Lecture 24:

Wednesday, November 27, 2002

1

Outline

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

2

The three components of an optimizer

We need three things in an optimizer:

- Algebraic laws
- An optimization algorithm
- A cost estimator

3

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
  - $R \bowtie (S \cup T) = (R \bowtie S) \cup (R \bowtie T)$

4

Algebraic Laws

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

5

Algebraic Laws

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

6

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)
  - $\Pi_{A,B,G}(R \bowtie S) = \Pi_{?}(\Pi_?(R) \bowtie \Pi_?(S))$

7

Algebraic Laws

Laws involving grouping and aggregation:

- $\delta(\gamma_{A,\ agg(B)}(R)) = \gamma_{A,\ agg(B)}(R)$
- $\gamma_{A,\ agg(B)}(\delta(R)) = \gamma_{A,\ agg(B)}(R)$ if agg is "duplicate insensitive"
  - Which of the following are "duplicate insensitive" ? sum, count, avg, min, max
- $\gamma_{A,\ agg(D)}(R(A,B) \bowtie_{B=C} S(C,D)) =$
  $\gamma_{A,\ agg(D)}(R(A,B) \bowtie_{B=C} (\gamma_{B,\ agg(D)}S(C,D)))$
  - Why is this true ?
  - Why would we do it ?

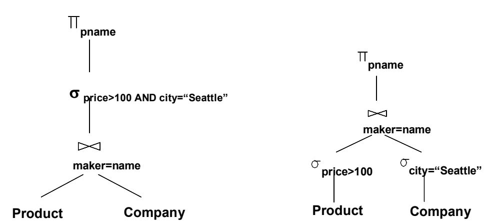
8

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

9

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).

10

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.**

11

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
```

12

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
```

13

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

- 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

15

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

16

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

17

Dynamic Programming

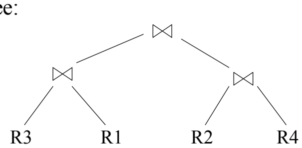
Unit of Optimization

- Select-project-join
  - Push selections down, pull projections up

18

Join Trees

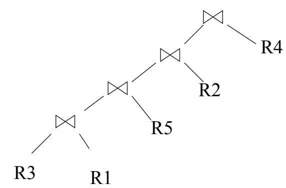
- R1 $\bowtie$ R2 $\bowtie$ …. $\bowtie$ Rn
- Join tree:



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

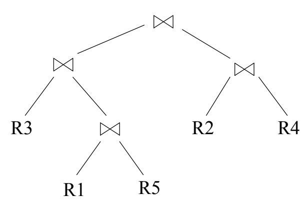
Types of Join Trees

- Left deep:



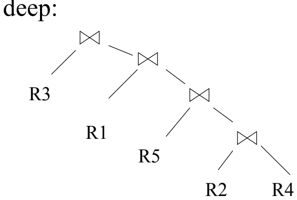
Types of Join Trees

- Bushy:



Types of Join Trees

- Right deep:



Problem

- Given: a query R1 $\bowtie$ R2 $\bowtie$ … $\bowtie$ Rn
- 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 {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)

25

Dynamic Programming

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

26

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

27

Dynamic Programming

- Return Plan({R1, …, Rn})

28

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

29

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)

30

Dynamic Programming

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

31

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

32

Subquery	Size	Cost	Plan
RS	100k	0	RS
RT	60k	0	RT
RU	20k	0	RU
ST	150k	0	ST
SU	50k	0	SU
TU	30k	0	TU
RST	3M	60k	(RT)S
RSU	1M	20k	(RU)S
RTU	0.6M	20k	(RU)T
STU	1.5M	30k	(TU)S
RSTU	30M	60k+50k=110k	(RT)(SU)

33

Dynamic Programming

- Summary: computes optimal plans for subqueries:
  - Step 1: {R1}, {R2}, …, {Rn}
  - Step 2: {R1, R2}, {R1, R3}, …, {Rn-1, Rn}
  - …
  - Step n: {R1, …, Rn}
- We used naïve size/cost estimations
- In practice:
  - more realistic size/cost estimations (next time)
  - 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"

34