Lecture 25: Query Optimization

Wednesday, November 22, 2000

Outline

- Finish index-based algorithms
- From SQL to logical plans (7.3)
- Algebraic laws (7.2)

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, compute the cost of $\sigma_{a=v}(R)$
- Cost of table scan:
  - If R is clustered: B(R) = 2000 I/Os
  - If R is unclustered: T(R) = 100,000 I/Os
- Cost of index based selection:
  - If index is clustered: B(R)/V(R,a) = 100
  - If index is unclustered: T(R)/V(R,a) = 5000
- 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)

Optimization

- Start: convert SQL to Logical Plan
- Algebraic laws: are the 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

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

Converting from SQL to Logical Plans

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

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

Converting Nested Queries

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

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

Converting Nested Queries

```
Select 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 ?

Converting Nested Queries

- Let's compute the complement first:

```
Select 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"
```

Converting Nested Queries

- This one becomes a SFW query:

```
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
```

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

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)
```

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

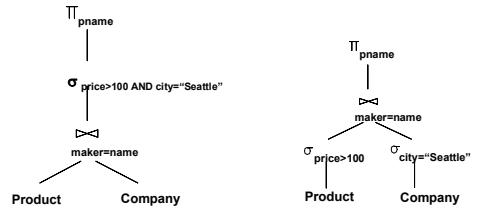
Algebraic Laws

- 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 !

Query Rewriting: 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).

Query Rewrites: Predicate Pushdown (through grouping)

```
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.**

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
```

```
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
```