Lecture 25:
Monday, December 27, 2002

Administrative

• Homework 5 is due on Monday, 12/9
• Project demos will be on Tuesday 12/10
  – 10am – 12pm: 6 teams
  – 2pm – 4pm: 6 teams
  – 4pm – 6pm: 6 teams
• Please send me email for an appointment
  – First come first served...

Outline

• Cost-based optimization: 16.5, 16.6
• Completing the physical query plan: 16.7
• Cost estimation: 16.4 (will continue next time)

Cost-based Optimizations

• Main idea: apply algebraic laws, until estimated cost is minimal
• Practically: start from partial plans, introduce operators one by one
  – Will see in a few slides
• Problem: there are too many ways to apply the laws, hence too many (partial) plans

Cost-based Optimizations

Approaches:

• Top-down: the partial plan is a top fragment of the logical plan
• Bottom up: the partial plan is a bottom fragment of the logical plan

Search Strategies

• Branch-and-bound:
  – Remember the cheapest complete plan P seen so far and its cost C
  – Stop generating partial plans whose cost is > C
  – If a cheaper complete plan is found, replace P, C
• Hill climbing:
  – Remember only the cheapest partial plan seen so far
• Dynamic programming:
  – Remember the all cheapest partial plans
Dynamic Programming

Unit of Optimization: select-project-join
- Push selections down, pull projections up

Join Trees

- $R_1 \bowtie R_2 \bowtie \ldots \bowtie R_n$
- Join tree:

```
      R1
     /   \
    R2   R3
   /     /
  R4     R5
```
- A plan = a join tree
- A partial plan = a subtree of a join tree

Types of Join Trees

- Left deep:

```
  R3
  |
  R1
```

- Bushy:

```
  R3
  |
  R1
```

Types of Join Trees

- Right deep:

```
  R3
  |
  R1
```

Problem

- Given: a query $R_1 \bowtie R_2 \bowtie \ldots \bowtie R_n$
- Assume we have a function cost() that gives us the cost of every join tree
- Find the best join tree for the query
Dynamic Programming

- Idea: for each subset of \{R_1, \ldots, R_n\}, compute the best plan for that subset
- In increasing order of set cardinality:
  - Step 1: for \{R_1\}, \{R_2\}, \ldots, \{R_n\}
  - Step 2: for \{R_1, R_2\}, \{R_1, R_3\}, \ldots, \{R_{n-1}, R_n\}
  - \ldots
  - Step n: for \{R_1, \ldots, R_n\}
- It is a bottom-up strategy
- A subset of \{R_1, \ldots, R_n\} is also called a subquery

Dynamic Programming

- Step 1: For each \{R_i\} do:
  - Size(\{R_i\}) = B(\{R_i\})
  - Plan(\{R_i\}) = \emptyset
  - Cost(\{R_i\}) = \text{(cost of scanning } R_i)\text{)

Dynamic Programming

- Step i: For each \{R_1, \ldots, R_n\} of cardinality i do:
  - Compute Size(Q) (later…)
  - For every pair of subqueries Q’, Q’’
    s.t. Q = Q’ \cup Q’’
    compute cost(\text{Plan}(Q’’))
  - Cost(Q) = the smallest such cost
  - Plan(Q) = the corresponding plan

Dynamic Programming

- Return Plan(\{R_1, \ldots, R_n\})

Dynamic Programming

To illustrate, we will make the following simplifications:
- Cost(P_1 \parallel P_2) = Cost(P_1) + Cost(P_2) + size(\text{intermediate result(s)})
- Intermediate results:
  - If P_1 \parallel \text{a join}, then the size of the intermediate result is size(P_1), otherwise the size is 0
  - Similarly for P_2
- Cost of a scan = 0
Dynamic Programming

- Example:
- \( \text{Cost}(R5 \bowtie R7) = 0 \) (no intermediate results)
- \( \text{Cost}(R2 \bowtie R1) \bowtie R7) \)
  \[= \text{Cost}(R2 \bowtie R1) + \text{Cost}(R7) + \text{size}(R2 \bowtie R1)\]
  \[= \text{size}(R2 \bowtie R1)\]

Dynamic Programming

- Relations: R, S, T, U
- Number of tuples: 2000, 5000, 3000, 1000
- Size estimation: \( T(A \bowtie B) = 0.01 \times T(A) \times T(B) \)

<table>
<thead>
<tr>
<th>Subquery</th>
<th>Size</th>
<th>Cost</th>
<th>Plan</th>
</tr>
</thead>
<tbody>
<tr>
<td>RS</td>
<td></td>
<td></td>
<td>RS</td>
</tr>
<tr>
<td>RT</td>
<td></td>
<td></td>
<td>RT</td>
</tr>
<tr>
<td>RU</td>
<td></td>
<td></td>
<td>RU</td>
</tr>
<tr>
<td>ST</td>
<td></td>
<td></td>
<td>ST</td>
</tr>
<tr>
<td>SU</td>
<td></td>
<td></td>
<td>SU</td>
</tr>
<tr>
<td>TU</td>
<td></td>
<td></td>
<td>TU</td>
</tr>
<tr>
<td>RST</td>
<td></td>
<td></td>
<td>RST</td>
</tr>
<tr>
<td>RSTU</td>
<td></td>
<td></td>
<td>RSTU</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Subquery</th>
<th>Size</th>
<th>Cost</th>
<th>Plan</th>
</tr>
</thead>
<tbody>
<tr>
<td>RS</td>
<td>10k</td>
<td></td>
<td>RS</td>
</tr>
<tr>
<td>RT</td>
<td>20k</td>
<td></td>
<td>RT</td>
</tr>
<tr>
<td>RU</td>
<td>20k</td>
<td></td>
<td>RU</td>
</tr>
<tr>
<td>ST</td>
<td>10k</td>
<td></td>
<td>ST</td>
</tr>
<tr>
<td>SU</td>
<td>20k</td>
<td></td>
<td>SU</td>
</tr>
<tr>
<td>TU</td>
<td>10k</td>
<td></td>
<td>TU</td>
</tr>
<tr>
<td>RST</td>
<td>30k</td>
<td></td>
<td>RST</td>
</tr>
<tr>
<td>RSTU</td>
<td>15k</td>
<td>20k</td>
<td>RSTU</td>
</tr>
<tr>
<td>RSTU</td>
<td>0.06k</td>
<td>20k</td>
<td>RSTU</td>
</tr>
<tr>
<td>STU</td>
<td>1.3k</td>
<td>20k</td>
<td>STU</td>
</tr>
<tr>
<td>RSTU</td>
<td>10M</td>
<td>50k-100k</td>
<td>RSTU</td>
</tr>
</tbody>
</table>

Dynamic Programming

- Summary: computes optimal plans for subqueries:
  - Step 1: \( \{R1, R2, \ldots, Rn\} \)
  - Step 2: \( \{R1, R2\}, \{R1, R3\}, \ldots, \{Rn-1, Rn\} \)
  - ...  
  - Step n: \( \{R1, \ldots, Rn\} \)
- We used naïve size/cost estimations
- In practice:
  - More realistic size/cost estimations (next time)
  - Heuristics for reducing the search space
  - Restrict to left linear trees
  - Restrict to trees “without cartesian product”
  - Need more than just one plan for each subquery:
    - "interesting orders"

Completing the Physical Query Plan

- Choose algorithm to implement each operator
  - Need to account for more than cost:
    - How much memory do we have?
    - Are the input operand(s) sorted?
- Decide for each intermediate result:
  - To materialize
  - To pipeline
Materialize Intermediate Results Between Operators

Question in class
Given B(R), B(S), B(T), B(U)
- What is the total cost of the plan?
  - Cost =
- How much main memory do we need?
  - M =

Pipeline Between Operators

Question in class
Given B(R), B(S), B(T), B(U)
- What is the total cost of the plan?
  - Cost =
- How much main memory do we need?
  - M =

Completing the Physical Query Plan

- Choose algorithm to implement each operator
  - Need to account for more than cost:
    - How much memory do we have?
    - Are the input operand(s) sorted?
- Decide for each intermediate result:
  - To materialize
  - To pipeline

Example 16.36

- Logical plan:
  - k blocks
  - 10 blocks
  - 10,000 blocks

- Main memory M = 101 buffers
Example 16.36

```
\begin{center}
\begin{tikzpicture}
  \node (R) at (0,0) {\text{R(w,x)}};
  \node (S) at (4,0) {\text{S(x,y)}};
  \node (T) at (2,2) {\text{T}};
  \draw[->] (R) -- (T) node[midway,above] {\text{k blocks}};
  \draw[->] (S) -- (T) node[midway,above] {\text{U(y,z)} 10,000 blocks};
  \draw[->] (R) -- (S) node[midway,above] {\text{R(w,x) 5,000 blocks S(x,y) 10,000 blocks}};
\end{tikzpicture}
\end{center}
```

Naïve evaluation:
- 2 partitioned hash-joins
- Cost $3B(R) + 3B(S) + 4k + 3B(U) = 75,000 + 4k$

Example 16.36

```
\begin{center}
\begin{tikzpicture}
  \node (R) at (0,0) {\text{R(w,x)}};
  \node (S) at (4,0) {\text{S(x,y)}};
  \node (T) at (2,2) {\text{T}};
  \draw[->] (R) -- (T) node[midway,above] {\text{k blocks}};
  \draw[->] (S) -- (T) node[midway,above] {\text{U(y,z) 10,000 blocks}};
  \draw[->] (R) -- (S) node[midway,above] {\text{R(w,x) 5,000 blocks S(x,y) 10,000 blocks}};
\end{tikzpicture}
\end{center}
```

Smarter:
- Step 1: hash R on x into 100 buckets, each of 50 blocks to disk
- Step 2: hash S on x into 100 buckets to disk
- Step 3: read each R in memory (50 buffers) join with Si (1 buffer); hash result on y into 50 buckets (50 buffers); -- here we pipeline
- Cost so far: $3B(R) + 3B(S)$

Example 16.36

```
\begin{center}
\begin{tikzpicture}
  \node (R) at (0,0) {\text{R(w,x)}};
  \node (S) at (4,0) {\text{S(x,y)}};
  \node (T) at (2,2) {\text{T}};
  \draw[->] (R) -- (T) node[midway,above] {\text{k blocks}};
  \draw[->] (S) -- (T) node[midway,above] {\text{U(y,z) 10,000 blocks}};
  \draw[->] (R) -- (S) node[midway,above] {\text{R(w,x) 5,000 blocks S(x,y) 10,000 blocks}};
\end{tikzpicture}
\end{center}
```

Continuing:
- How large are the 50 buckets on y? Answer: k/50.
- If k <= 50 then keep all 50 buckets in Step 3 in memory, then:
- Step 4: read U from disk, hash on y and join with memory
- Total cost: $3B(R) + 3B(S) + B(U) = 55,000$

Example 16.36

```
\begin{center}
\begin{tikzpicture}
  \node (R) at (0,0) {\text{R(w,x)}};
  \node (S) at (4,0) {\text{S(x,y)}};
  \node (T) at (2,2) {\text{T}};
  \draw[->] (R) -- (T) node[midway,above] {\text{k blocks}};
  \draw[->] (S) -- (T) node[midway,above] {\text{U(y,z) 10,000 blocks}};
  \draw[->] (R) -- (S) node[midway,above] {\text{R(w,x) 5,000 blocks S(x,y) 10,000 blocks}};
\end{tikzpicture}
\end{center}
```

Continuing:
- If 50 <= k <= 5000 then send the 50 buckets in Step 3 to disk
  - Each bucket has size k/5000 <= 100
- Step 4: partition U into 50 buckets
- Step 5: read each partition and join in memory
- Total cost: $3B(R) + 3B(S) + 2k + 3B(U) = 75,000 + 2k$

Example 16.36

```
\begin{center}
\begin{tikzpicture}
  \node (R) at (0,0) {\text{R(w,x)}};
  \node (S) at (4,0) {\text{S(x,y)}};
  \node (T) at (2,2) {\text{T}};
  \draw[->] (R) -- (T) node[midway,above] {\text{k blocks}};
  \draw[->] (S) -- (T) node[midway,above] {\text{U(y,z) 10,000 blocks}};
  \draw[->] (R) -- (S) node[midway,above] {\text{R(w,x) 5,000 blocks S(x,y) 10,000 blocks}};
\end{tikzpicture}
\end{center}
```

Continuing:
- If k <= 50, cost = 55,000
- If 50 < k <= 5000, cost = 75,000 + 2k
- If k > 5000, cost = 75,000 + 4k

Example 16.36

```
\begin{center}
\begin{tikzpicture}
  \node (R) at (0,0) {\text{R(w,x)}};
  \node (S) at (4,0) {\text{S(x,y)}};
  \node (T) at (2,2) {\text{T}};
  \draw[->] (R) -- (T) node[midway,above] {\text{k blocks}};
  \draw[->] (S) -- (T) node[midway,above] {\text{U(y,z) 10,000 blocks}};
  \draw[->] (R) -- (S) node[midway,above] {\text{R(w,x) 5,000 blocks S(x,y) 10,000 blocks}};
\end{tikzpicture}
\end{center}
```

Summary:
- If k <= 50, cost = 55,000
- If 50 < k <= 5000, cost = 75,000 + 2k
- If k > 5000, cost = 75,000 + 4k
Estimating Sizes

• Need size in order to estimate cost
• Example:
  – Cost of partitioned hash-join $E_1 \bowtie E_2$ is $3B(E_1) + 3B(E_2)$
  – $B(E_1) = T(E_1) /$ block size
  – $B(E_2) = T(E_2) /$ block size
  – So, we need to estimate $T(E_1), T(E_2)$

Estimating Sizes

Estimating the size of a projection
• Easy: $T(\Pi_k(R)) = T(R)$
• This is because a projection doesn’t eliminate duplicates

Estimating Sizes

Estimating the size of a selection
• $S = \sigma_{A \in S}(R)$
  – $T(S)$ can be anything from 0 to $T(R) - V(R, A) + 1$
  – Mean value: $T(S) = T(R) / V(R, A)$
• $S = \sigma_{A \in S}(R)$
  – $T(S)$ can be anything from 0 to $T(R)$
  – Heuristics: $T(S) = T(R) / 3$

Estimating Sizes

Estimating the size of a natural join, $R \bowtie S$
• When the set of $A$ values are disjoint, then $T(R \bowtie S) = 0$
• When $A$ is a key in $S$ and a foreign key in $R$, then $T(R \bowtie S) = T(R)$
• When $A$ has a unique value, the same in $R$ and $S$, then $T(R \bowtie S) = T(R) \cdot T(S)$

Estimating Sizes

Assumptions:
• Containment of values: if $V(R, A) \subseteq V(S, A)$, then the set of $A$ values of $R$ is included in the set of $A$ values of $S$
  – Note: this indeed holds when $A$ is a foreign key in $R$, and a key in $S$
• Preservation of values: for any other attribute $B$, $V(R \bowtie S, B) = V(R, B)$ (or $V(S, B)$)

Estimating Sizes

Assume $V(R, A) \subseteq V(S, A)$
• Then each tuple $t$ in $R$ joins some tuple(s) in $S$
  – How many?
  – On average $S / V(S, A)$
  – $t$ will contribute $S / V(S, A)$ tuples in $R \bowtie S$
• Hence $T(R \bowtie S) = T(R) \cdot T(S) / V(S, A)$

In general: $T(R \bowtie S) = T(R) \cdot T(S) / \max(V(R, A), V(S, A))$
Estimating Sizes

Example:
- \( T(R) = 10000, \ T(S) = 20000 \)
- \( V(R,A) = 100, \ V(S,A) = 200 \)
- How large is \( R \bowtie_3 S \)?

Answer: \( T(R \bowtie_3 S) = 10000 \times 20000 / 200 = 1M \)

Estimating Sizes

Joins on more than one attribute:
- \( T(R \bowtie_{A,B} S) = T(R) / \max(V(R,A),V(S,A)) \max(V(R,B),V(S,B)) \)

Histories

- Statistics on data maintained by the RDBMS
- Makes size estimation much more accurate (hence, cost estimations are more accurate)

Histories

Employee(name, name, salary, phone)
- Maintain a histogram on salary:

<table>
<thead>
<tr>
<th>Salary</th>
<th>0-20k</th>
<th>20k-40k</th>
<th>40k-60k</th>
<th>60k-80k</th>
<th>80k-100k</th>
<th>&gt; 100k</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tuples</td>
<td>200</td>
<td>600</td>
<td>500</td>
<td>1200</td>
<td>600</td>
<td>500</td>
</tr>
</tbody>
</table>

- \( T(Employee) = 25000 \), but now we know the distribution

Histories

Ranks(rankName, salary)
- Estimate the size of Employee \( \bowtie_{Salary} Ranks \)

<table>
<thead>
<tr>
<th>Employee</th>
<th>0-20k</th>
<th>20k-40k</th>
<th>40k-60k</th>
<th>60k-80k</th>
<th>80k-100k</th>
<th>&gt; 100k</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tuples</td>
<td>200</td>
<td>600</td>
<td>500</td>
<td>1200</td>
<td>600</td>
<td>500</td>
</tr>
</tbody>
</table>

Histories

- Assume:
  - \( V(Employee, Salary) = 200 \)
  - \( V(Ranks, Salary) = 250 \)
- Then \( T(Employee \bowtie_{Salary} Ranks) = \sum_{i=1}^{6} T_i \ T'_i / 250 \)

\[
= (200 \times 8 + 800 \times 20 + 5000 \times 40 + 12000 \times 80 + 6500 \times 100 + 500 \times 2)/250 \\
= \ldots.
\]