</td>
<td>V(R3,C)=90</td>
<td>V(R3,D)=500</td>
</tr>
</tbody>
</table>
Partial Result:  \( U = R \bowtie S \)

\[
\begin{align*}
T(U) &= 1000 \times 2000 \\
&= 200 \\
V(U,A) &= 50 \\
V(U,B) &= 100 \\
V(U,C) &= 300
\end{align*}
\]
$$Z = U \bowtie R3$$

$$T(Z) = \frac{1000 \times 2000 \times 3000}{200 \times 300}$$

\begin{align*}
V(Z,A) &= 50 \\
V(Z,B) &= 100 \\
V(Z,C) &= 90 \\
V(Z,D) &= 500
\end{align*}
Recap

• Estimating size of results is an “art”
• Computing Statistics: Scan, Sample.
• Also, statistics must be kept up to date…

• Other Statistics:
  – Histograms (equi-height, equi-width, most-freq.)
• Next: Using size estimates to improve an LQP.
Improve LPQ with Size

- Size estimates can also help improve the LQP.
  - But, need some heuristic to estimate costs.
  - Good heuristic: Cost = Sum of intermediate-result sizes.
  - Next Slide: Example.
Ex: Improve LQP with Sizes

- $R(a,b)$: $T(R)=5000$, $V(R,a)=50$, $V(R,b)=100$
- $S(b,c)$: $T(S) = 2000$, $V(S,b)=200$, $V(S,c)=100$.

$\delta(\sigma_p(R \ JOIN \ S))$. Where $p$ is $(R.a=20)$.

Two choices:
1. $\delta(\sigma_p(R)) \ JOIN (\delta (S))$
2. $\delta(\sigma_p(R) \ JOIN \ S)$

Cost Model = Sum of the sizes of intermediate results. [1100 vs 1150]
Query Processing

1. SQL query
2. Parse
   - Parse tree
3. Convert
4. Logical query plan
5. Apply laws
   - "Improved" L.Q.P.
6. Estimate result sizes
   - L.Q.P. + sizes
7. More-Improved L.Q.P.
8. Enumerate physical plans
   - \{P1,P2,\ldots\}
9. Estimate costs
10. Pick best
11. Execute
12. Answer

Statistics

\{\,(P1,C1),(P2,C2),\ldots,\}\
Chapter 15

SQL query

parse

parse tree

convert

logical query plan

apply laws

“improved” l.q.p

estimate result sizes

l.q.p. + sizes

More-Improved l.q.p

Enumerate physical plans

\{P1, P2, \ldots\}

\{(P1, C1), (P2, C2), \ldots\}

Pi

pick best

estimate costs

execute

answer

statistics

Himanshu Gupta
SQL query → parse → parse tree → convert → logical query plan → apply laws → “improved” l.q.p → estimate result sizes → l.q.p. + sizes → More-Improved l.q.p

Done (previous slides)

Pi

answer

statistics

execute

pick best

{P1,P2,.....}

Enumerate physical plans

estimate costs
SQL query

parse

parse tree

convert

logical query plan

apply laws

“improved” l.q.p

estimate result sizes

l.q.p. + sizes

More-Improved l.q.p

Enumerate physical plans

{(P1,C1),(P2,C2).....}

answer

pi

execute

pick best

estimate costs

statistics

Next few slides …
LQP vs. PQP

• LQP is a high-level expression-tree.
• PQP is (LQP + more execution details such as):
  – Order/grouping of join, union, intersection, etc.
  – Choice of join algorithm, etc.
  – Additional operators such as scanning, sorting, etc.
  – Intermediate steps between two operations. E.g., sorting, storage, pipelining, etc.
Generating and comparing physical plans

LQP $\rightarrow$ PQP

Generate
Pruning
Estimate Cost
Select

Pick Min
PQP Enumeration Heuristics

1. Various techniques (details skipped):
   - Top-down
   - Bottom-up
   - Heuristics
   - Branch and Bound
   - Hill Climbing
   - Dynamic Programming
   - Selinger-style Optimization (Improved DP)
Join Ordering

• R1 x R2 x R3 x ….. Rn (join of n tables).

• How many possible orderings?
• Join-selectivity.
• Left-deep, Right-deep trees.
• Dynamic Programming approach.
PQP Selection Example

Pipelining vs. Materialization

• (R join S) join U
• 5000, 10000, 10000 blocks respectively.
• (R join S) is of size k (we’ll consider different k)
• We’ll only use hash-joins
• Memory size: 101 blocks.
Example (contd)

• (R join S) join U.

R join S

• 100 buckets at most.

• R-buckets have 50 tuples each. So, need 51 buffers for the second pass.

• **Pipeline:** Use remaining 50 buffers to join the result with U.
  
  – If k < 49, then keep (R join S) in memory, and read U one block at a time.
Example (continued)

- If \( k < 49 \), then read \( U \) one block at a time, and do everything in main memory. \( \text{Cost} = 55k \).
- If \( k > 49 \), but \( < 5000 \):
  - First, hash all tables. \( \text{Hash U with 50 buckets} \).
  - When doing \( R \) join \( S \), use the remaining 50 buffers to “hash” \( (R \) join \( S) \) result into the 50 buckets.
  - Each bucket of \( (R \) join \( S) \) is of size 100.
  - Then, do two-pass hash join of \( (R \) join \( S) \) with \( U \).
  - \( \text{Cost} = 50000 + 15000 + (k + (10000 + k)) \)
Example – contd.

• If $k > 5000$:
  – Last step will require a 3-pass join, since each bucket of $(R \text{ join } S)$ as well as $U$ is more than 100 blocks (buckets of $U$ are 200 blocks each).
  – Better: **No pipelining.** Write $(R \text{ join } S)$. Then, do 2-pass hash-join with $U$ (now the buckets of $U$ are 100 blocks each).

Cost $= 45000 + k + 3(10000 + k) = 75000 + 4k$