Lecture 26:
Wednesday, December 4, 2002

Outline

• Cost-based Query Optimization
• Completing the physical query plan: 16.7
• Cost estimation: 16.4

Reducing the Search Space

• Left-linear trees vs. Bushy trees
• Trees without cartesian product

Example: \( R(A,B) \Join S(B,C) \Join T(C,D) \)

Plan: \( (R(A,B) \Join T(C,D)) \Join S(B,C) \)

Counting the Number of Join Orders

• The mathematics we need to know:

Given \( x_1, x_2, ..., x_n \), how many permutations can we construct? Answer: \( n! \)

• Example: permutations of \( x_1 x_2 x_3 x_4 \) are:
  \( x_1 x_2 x_3 x_4 \)
  \( x_2 x_1 x_3 x_4 \)
  \( x_3 x_1 x_2 x_4 \)
  \( x_4 x_1 x_2 x_3 \)

  (there are 4! = 4*3*2*1 = 24 permutations)

Counting the Number of Join Orders

• The mathematics we need to know:

Given the product \( x_1 x_2 x_3 ... x_n \), in how many ways can we place \( n \) pairs of parenthesis around them? Answer:

\[ \frac{1}{n+1} \binom{2n+1}{n} = \frac{(2n)!}{(n+1)!(n)!} \]

• Example: for \( n = 3 \)
  
  \( ((x_1 x_2) x_3) \)
  \( (x_1 (x_2 x_3)) \)
  \( (x_1 x_2) x_3) \)
  \( (x_1 x_2) (x_3) \)
  \( (x_1) (x_2 x_3)) \)

  (there are 6*(4*3*2*1) = 5 ways)
Counting the Number of Join Orders (Exercise)
\[ R_d(A_{0:a}) \land R_1(A_{1:a}) \land \ldots \land R_d(A_{d:a}) \]
- The number of left linear join trees is:
- The number of left linear join trees without cartesian products is:
- The number of bushy join trees is:
- The number of bushy join trees without cartesian product is:

Number of Subplans Inspected by Dynamic Programming
\[ R_d(A_{0:a}) \land R_1(A_{1:a}) \land \ldots \land R_d(A_{d:a}) \]
- The number of left linear subplans inspected is:
- The number of left linear subplans without cartesian products inspected is:
- The number of bushy join subplans inspected is:
- The number of bushy join subplans without cartesian product:

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
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 =

Example 16.36

- Logical plan is:
  - k blocks
    - R(w,x)
      - 5,000 blocks
    - S(x,y)
      - 10,000 blocks

- Main memory M = 101 buffers

Example 16.36

M = 101

Naïve evaluation:
- 2 partitioned hash-joins
- Cost 3B(R) + 3B(S) + 4k + 3B(U) = 75000 + 4k

Example 16.36

M = 101

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_i in memory (50 buffers) join with S_y (1 buffer); hash result on y into 50 buckets (50 buffers) — here we pipeline
- Cost so far: 3B(R) + 3B(S)

Continuing:
- How large are the 50 buckets on y? Answer: k/50.
- If k <= 50 then keep all 50 buckets 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

M = 101

\[
\begin{array}{c}
\text{k blocks} \\
R(w,x) \\
S(x,y) \\
U(y,z) \\
\end{array}
\]

10,000 blocks
5,000 blocks
Continuing:

- If \(50 < k \leq 5000\) then send the 50 buckets in Step 3 to disk
  - Each bucket has size \(k/50 \leq 100\)
- 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

Summary:

- If \(k \leq 50\), \(\text{cost} = 55,000\)
- If \(50 < k \leq 5000\), \(\text{cost} = 75,000 + 2k\)
- If \(k > 5000\), \(\text{cost} = 75,000 + 4k\)

Estimating Sizes

Estimating the size of a projection

- Easy: \(T(\Pi_i(R)) = T(R)\)
- This is because a projection doesn’t eliminate duplicates

Estimating the size of a selection

- \(S = \sigma_{A_i \leq a_j}(R)\)
  - \(T(S)\) can be anything from 0 to \(T(R) - V(R,A) + 1\)
  - Mean value: \(T(S) = T(R) \cdot V(R,A)\)
- \(S = \sigma_{A_i \geq a_j}(R)\)
  - \(T(S)\) can be anything from 0 to \(T(R)\)
  - Heuristics: \(T(S) = T(R)/3\)
Estimating Sizes

Estimating the size of a natural join, \( R \bowtie_A S \)

- When the set of \( A \) values are disjoint, then \( T(R \bowtie_A S) = 0 \)
- When \( A \) is a key in \( S \) and a foreign key in \( R \), then \( T(R \bowtie_A S) = T(R) \)
- When \( A \) has a unique value, the same in \( R \) and \( S \), then \( T(R \bowtie_A S) = T(R) \cdot T(S) \)

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 \( B \), \( V(R \bowtie_A S, B) = V(R, B) \) (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 \( T(S)/V(S,A) \)
  - \( t \) will contribute \( T(S)/V(S,A) \) tuples in \( R \bowtie_A S \)
- Hence \( T(R \bowtie_A S) = T(R) \cdot T(S) / V(S,A) \)

In general: \( T(R \bowtie_A S) = T(R) \cdot T(S) / \max(V(R,A),V(S,A)) \)

Estimating Sizes

Example:

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

Answer: \( T(R \bowtie_A S) = 10000 \cdot 20000 / 200 = 1M \)

Estimating Sizes

Joins on more than one attribute:

- \( T(R \bowtie_{A,B} S) = \)
  \[ T(R) \cdot T(S) / (\max(V(R,A),V(S,A)) \cdot \max(V(R,B),V(S,B))) \]

Histograms

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

Employee(ssn, 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>800</td>
<td>5000</td>
<td>12000</td>
<td>6500</td>
<td>500</td>
</tr>
</tbody>
</table>

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

Histories

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

<table>
<thead>
<tr>
<th>Salary</th>
<th>0-20k</th>
<th>20k-40k</th>
<th>40k-60k</th>
<th>60k-100k</th>
<th>&gt; 100k</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tuples</td>
<td>200</td>
<td>400</td>
<td>5000</td>
<td>22000</td>
<td>6500</td>
</tr>
</tbody>
</table>

Histories

- Assume:
  - \( V(\text{Employee}, \text{Salary}) = 200 \)
  - \( V(\text{Ranks}, \text{Salary}) = 250 \)
- Main property:
  Employee \( p_{\text{Salary}} \) Ranks = Employee, \( p_{\text{Salary}} \) Ranks, \( \ldots \) Employee, \( p_{\text{Salary}} \) Ranks

- A tuple \( t \) in Employee, joins with how many tuples in Ranks, \( ? \)
  - Answer: with \( T(\text{Employee}) \cdot T(\text{Employee}) / 250 = T_t / 250 \)

- Then \( T(\text{Employee}, p_{\text{Salary}} \text{ Ranks}) = \)
  \[ = \sum_{t \in T} T_t / 250 \]
  \[ = (200x1 + 800x20 + 5000x40 + 12000x80 + 6500x180 + 500x2) / 250 \]
  \[ = \ldots \]