Introduction to Database Systems
CSE 444
Lecture #16
March 5, 2001

Query Optimization
Required Reading: 7.2, 7.4, 7.5, 7.6

Query Optimization: Phases
- Parsing phase
  - Produces a parse tree
- Query-Rewrite phase
  - Produces a logical tree
- Physical Query plan generation
  - Produces executable (physical) plan

Query Optimization
- Algebraic laws provide alternative execution plans
- Estimate costs of alternative modes of execution
- Efficiently search the space of alternatives
  - Simplify search by applying heuristics (without costing)
    - apply laws that seem to result in cheaper plans

Converting from SQL to Logical Plans
Select a1, ..., an
From R1, ..., Rk
Where C
\[ \Pi_{a_1, \ldots, a_n}(\sigma_c (R_1 \bowtie R_2 \bowtie \ldots \bowtie R_k)) \]

Converting from SQL to Logical Plans
Select a1, ..., an
From R1, ..., Rk
Where C
Group by b1, ..., bl
\[ \Pi_{a_1, \ldots, a_n}(\gamma_{b_1, \ldots, b_m, \text{aggs}} (\sigma_c (R_1 \bowtie R_2 \bowtie \ldots \bowtie R_k))) \]
Algebraic Laws

- **Commutative and Associative Laws**
  - \( R U S = S U R, \quad R U (S U T) = (R U S) U T \)
  - \( R \cap S = S \cap R, \quad R \cap (S \cap T) = (R \cap S) \cap T \)
- **Distributive Laws**
  - \( R \cap (S U T) = (R \cap S) U (R \cap T) \)

Algebraic Laws: Selection

- **Laws involving selection:**
  - \( \sigma_{C \text{ AND } C'}(R) = \sigma_C(\sigma_{C'}(R)) = \sigma_C(R) \cap \sigma_{C'}(R) \)
  - \( \sigma_{C \text{ OR } C'}(R) = \sigma_C(R) U \sigma_{C'}(R) \)
  - \( \sigma_C(R \Join S) = \sigma_C(R) \Join S \)
  - When \( C \) involves only attributes of \( R \)
  - \( \sigma_C(R - S) = \sigma_C(R) - S \)
  - \( \sigma_C(R U S) = \sigma_C(R) U \sigma_C(S) \)
  - \( \sigma_C(R \cap S) = \sigma_C(R) \cap S \)

Algebraic Laws: Projection

- **Laws involving projections**
  - \( \Pi_{\alpha}(R \Join S) = \Pi_{\alpha}(\Pi_{\beta}(R) \Join \Pi_{\gamma}(S)) \)
  - Where \( \alpha, \beta, \gamma \) are appropriate subsets of attributes of \( M \)
  - \( \Pi_{\alpha}(\Pi_{\beta}(R)) = \Pi_{\beta}(R) \)
- **Example**
  - \( R(A, B, C, D), \quad S(E, F, G) \)
  - \( \sigma_{F=3}(R \Join S) = \) ?
  - \( \sigma_{A=5 \text{ AND } G=9}(R \Join S) = \) ?

Other Algebraic Laws

- **Duplicate Elimination**
  - \( \delta(R \Join S) = \delta(R) \Join \delta(S) \)
- **Grouping**
  - \( \delta(\gamma(R)) = \gamma(\delta(R)) \)
  - Many transformations depend on aggregate
    - \( \text{MAX}, \text{SUM} \) etc.
Cost Estimation

- For a given logical plan, there may be many possible physical plans
- We want to choose physical plan with lowest execution cost
- Goal: For a given physical plan, estimate cost without executing the query

Estimating Size of Selection

- How to estimate size of $S = \sigma_{A=20}(R)$?
- Approach 1: Guess!
  - Surprisingly popular method e.g. $T(R)/10$
- Approach 2: Use statistics
  - $T(S) = T(R)/V(R,A)$
  - Where $V(R,A)$ = number of distinct values of A in R
- How about $S = \sigma_{A \leq 20}(R)$?
  - Guess: $T(R)/3$
  - Statistics: Use histogram if available (more later)

Estimating Size of Projection

- Projection does not change number of tuples
- Size estimate depends on length of columns
- Example: $R(a,b,c)$: $a, b$ are integers, $c$ string of 100 bytes. Tuple header = 12 bytes, Block size = 1024
  - $\pi_{a,b,c}(R) =$ ? $\pi_{a,b}(R) =$ ?
- What if c is variable length string?

Estimating Size of Join

- $R(a,b), S(b,c)$, estimate $T(R \bowtie S)$
- Problem: Don’t know how values of R.b and S.b are related
  - May be disjoint sets of values =>$T(R \bowtie S) = 0$
  - S.b may be key of S and R.b may be foreign key =>$T(R \bowtie S) = T(R)$
- Estimate for $T(R \bowtie S)$
  - $T(R)T(S)/\max(V(R,b), V(S,b))$

Estimating Size of Join

- Example: $T(R) = 1000$, $T(S) = 2000$, $V(R,b) = 20$, $V(S,b) = 50$
  - $T(R \bowtie S) =$ ?
**Estimating Size of Join**

- What happens if query has multiple join attributes?
  - Example: \( R(a,b,c), S(b,c,d) \)
  - Estimate = ?
- What happens if query has joins of many relations?
  - Example: \( R(a,b), S(b,c), U(b,e) \)
  - Estimate = ?

**Estimating Size of Other Operators**

- Union \((R,S)\)
  - Bag Union: \( T(R) + T(S) \)
  - Set Union: \( \max(T(R),T(S)) + \min(T(R),T(S))/2 \)
- Intersection \((R,S)\)
  - Min\(T(R),T(S))/2\)
- Difference \((R,S)\)
  - \( T(R) - T(S)/2 \)
- Duplicate Elimination

**Cost Based Plan Selection**

- Estimates for size parameters
- Use statistics, e.g. histograms
- Enumerating physical plans

**Histograms**

- Popular in commercial DBMSs
- Can give much more accurate cost estimates
- Many types of histograms
  - Equal-width
  - Equal-depth
  - Frequent values
  - ...

**Equal-width Histogram**

- Each bucket in histogram has same width
- Example: Values = \( \{2,5,23,25,29,31\} \)
  - Range | Count
  - 1-10  | 2
  - 11-20 | 0
  - 21-30 | 3
  - 31-40 | 1
- \( T(\sigma_{A < 20}(R)) = 2 \)

**Equi-depth Histogram**

- Each bucket in histogram has same number of values
- Example: \( \{2,5,33,35,39,41\} \)
  - Bucket Boundary
  - 5
  - 35
  - 41

---

**Page 19**

**Page 20**

**Page 21**

**Page 22**

**Page 23**

**Page 24**
Frequent Values

- Keep exact counts of frequent values
- Total count of all other (non-frequent) values
- Example: Values = \{1, 3, 4, 4, 4, 4, 4, 9\}
- Histogram: 4: 5, Others: 3

Using Histogram for Size Estimation

- Example: R(a,b) S(b,c).
- Histograms:
  - R.b: 1:200, 0:150, 5:100, Others: 550
  - S.b: 0:100, 1:80, 2:70, Others: 250
- Size of join = ?

Creating and Maintaining Statistics in a DBMS

- For large tables, creating/refreshing statistics can be expensive
- Alternatives:
  - Refresh statistics only after many changes to data
  - Incremental updating
  - Sampling – need to be careful...

Enumerating Physical Plans

- Exhaustive – Consider all possible:
  - Join Orders
  - Algorithms for each operator
- Heuristic Search
  - E.g. Greedy approach
  - Pick next relation such that join size is smallest

Enumerating Physical Plans

- Branch-and-Bound Enumeration
  - Find a good starting plan (having cost C)
  - In subsequent search, eliminate any subquery with cost > C
- Hill Climbing
  - Start with heuristically selected plan
  - Explore plans in the "neighborhood"
    - E.g. replace Nested-Loops join with Hash-Join

Enumerating Physical Plans

- Dynamic Programming
  - Bottom-up strategy
  - For each subexpression, only keep plan with the least cost
  - Consider possible implementations of each node assuming
  - Extension: also consider interesting orders
    - E.g., when subexpression is sorted on a sort attribute at the node
  - More later
Determining Join Order

- Select-project-join
- Push selections down, pull projections up
- Hence: we need to choose the join order
- This is the main focus of an optimizer

Determining Join Order: Join Trees

- R1 ⋈ R2 ⋈ ... ⋈ Rn
- Join tree:

Bushy Join Trees

- Left deep:

Join Ordering Problem

- Given: a query R1 ⋈ R2 ⋈ ... ⋈ Rn
- Assume we have a function cost() that gives us the cost of every join tree
- Find the best linear join tree for the query

Dynamic Programming

- For each subquery Q \{R1, ..., Rn\}
  - compute the following:
    - Size(Q)
    - A best plan for Q: Plan(Q)
    - The cost of that plan: 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 \( Q \{R_1, \ldots, R_n\} \) of cardinality \( i \) do:
  - Compute \( \text{Size}(Q) \)
  - For every pair of subqueries \( Q', Q'' \) s.t. \( Q = Q' \cup Q'' \)
    - \( \text{compute cost(Plan}(Q') \bowtie \text{Plan}(Q'')) \)
  - \( \text{Cost}(Q) = \text{the smallest such cost} \)
  - \( \text{Plan}(Q) = \text{the corresponding plan} \)

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

---

**Dynamic Programming**

To illustrate, we will make the following simplifications:

- \( \text{Cost}(P_1 \bowtie P_2) = \text{Cost}(P_1) + \text{Cost}(P_2) + \text{size(intermediate result(s))} \)

- **Intermediate results:**
  - If \( P_1 \) = 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

- We used naïve size/cost estimations

- In practice:
  - More realistic size/cost estimations
  - 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"