Introduction to Database Systems  
CSE 444

Lecture 22:  
Query Optimization  
November 26-30, 2007

Outline

• An example
• Query optimization: algebraic laws 16.2
• Cost-based optimization 16.5, 16.6
• Cost estimation: 16.4

Example

Product(pname, maker), Company(cname, city)

Select Product.pname  
From Product, Company  
Where Product.maker = Company.cname  
and Company.city = “Seattle”

• How do we execute this query?

Logical Plan:

Physical plan 1:

Index-based selection

σ_city=“Seattle”

Product(pname, maker)

σ_city=“Seattle”

Company(cname, city)

σメーカー=cname

σメーカー=make

Company(cname, city)

σメーカー=make

Product(pname, maker)
Physical plans 2a and 2b:

Which one is better??

Physical plan 1:
Index-based selection
σ<sub>city</sub> = “Seattle”

Physical plan 2a and 2b:
Merge-join
σ<sub>maker</sub> = cname

Total cost:
(2a): 3B(Product) + B(Company)
(2b): T(Product) + B(Company)

Which one is better??

No extra cost (why?)

Total cost:
T(Company) / V(Company, city) × T(Product) / V(Product, maker)

Which Plan is Best?

Plan 1: T(Company) / V(Company, city) × T(Product) / V(Product, maker)
Plan 2a: B(Company) + 3B(Product)
Plan 2b: B(Company) + T(Product)

Which one is better??

It depends on the data!!

Example

T(Company) = 5,000  B(Company) = 500  M = 100
T(Product) = 100,000  B(Product) = 1,000

We may assume V(Product, maker) = T(Company) (why?)

- Case 1: V(Company, city) = T(Company)
  V(Company, city) = 2,000
- Case 2: V(Company, city) ≪ T(Company)
  V(Company, city) = 20

Plan 1: T(Company) / V(Company, city) × T(Product) / V(Product, maker)
Plan 2a: B(Company) + 3B(Product)
Plan 2b: B(Company) + T(Product)

Case 1:

Case 2:
Lessons

• Need to consider several physical plan
  – even for one, simple logical plan
• No magic “best” plan: depends on the data
• In order to make the right choice
  – need to have statistics over the data
  – the B’s, the T’s, the V’s

Query Optimization

• Have a SQL query Q
• Create a plan P
• Find equivalent plans P = P’ = P” = …
• Choose the “cheapest”.

Logical Query Plan

```
SELECT P.buyer
FROM Purchase P, Person Q
WHERE P.buyer=Q.name AND
  P.city='seattle' AND
  Q.phone > '5430000'
```

P= \( \Pi_{\text{buyer}} P \), \( \sigma_{\text{city}=\text{seattle} \land \text{phone}>'5430000'} P \)

In class: find a “better” plan P’

Logical Query Plan

```
SELECT city, sum(quantity)
FROM sales
GROUP BY city
HAVING sum(quantity) < 100
```

Q= \( \sigma_{\text{sum(quantity)} < 100} (\gamma_{\text{sales(product, city, quantity)}} \Pi_{\text{city}, \text{sum(quantity)}}) \)

P= \( \sigma_{\text{p} < 100} (\gamma_{\text{sales(product, city, quantity)}} \Pi_{\text{city}}) \)

In class: find a “better” plan P’

The three components of an optimizer

We need three things in an optimizer:

• Algebraic laws
• An optimization algorithm
• A cost estimator

Algebraic Laws (incomplete list)

• Commutative and Associative Laws
  \( R \cup S = S \cup R, \ R \cup (S \cup T) = (R \cup S) \cup T \)
  \( R \Join S = S \Join R, \ R \Join (S \Join T) = (R \Join S) \Join T \)
• Distributive Laws
  \( R \Join (S \cup T) = (R \Join S) \cup (R \Join T) \)
Algebraic Laws (incomplete list)

- Laws involving selection:
  \[ \sigma_{C \land C'}(R) = \sigma_C(\sigma_{C'}(R)) \]
  \[ \sigma_{C \lor C'}(R) = \sigma_C(R) \cup \sigma_{C'}(R) \]
- When \( C \) involves only attributes of \( R \)
  \[ \sigma_C(R \times S) = \sigma_C(R) \times S \]
  \[ \sigma_C(R - S) = \sigma_C(R) - S \]
  \[ \sigma_C(R \times S) = \sigma_C(R) \times S \]

Algebraic Laws

- Example: \( R(A, B, C, D), S(E, F, G) \)
  \[ \sigma_{A=3}(R \times D=E S) = \] ?
  \[ \sigma_{A=5 \land G=9}(R \times D=E S) = \] ?

Algebraic Laws

- Example: \( R(A, B, C, D), S(E, F, G) \)
  \[ \Pi_{A,B,G}(R \times D=E S) = \Pi_A(\Pi_B(R) \times D=E S) \]

Algebraic Laws

- Laws involving grouping and aggregation:
  \[ \delta(\gamma_{A, \text{agg}(D)}(R)) = \gamma_{A, \text{agg}(D)}(R) \]
  \[ \gamma_{A, \text{agg}(D)}(\delta(R)) = \gamma_{A, \text{agg}(D)}(R) \text{ if agg is "duplicate insensitive"} \]
- Which of the following are “duplicate insensitive”? sum, count, avg, min, max
  \[ \gamma_{A, \text{agg}(D)}(R(A,B) \times D=E S(C,D)) = \]
  \[ \gamma_{A, \text{agg}(D)}(R(A,B) \times D=E (\gamma_{C, \text{agg}(D)} S(C,D))) \]

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
Dynamic Programming

Originally proposed in System R (the first research prototype for a relational database system -- late 70s)
- Only handles single block queries:
  ```sql
  SELECT list
  FROM list
  WHERE cond1 AND cond2 AND ... AND cond_k
  ```
- Heuristics: selections down, projections up
- Dynamic programming: join reordering

Join Trees

- \[ R_1 \bowtie R_2 \bowtie \ldots \bowtie R_n \]
- Join tree:
  ```plaintext
  R3 \bowtie R1 \bowtie R2 \bowtie R4
  ```
- A plan = a join tree
- A partial plan = a subtree of a join tree

Types of Join Trees

- Left deep:
  ```plaintext
  R3 \bowtie R1 \bowtie R5 \bowtie R2 \bowtie R4
  ```
- Bushy:
  ```plaintext
  R3 \bowtie R1 \bowtie R2 \bowtie R4
  ```
- Right deep:
  ```plaintext
  R3 \bowtie R1 \bowtie R5 \bowtie R2 \bowtie R4
  ```

Dynamic Programming

- 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\}
  - ...
  - 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

For each subquery \(Q \subseteq \{R_1, \ldots, R_n\}\) compute the following:
- \(\text{Size}(Q)\)
- A best plan for \(Q\): \(\text{Plan}(Q)\)
- The cost of that plan: \(\text{Cost}(Q)\)

Dynamic Programming

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

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

Dynamic Programming

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

Dynamic Programming

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

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

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

Subquery Size Cost Plan

<table>
<thead>
<tr>
<th>Subquery</th>
<th>Size</th>
<th>Cost</th>
<th>Plan</th>
</tr>
</thead>
<tbody>
<tr>
<td>RS</td>
<td>100k</td>
<td>0</td>
<td>RS</td>
</tr>
<tr>
<td>RT</td>
<td>60k</td>
<td>0</td>
<td>RT</td>
</tr>
<tr>
<td>RU</td>
<td>20k</td>
<td>0</td>
<td>RU</td>
</tr>
<tr>
<td>ST</td>
<td>150k</td>
<td>0</td>
<td>ST</td>
</tr>
<tr>
<td>SU</td>
<td>90k</td>
<td>0</td>
<td>SU</td>
</tr>
<tr>
<td>TU</td>
<td>30k</td>
<td>0</td>
<td>TU</td>
</tr>
<tr>
<td>RST</td>
<td>3M</td>
<td>60k</td>
<td>(RT)S</td>
</tr>
<tr>
<td>RSU</td>
<td>1M</td>
<td>20k</td>
<td>(RU)S</td>
</tr>
<tr>
<td>RTU</td>
<td>0.6M</td>
<td>20k</td>
<td>(RU)T</td>
</tr>
<tr>
<td>STU</td>
<td>1.5M</td>
<td>30k</td>
<td>(TU)S</td>
</tr>
<tr>
<td>RSTU</td>
<td>30M</td>
<td>110k</td>
<td>(RT)(SU)</td>
</tr>
</tbody>
</table>

Reducing the Search Space

- Left-linear trees v.s. Bushy trees
- Trees without cartesian product

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

Plan: \( R(A,B) \times T(C,D) \times S(B,C) \) has a cartesian product – most query optimizers will not consider it

Dynamic Programming: Summary

- Handles only join queries:
  - Selections are pushed down (i.e. early)
  - Projections are pulled up (i.e. late)
- Takes exponential time in general, BUT:
  - Left linear joins may reduce time
  - Non-cartesian products may reduce time further
Rule-Based Optimizers

• **Extensible** collection of rules
  Rule = Algebraic law with a direction

• Algorithm for firing these rules
  Generate many alternative plans, in some order
  Prune by cost

• Volcano (later SQL Sever)
• Starburst (later DB2)

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?
  – \( \text{Cost} = \)
• 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?
  – \( \text{Cost} = \)
• How much main memory do we need?
  – \( M = \)
Pipeline in Bushy Trees

Example

• Logical plan is:

  \[ \text{R}(w, x) \]
  \[ \text{S}(x, y) \]
  \[ \text{U}(y, z) \]

• Main memory \( M = 101 \) buffers

Example

\[ \text{M} = 101 \]

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

Example

\[ \text{M} = 101 \]

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

Example

\[ \text{M} = 101 \]

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

Example

\[ \text{M} = 101 \]

Continuing:
- If \( 50 < k \leq 500 \) then send the 50 buckets in Step 3 to disk
  - Each bucket has size \( k/50 \leq 100 \)
  - Step 4: partition \( \text{U} \) into 50 buckets
    - Step 5: read each partition and join in memory
    - Total cost: \( 3B(\text{R}) + 3B(\text{S}) + 2k + 3B(\text{U}) = 75,000 + 2k \)
Example

\[ M = 101 \]

\[ U(y,z) \]

10,000 blocks

Continuing:
- If \( k > 5000 \) then materialize instead of pipeline
- 2 partitioned hash-joins
- Cost \( 3B(R) + 3B(S) + 4k + 3B(U) = 75000 + 4k \)

Example

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

Size Estimation

The problem: Given an expression \( E \), compute

\[ T(E) \text{ and } V(E, A) \]

- This is hard without computing \( E \)
- Will ‘estimate’ them instead

Size Estimation

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

Size Estimation

Estimating the size of a selection
- \( S = \sigma_{A < c}(R) \)
  - \( T(S) \) can be anything from 0 to \( T(R) - V(R,A) + 1 \)
  - Estimate: \( T(S) = T(R)\sqrt[3]{V(R,A)} \)
  - When \( V(R,A) \) is not available, estimate \( T(S) = T(R)/10 \)
- \( S = \sigma_{A > c}(R) \)
  - \( T(S) \) can be anything from 0 to \( T(R) \)
  - Estimate: \( T(S) = (c - Low(R,A))/(High(R,A) - Low(R,A))T(R) \)
  - When \( Low, High \) unavailable, estimate \( T(S) = T(R)/3 \)

Size Estimation

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

Assumptions:
• **Containment of values**: if \( V(R,A) \leq V(S,A) \), 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 \( B \),
  \( V(R \mid A) \cap S, B) = V(R, B) \) (or \( V(S, B) \))

Note: this indeed holds when \( A \) is a foreign key in \( R \) and a key in \( S \)

Size Estimation

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

Answer: \( T(R \mid A S) = 10000 \times \frac{20000}{200} = 1M \)

Size Estimation

Assume \( V(R,A) \leq V(S,A) \)
• 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 \( R \mid A S \)
• Hence \( T(R \mid A S) = T(R) T(S) / V(S,A) \)

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

Histories

• 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>Tuples</th>
</tr>
</thead>
<tbody>
<tr>
<td>0..20k</td>
<td>200</td>
</tr>
<tr>
<td>20k..40k</td>
<td>800</td>
</tr>
<tr>
<td>40k..60k</td>
<td>5000</td>
</tr>
<tr>
<td>60k..80k</td>
<td>12000</td>
</tr>
<tr>
<td>80k..100k</td>
<td>6500</td>
</tr>
<tr>
<td>&gt; 100k</td>
<td>500</td>
</tr>
</tbody>
</table>
• \( T(Employee) = 25000 \), but now we know the distribution
Histograms

Ranks(rankName, salary)
- Estimate the size of Employee | ![Salary Ranks](chart)

<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></td>
<td>200</td>
<td>400</td>
<td>5000</td>
<td>12000</td>
<td>8500</td>
<td>500</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Ranks</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></td>
<td>8</td>
<td>20</td>
<td>40</td>
<td>80</td>
<td>100</td>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Eqwidth</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0.20</td>
<td>20</td>
<td>40</td>
<td>60</td>
<td>80</td>
<td>100</td>
<td>1200</td>
</tr>
<tr>
<td></td>
<td>104</td>
<td>9739</td>
<td>152</td>
<td>3</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Eqdepth</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0.44</td>
<td>44.48</td>
<td>48.50</td>
<td>50.56</td>
<td>55.100</td>
<td>2000</td>
<td>2000</td>
</tr>
</tbody>
</table>