Lecture 26

Monday, December 3, 2001

Outline

- Query optimization
  - Algebraic laws
  - Heuristic-based optimizations
  - Choosing an order for Joins (7.6)

Algebraic Laws

- Commutative and Associative Laws
  - $R \cup S = S \cup R$, $R \cup (S \cup T) = (R \cup S) \cup T$
  - $R \cap S = S \cap R$, $R \cap (S \cap T) = (R \cap S) \cap T$
  - $R \bowtie S = S \bowtie R$, $R \bowtie (S \bowtie T) = (R \bowtie S) \bowtie T$
- Distributive Laws
  - $R \bowtie (S \cup T) = (R \bowtie S) \cup (R \bowtie T)$

Algebraic Laws

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

Algebraic Laws

- Example: R(A, B, C, D), S(E, F, G)
  - $\sigma_{F=3}(R \underset{D=E}{\bowtie} S) =$?
  - $\sigma_{A=5 \text{ AND } G=9}(R \underset{D=E}{\bowtie} S) =$?

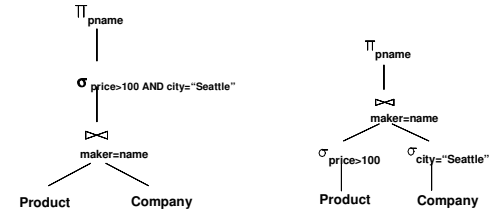
Algebraic Laws

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

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



The earlier we process selections, less tuples we need to manipulate higher up in the tree (but may cause us to loose an important ordering of the tuples, if we use indexes).

Predicate Pushdown

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

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

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
```

```
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
  AND V1.p < 20
```

Cost-based Optimizations

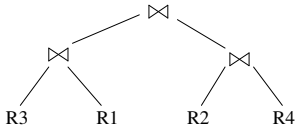
Unit of Optimization

- Select-project-join
  - Push selections down, pull projections up
- Hence: remains to choose the join order
- This is the main focus of the cost-based optimizer

13

Join Trees

- R1 ⋈ R2 ⋈ …. ⋈ Rn
- Join tree:

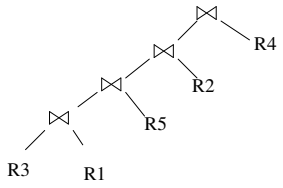


- A join tree represents a plan. An optimizer needs to inspect many (all ?) join trees

14

Types of Join Trees

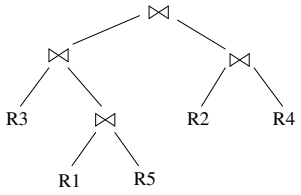
- Left deep:



15

Types of Join Trees

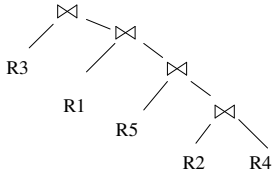
- Bushy:



16

Types of Join Trees

- Right deep:



17

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 join tree for the query

18

Dynamic Programming

- Idea: for each subset of {R1, …, Rn}, compute the best plan for that subset
- In increasing order of set cardinality:
  - Step 1: for {R1}, {R2}, …, {Rn}
  - Step 2: for {R1,R2}, {R1,R3}, …, {Rn-1, Rn}
  - …
  - Step n: for {R1, …, Rn}
- A subset of {R1, …, Rn} is also called a *subquery*

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 {Ri} do:
  - Size({Ri}) = B(Ri)
  - Plan({Ri}) = Ri
  - Cost({Ri}) = (cost of scanning Ri)

Dynamic Programming

- **Step i**: For each Q ⊆ {R1, …, Rn} of cardinality i do:
  - Compute Size(Q) (later…)
  - For every pair of subqueries Q', Q''
    s.t. Q = Q' U Q''
    compute cost(Plan(Q') ⋈ Plan(Q''))
  - Cost(Q) = the smallest such cost
  - Plan(Q) = the corresponding plan

Dynamic Programming

- Return Plan({R1, …, Rn})

Dynamic Programming

To illustrate, we will make the following simplifications:

- Cost(P1 ⋈ P2) = Cost(P1) + Cost(P2) + size(intermediate result(s))
- Intermediate results:
  - If P1 = a join, then the size of the intermediate result is size(P1), otherwise the size is 0
  - Similarly for P2
- 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)
  = size(R2 ⋈ R1)

25

Dynamic Programming

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

26

Subquery	Size	Cost	Plan
RS			
RT			
RU			
ST			
SU			
TU			
RST			
RSU			
RTU			
STU			
RSTU			

27