Lecture 25

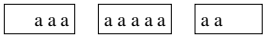
Friday, November 30, 2001

Outline

- Query execution
  - Two pass algorithms based on indexes (6.7)
- Query optimization
  - From SQL to logical plans (7.3)
  - Algebraic laws (7.2)
  - Choosing an order for Joins (7.6)

Indexed Based Algorithms

- Recall that in a clustered index all tuples with the same value of the key are clustered on as few blocks as possible



- Note: book uses another term: "clustering index". Difference is minor…

Index Based Selection

- Selection on equality: $\sigma_{a=v}(R)$
- Clustered index on a: cost B(R)/V(R,a)
- Unclustered index on a: cost T(R)/V(R,a)

Index Based Selection

- Example:

B(R) = 2000
T(R) = 100,000
V(R, a) = 20

cost of $\sigma_{a=v}(R)$ = ?

- Table scan:
  - If R is clustered: B(R) = 2,000 I/Os
  - If R is unclustered: T(R) = 100,000 I/Os
- Index based selection:
  - If index is clustered: B(R)/V(R,a) = 100
  - If index is unclustered: T(R)/V(R,a) = 5,000
- Notice: when V(R,a) is small, then unclustered index is useless

Index Based Join

- $R \bowtie S$
- Assume S has an index on the join attribute
- Iterate over R, for each tuple fetch corresponding tuple(s) from S
- Assume R is clustered. Cost:
  - If index is clustered: B(R) + T(R)B(S)/V(S,a)
  - If index is unclustered: B(R) + T(R)T(S)/V(S,a)

Index Based Join

- Assume both R and S have a sorted index (B+ tree) on the join attribute
- Then perform a merge join
  - called zig-zag join
- Cost: B(R) + B(S)

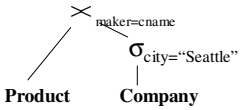
Example

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

Clustered index: **Product**.pname, **Company**.cname
Unclustered index: **Product**.maker, **Company**.city

```
Select Product.pname
From  Product, Company
Where Product.maker=Company.cname
   and  Company.city = "Seattle"
```


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



- Case 1: V(**Company**, city) ≈ T(**Company**)
  V(**Product**, maker) ≈ T(**Product**)

  V(**Company**,city) = 2,000
  V(**Product**,maker) = 20,000

- Case 2: V(**Company**, city) << T(**Company**)
  V(**Product**, maker) ≈ T(**Product**)

  V(**Company**,city) = 20
  V(**Product**,maker) = 20,000

- Case 3: V(**Company**, city) << T(**Company**)
  V(**Product**, maker) << T(**Product**)

  V(**Company**,city) = 20
  V(**Product**,maker) = 100

- Case 1:
  - Index-based selection: T(**Company**) / V(**Company**, city)
  - Index-based join: x T(**Product**) / V(**Product**, maker)

  T(**Company**) / V(**Company**, city) x T(**Product**) / V(**Product**, maker)
  = 2.5 x 5
  ≈ 13

- Case 3:
  - Table scan and selection on Company: B(**Company**) sorted !!!
  - Merge-join with Product: + B(**Product**)

  B(**Company**) + B(**Product**) = 1,500

- Case 2:
  - Plan 1 costs 250 x 5 = 1250, Plan 3 costs 1,500
  - Plan 1 is better

Optimization

- Start: convert SQL to Logical Plan
- Algebraic laws:
  - foundation for every optimization
- Two approaches to optimizations:
  - Heuristics: apply laws that *seem* to result in cheaper plans
  - Cost based: estimate size and cost of intermediate results, search systematically for best plan

Converting from SQL to Logical Plans

```
Select a1, …, an
From R1, …, Rk
Where C
```

$\Pi_{a1,\ldots,an}(\sigma_C(R1 \times R2 \times \ldots \times Rk))$

```
Select a1, …, an
From R1, …, Rk
Where C
Group by b1, …, bl
```

$\Pi_{a1,\ldots,an}(\gamma_{b1,\ldots,bm,aggs}(\sigma_C(R1 \times R2 \times \ldots \times Rk)))$

Converting Nested Queries

```
Select distinct product.name
From product
Where product.maker in (Select company.name
                        From company
                        where company.city="Seattle")
```

```
Select distinct product.name
From product, company
Where product.maker = company.name AND
      company.city="Seattle"
```

13

Converting Nested Queries

```
Select distinct x.name, x.maker
From product x
Where x.color= "blue"
  AND x.price >= ALL (Select y.price
                      From  product y
                      Where x.maker = y.maker
                        AND y.color="blue")
```

How do we convert this one to logical plan ?

14

Converting Nested Queries

Let's compute the complement first:

```
Select distinct x.name, x.maker
From product x
Where x.color= "blue"
  AND x.price < SOME (Select y.price
                      From  product y
                      Where x.maker = y.maker
                        AND y.color="blue")
```

15

Converting Nested Queries

This one becomes a SFW query:

```
Select distinct x.name, x.maker
From product x, product y
Where x.color= "blue" AND x.maker = y.maker
  AND y.color="blue"  AND x.price < y.price
```

This returns exactly the products we DON'T want, so…

16

Converting Nested Queries

```
(Select x.name, x.maker
 From product x
 Where x.color = "blue")

EXCEPT

(Select x.name, x.maker
 From product x, product y
 Where x.color= "blue" AND x.maker = y.maker
   AND y.color="blue"  AND x.price < y.price)
```

17

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

18

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
  - $\sigma_C(R - S) = \sigma_C(R) - S$
  - $\sigma_C(R \cup S) = \sigma_C(R) \cup \sigma_C(S)$
  - $\sigma_C(R \cap S) = \sigma_C(R) \cap S$

19

Algebraic Laws

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

20

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

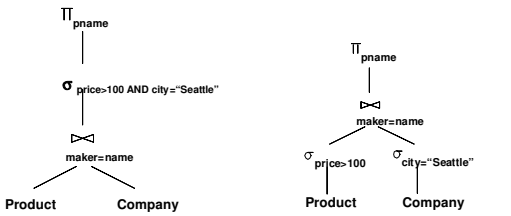
21

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

22

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).

23

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.**

24

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
```

25

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
```

26

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

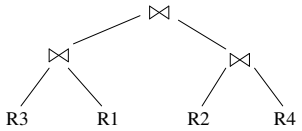
Unit of Optimization

- 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

28

Join Trees

- R1 ⋈ R2 ⋈ …. ⋈ Rn
- Join tree:

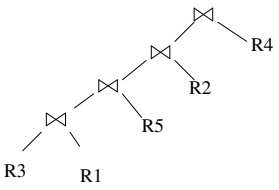


- A join tree represents a plan. An optimizer needs to inspect many (all ?) join trees

29

Types of Join Trees

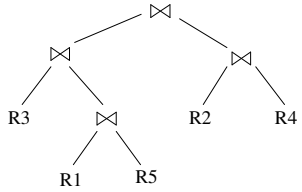
- Left deep:



30

Types of Join Trees

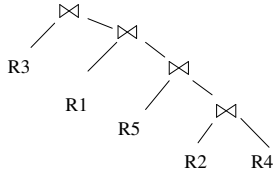
- Bushy:



31

Types of Join Trees

- Right deep:



32

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

33

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*

34

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)

35

Dynamic Programming

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

36

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

37

Dynamic Programming

- Return Plan({R1, …, Rn})

38

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

39

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)

40