Lecture 24:

Wednesday, November 27, 2002

Outline

• Query optimization: algebraic laws 16.2
• Cost-based optimization 16.5, 16.6

The three components of an optimizer

We need three things in an optimizer:

• Algebraic laws
• An optimization algorithm
• A cost estimator

Algebraic Laws

• Commutative and Associative Laws
  – \(R \cup S = S \cup R, \ R \cup (S \cup T) = (R \cup S) \cup T\)
  – \(R \cap S = S \cap R, \ R \cap (S \cap T) = (R \cap S) \cap T\)
  – \(R \times S = S \times R, \ R \times (S \times T) = (R \times S) \times T\)

• Distributive Laws
  – \(R \times (S \cup T) = (R \times S) \cup (R \times T)\)

Algebraic Laws

• Laws involving selection:
  – \(\sigma_{C AND \ C}(R) = \sigma_{C}(\sigma_{C}(R)) = \sigma_{C}(R) \cap \sigma_{C}(R)\)
  – \(\sigma_{C OR \ C}(R) = \sigma_{C}(R) \cup \sigma_{C}(R)\)
  – \(\sigma_{C}(R \times S) = \sigma_{C}(R) \times S\)
    • When \(C\) involves only attributes of \(R\)
  – \(\sigma_{C}(R \minus{} S) = \sigma_{C}(R) \minus{} S\)
  – \(\sigma_{C}(R \cup S) = \sigma_{C}(R) \cup \sigma_{C}(S)\)
  – \(\sigma_{C}(R \cap S) = \sigma_{C}(R) \cap S\)

• Example: \(R(A, B, C, D), S(E, F, G)\)
  – \(\sigma_{F \uparrow 3}(R \times_{D \equiv E} S) = \) ?
  – \(\sigma_{A \uparrow 5 \ AND \ G \uparrow 9}(R \times_{D \equiv E} S) = \) ?
Algebraic Laws

- Laws involving projections
  - $\Pi_M(R \times S) = \Pi_M(\Pi_P(R) \times \Pi_Q(S))$
    - Where $N, P, Q$ are appropriate subsets of attributes of $M$
  - $\Pi_M(\Pi_P(R)) = \Pi_{M\cup P}(R)$
- Example $R(A,B,C,D), S(E, F, G)$
  - $\Pi_{A,B,D}(R \times S) = \Pi_{A}(\Pi_{E}(R) \times \Pi_{F}(S))$

Heuristic Based Optimizations

- Query rewriting based on algebraic laws
- Result in better queries most of the time
- Heuristics number 1:
  - Push selections down
- Heuristics number 2:
  - Sometimes push selections up, then down

Predicate Pushdown

```
Select y.name, Max(x.price)
From product x, company y
Where x.maker = y.name
Having Max(x.price) > 100
```

- For each company, find the maximal price of its products.
- Advantage: the size of the join will be smaller.
- Requires transformation rules specific to the grouping/aggregation operators.
- Won’t work if we replace Max by Min.

Algebraic Laws

- Laws involving grouping and aggregation:
  - $\delta(\gamma_{A \text{ agg}}(R)) = \gamma_{A \text{ agg}}(R)$
  - $\gamma_{A \text{ agg}}(\delta(R)) = \gamma_{A \text{ agg}}(R)$ if agg is “duplicate insensitive”
    - Which of the following are “duplicate insensitive”?
      - sum, count, avg, min, max
  - $\gamma_{A \text{ agg}}(\delta(\tau_{A \text{ agg}}(R(A,B)_{B=C} S(C,D))))$
    - Why is this true?
    - Why would we do it?

Predicate Pushdown

```
Select V2.name, V2.price
From V1, V2
Where V1.category = V2.category and V1.p = V2.price
```

Pushing predicates up

Bargain view V1: categories with some price > 20, and the cheapest price

Create View V1 AS
```
Select x.category, Min(x.price) AS p
From product x
Where x.price < 20
GroupBy x.category
```

Create View V2 AS
```
Select y.name, x.category, x.price
From product x, company y
Where x.maker = y.name
```
Query Rewrite: Pushing predicates up
Bargain view V1: categories with some price<20, and the cheapest price

Select V2.name, V2.price
From V1, V2
Where V1.category = V2.category and V1.p = V2.price AND V1.p < 20

Create View V1 AS
Select x.category,
    Min(x.price) AS p
From product x
Where x.price < 20
GroupBy x.category

Create View V2 AS
Select y.name, x.category, x.price
From product x, company y
Where x.maker = y.name

Query Rewrite: Pushing predicates up
Bargain view V1: categories with some price<20, and the cheapest price

Select V2.name, V2.price
From V1, V2
Where V1.category = V2.category and V1.p = V2.price AND V1.p < 20

Create View V1 AS
Select x.category,
    Min(x.price) AS p
From product x
Where x.price < 20
GroupBy x.category

Create View V2 AS
Select y.name, x.category, x.price
From product x, company y
Where x.maker = y.name

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

- R1 \bowtie R2 \bowtie \ldots \bowtie R_n
- Join tree:

![Join Tree Diagram]

- A join tree represents a plan. An optimizer needs to inspect many (all ?) join trees

Types of Join Trees

- Left deep:

![Left Deep Join Tree Diagram]

Types of Join Trees

- Bushy:

![Bushy Join Tree Diagram]

Types of Join Trees

- Right deep:

![Right Deep Join Tree Diagram]

Problem

- Given: a query R1 \bowtie R2 \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\}
- A subset of \{R_1, \ldots, R_n\} is also called a subquery
Dynamic Programming

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

---

Dynamic Programming

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

---

Dynamic Programming

- **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 $\text{cost}($Plan($Q'$) $\Join$ Plan($Q''$))$
  - Cost($Q$) = the smallest such cost
  - Plan($Q$) = the corresponding plan

---

Dynamic Programming

To illustrate, we will make the following simplifications:

- Cost($P_1 \Join P_2$) = Cost($P_1$) + Cost($P_2$) + size(intermediate result(s))
- Intermediate results:
  - If $P_1$ is 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:
  - Cost(R5$\Join$R7) = 0 (no intermediate results)
  - Cost((R2$\Join$R1) $\Join$ R7)
    = Cost(R2$\Join$R1) + Cost(R7) + size(R2$\Join$R1)
    = size(R2) $\Join$ size(R1)
Dynamic Programming

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