Lecture 21: Query Optimization (1)

November 17, 2010

Administrivia (Preview for Friday)

- For project 4, students are expected (but not required) to work in pairs.
- Ideally you should pair up by end of day Monday.
- That way, Michael can give each group their shared Amazon AWS grant code by Tuesday.
- Once you run out of money on your AWS grant, your *personal* credit cards will be charged!
- Because students are asked to do interactive rather than batch jobs on AWS, they should remember to explicitly kill every job.

Where We Are

- We are learning how a DBMS executes a query
- What we learned so far
  – How data is stored and indexed
  – Logical query plans and physical operators
- This week:
  – How to select logical & physical query plans

Review

Supplier(<u>sid</u>, sname, scity, sstate)
Supply(<u>sid, pno</u>, quantity)

```
SELECT sname
FROM Supplier x, Supply y
WHERE x.sid = y.sid
   and  y.pno = 2
   and x.scity = 'Seattle'
   and x.sstate = 'WA'
```

Give a relational algebra expression for this query

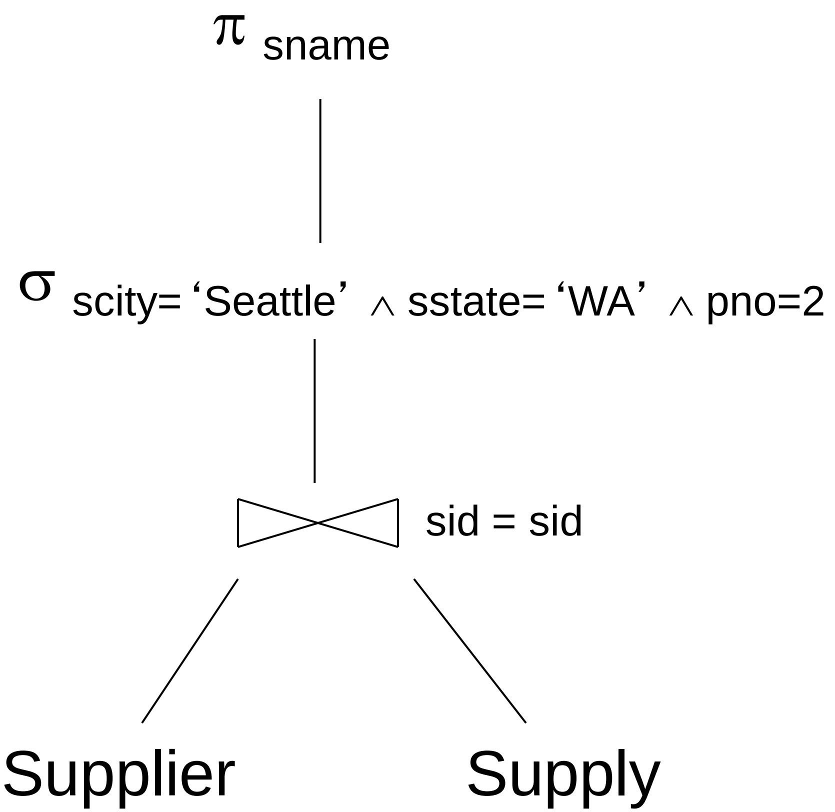
Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)

Relational Algebra

$$\pi_{sname}(\sigma_{scity=\text{'Seattle'} \wedge sstate=\text{'WA'} \wedge pno=2}(\text{Supplier} \bowtie_{sid = sid} \text{Supply}))$$

Relational Algebra

Supplier(<u>sid</u>, sname, scity, sstate)
Supply(<u>sid, pno</u>, quantity)



Key Idea: *Algebraic Optimization*

$$N = ((z*2)+((z*3)+y))/x$$

Given x = 1, y = 0, and z = 4, solve for N

What order did you perform the operations?

Key Idea: *Algebraic Optimization*

$$N = ((z*2)+((z*3)+0))/1$$

Given x = 1, y = 0, and z = 4, solve for N again, but now assume:

* costs 10 units

\+ costs 2 units

/ costs 50 units

Which execution plan offers the lowest cost?

Key Idea: Algebraic Optimization

N = ((z*2)+((z*3)+0))/1

Algebraic Laws:

1. (+) identity: x+0 = x
2. (/) identity: x/1 = x
3. (*) distributes: (n*x+n*y) = n*(x+y)
4. (*) commutes: x*y = y*x

Apply rules 1, 3, 4, 2:

N = (2+3)*z

two operations instead of five, no division operator

Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)

```
SELECT sname
FROM Supplier x, Supply y
WHERE x.sid = y.sid
   and  y.pno = 2
   and x.scity = 'Seattle'
   and x.sstate = 'WA'
```

$$\pi_{\text{sname}}(\sigma_{\text{scity='Seattle'} \wedge \text{sstate='WA'} \wedge \text{pno=2}}(\text{Supplier} \bowtie_{\text{sid = sid}} \text{Supply}))$$

Give a different relational algebra expression for this query

Query Optimization Goal

- For a query
  - There exist many logical and physical query plans
  - Query optimizer needs to pick a good one

Example

Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)

```
SELECT sname
FROM Supplier x, Supply y
WHERE x.sid = y.sid
   and  y.pno = 2
   and x.scity = 'Seattle'
   and x.sstate = 'WA'
```

- Some statistics
  - T(Supplier) = 1000 records
  - T(Supply) = 10,000 records
  - B(Supplier) = 100 pages
  - B(Supply) = 100 pages
  - V(Supplier,scity) = 20, V(Supplier,state) = 10
  - V(Supply,pno) = 2,500
  - Both relations are clustered
- M = 10

T(Supplier) = 1000
T(Supply) = 10,000

B(Supplier) = 100
B(Supply) = 100

V(Supplier,scity) = 20
V(Supplier,state) = 10
V(Supply,pno) = 2,500

M = 10

Physical Query Plan 1

(On the fly) π_{sname}

(On the fly) $\sigma_{\text{scity='Seattle'} \wedge \text{sstate='WA'} \wedge \text{pno=2}}$

(Block-nested loop) $\bowtie_{\text{sid = sid}}$

Supplier (File scan)

Supply (File scan)

Selection and project on-the-fly
-> No additional cost.

Total cost of plan is thus cost of join:
= B(Supplier)+B(Supplier)*B(Supply)/M
= 100 + 10 * 100
= 1,100 I/Os

T(Supplier) = 1000
T(Supply) = 10,000

B(Supplier) = 100
B(Supply) = 100

V(Supplier,scity) = 20
V(Supplier,state) = 10
V(Supply,pno) = 2,500

M = 10

Physical Query Plan 2

(On the fly)

π_{sname} (d)

(Sort-merge join)

$\bowtie_{\text{sid = sid}}$ (c)

(Scan write to T1)

(a) $\sigma_{\text{scity= 'Seattle'} \wedge \text{sstate= 'WA'}}$

(Scan write to T2)

(b) $\sigma_{\text{pno=2}}$

Supplier (File scan)

Supply (File scan)

Total cost
= 100 + 100 * 1/20 * 1/10 (a)
+ 100 + 100 * 1/2500 (b)
+ 2 (c)
+ 0 (d)

Total cost ≈ **204 I/Os**

T(Supplier) = 1000
T(Supply) = 10,000

B(Supplier) = 100
B(Supply) = 100

V(Supplier,scity) = 20
V(Supplier,state) = 10
V(Supply,pno) = 2,500

M = 10

Physical Query Plan 3

(On the fly) (d) π_{sname}

(On the fly)

(c) $\sigma_{\text{scity='Seattle'} \wedge \text{sstate='WA'}}$

(b) $\bowtie_{\text{sid = sid}}$ (Index nested loop)

4 tuples

(Use index)

(a) $\sigma_{\text{pno=2}}$

Supply

Supplier

(Index lookup on pno)
Assume: clustered

(Index lookup on sid)
Doesn't matter if clustered or not

Total cost
= 1 (a)
+ 4 (b)
+ 0 (c)
+ 0 (d)

Total cost ≈ **5 I/Os**

Simplifications

- In the previous examples, we assumed that all index pages were in memory
- When this is not the case, we need to add the cost of fetching index pages from disk

Query Optimization Goal

- For a query
  - There exist many logical and physical query plans
  - Query optimizer needs to pick a good one

How do we choose a good one?

Query Optimization Algorithm

- Enumerate alternative plans
- Compute estimated cost of each plan
  - Compute number of I/Os
  - Compute CPU cost
- Choose plan with lowest cost
  - This is called cost-based optimization

Lessons

- Need to consider several physical plan
  - even for one, simple logical plan
- No magic “best” plan: depends on the data
- In order to make the right choice
  - need to have ***statistics*** over the data
  - the B’s, the T’s, the V’s

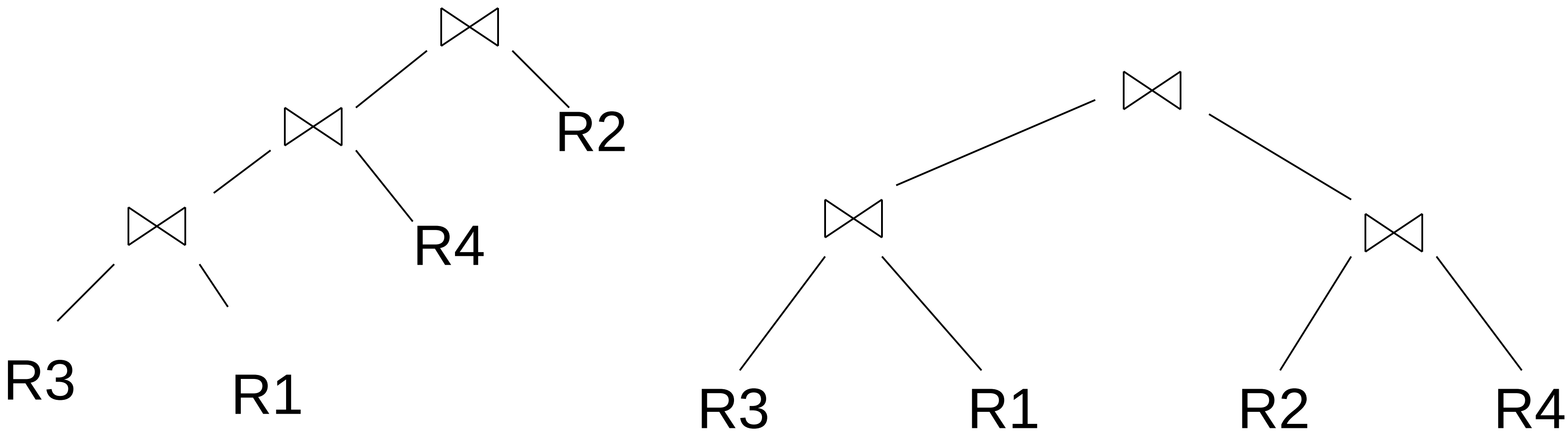
Outline

- Search space (Today)
- Algorithm for enumerating query plans
- Estimating the cost of a query plan

Relational Algebra Equivalences

- Selections
  - Commutative: $\sigma_{c1}(\sigma_{c2}(R))$ same as $\sigma_{c2}(\sigma_{c1}(R))$
  - Cascading: $\sigma_{c1 \wedge c2}(R)$ same as $\sigma_{c2}(\sigma_{c1}(R))$
- Projections
- Joins
  - Commutative : $R \bowtie S$ same as $S \bowtie R$
  - Associative: $R \bowtie (S \bowtie T)$ same as $(R \bowtie S) \bowtie T$

Left-Deep Plans and Bushy Plans



Left-deep plan

Bushy plan

Commutativity, Associativity, Distributivity

$$R \cup S = S \cup R, \quad R \cup (S \cup T) = (R \cup S) \cup T$$
$$R \bowtie S = S \bowtie R, \quad R \bowtie (S \bowtie T) = (R \bowtie S) \bowtie T$$
$$R \bowtie S = S \bowtie R, \quad R \bowtie (S \bowtie T) = (R \bowtie S) \bowtie T$$

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

Example

Which plan is more efficient ?

$R \bowtie (S \bowtie T)$ or $(R \bowtie S) \bowtie T$?

- Assumptions:
  - Every join selectivity is 10%
    - That is: $T(R \bowtie S) = 0.1 * T(R) * T(S)$ etc.
  - B(R)=100, B(S) = 50, B(T)=500
  - All joins are main memory joins
  - All intermediate results are materialized

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

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

Example: Simple Algebraic Laws

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

Laws Involving Projections

$$\Pi_M(R \bowtie S) = \Pi_M(\Pi_P(R) \bowtie \Pi_Q(S))$$

$$\Pi_M(\Pi_N(R)) = \Pi_M(R)$$

/* note that $M \subseteq N$ */

- Example R(A,B,C,D), S(E, F, G)

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

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

$$\gamma_{A,\ agg(D)}(R(A,B) \bowtie_{B=C} S(C,D)) =$$
$$\gamma_{A,\ agg(D)}(R(A,B) \bowtie_{B=C} (\gamma_{C,\ agg(D)}S(C,D)))$$

Laws Involving Constraints

Foreign key

Product(<u>pid</u>, pname, price, cid)
Company(<u>cid</u>, cname, city, state)

$$\Pi_{pid,\ price}(\text{Product} \bowtie_{cid=cid} \text{Company}) = \Pi_{pid,\ price}(\text{Product})$$

Need a second constraint for this law to hold. Which one ?

Example

Foreign key

Product(pid, pname, price, cid)
Company(cid, cname, city, state)

```
CREATE VIEW CheapProductCompany
    SELECT *
    FROM Product x, Company y
    WHERE x.cid = y.cid and x.price < 100
```

```
SELECT pname, price
FROM CheapProductCompany
```

⇨

```
SELECT pname, price
FROM Product
WHERE price < 100
```

Laws with Semijoins

Recall the definition of a semijoin:

- $R \ltimes S = \Pi_{A1,\ldots,An} (R \bowtie S)$
- Where the schemas are:
  - Input: R(A1,….An), S(B1,…,Bm)
  - Output: T(A1,…,An)

Laws with Semijoins

Semijoins: a bit of theory (see *Database Theory,* AHV)

- Given a query: $Q = R_1 \bowtie R_2 \bowtie \ldots \bowtie R_n$
- A *semijoin reducer* for Q is

$$
\begin{aligned}
R_{i1} &= R_{i1} \ltimes R_{j1} \\
R_{i2} &= R_{i2} \ltimes R_{j2} \\
&\ldots\ldots \\
R_{ip} &= R_{ip} \ltimes R_{jp}
\end{aligned}
$$

  such that the query is equivalent to:

$$Q = R_{k1} \bowtie R_{k2} \bowtie \ldots \bowtie R_{kn}$$

- A *full reducer* is such that no dangling tuples remain

Laws with Semijoins

- Example:

$$Q = R(A,B) \bowtie S(B,C)$$

- A reducer is:

$$R_1(A,B) = R(A,B) \ltimes S(B,C)$$

- The rewritten query is:

$$Q = R_1(A,B) \bowtie S(B,C)$$

Why would we do this ?

Why Would We Do This ?

- Large attributes:

$$Q = R(A,B,\ D, E, F,\ldots) \bowtie S(B,C,\ M, K, L,\ \ldots)$$

- Expensive side computations

$$Q = \gamma_{A,B,count(*)}R(A,B,D) \bowtie \sigma_{C=value}(S(B,C))$$

$$R_1(A,B,D) = R(A,B,D) \ltimes \sigma_{C=value}(S(B,C))$$
$$Q = \gamma_{A,B,count(*)}R_1(A,B,D) \bowtie \sigma_{C=value}(S(B,C))$$

Laws with Semijoins

- Example:

$$Q = R(A,B) \bowtie S(B,C)$$

- A reducer is:

$$R_1(A,B) = R(A,B) \ltimes S(B,C)$$

- The rewritten query is:

$$Q = R_1(A,B) \bowtie S(B,C)$$

Are there dangling tuples ?

Laws with Semijoins

- Example:

$$Q = R(A,B) \bowtie S(B,C)$$

- A full reducer is:

$$R_1(A,B) = R(A,B) \ltimes S(B,C)$$
$$S_1(B,C) = S(B,C) \ltimes R_1(A,B)$$

- The rewritten query is:

$$Q :- R_1(A,B) \bowtie S_1(B,C)$$

No more dangling tuples

Laws with Semijoins

- More complex example:

$$Q = R(A,B) \bowtie S(B,C) \bowtie T(C,D,E)$$

- A full reducer is:

$$S'(B,C) := S(B,C) \ltimes R(A,B)$$
$$T'(C,D,E) := T(C,D,E) \ltimes S(B,C)$$
$$S''(B,C) := S'(B,C) \ltimes T'(C,D,E)$$
$$R'(A,B) := R(A,B) \ltimes S''(B,C)$$

$$Q = R'(A,B) \bowtie S''(B,C) \bowtie T'(C,D,E)$$

Laws with Semijoins

- Example:

$$\boxed{Q = R(A,B) \bowtie S(B,C) \bowtie T(A,C)}$$

- Doesn’t have a full reducer (we can reduce forever)

Theorem a query has a full reducer iff it is “acyclic”
[*Database Theory*, by Abiteboul, Hull, Vianu]

Example with Semijoins

Emp(<u>eid</u>, ename, sal, did)
Dept(<u>did</u>, dname, budget)
DeptAvgSal(did, avgsal) /* view */

[Chaudhuri' 98]

View:

```
CREATE VIEW DepAvgSal As (
        SELECT E.did, Avg(E.Sal) AS avgsal
        FROM Emp E
        GROUP BY E.did)
```

Query:

```
SELECT E.eid, E.sal
FROM Emp E, Dept D, DepAvgSal V
WHERE E.did = D.did AND E.did = V.did
        AND E.age < 30 AND D.budget > 100k
        AND E.sal > V.avgsal
```

Goal: compute only the necessary part of the view

Example with Semijoins

Emp(eid, ename, sal, did)
Dept(did, dname, budget)
DeptAvgSal(did, avgsal) /* view */

[Chaudhuri' 98]

New view uses a reducer:

```
CREATE VIEW LimitedAvgSal As (
        SELECT E.did, Avg(E.Sal) AS avgsal
        FROM Emp E, Dept D
        WHERE E.did = D.did AND D.buget > 100k
        GROUP BY E.did)
```

New query:

```
SELECT E.eid, E.sal
FROM Emp E, Dept D, LimitedAvgSal V
WHERE E.did = D.did AND E.did = V.did
        AND E.age < 30 AND D.budget > 100k
        AND E.sal > V.avgsal
```

Example with Semijoins

Emp(<u>eid</u>, ename, sal, did)

Dept(<u>did</u>, dname, budget)

DeptAvgSal(did, avgsal) /* view */

[Chaudhuri' 98]

Full reducer:

```
CREATE VIEW PartialResult AS
        (SELECT E.eid, E.sal, E.did
        FROM Emp E, Dept D
        WHERE E.did=D.did AND E.age < 30
        AND D.budget > 100k)

CREATE VIEW Filter AS
        (SELECT DISTINCT P.did FROM PartialResult P)

CREATE VIEW LimitedAvgSal AS
        (SELECT E.did, Avg(E.Sal) AS avgsal
        FROM Emp E, Filter F
        WHERE E.did = F.did GROUP BY E.did)
```

Example with Semijoins

New query:

```
SELECT P.eid, P.sal
FROM PartialResult P, LimitedDepAvgSal V
WHERE P.did = V.did AND P.sal > V.avgsal
```

Search Space Challenges

- Search space is huge!
  - Many possible equivalent trees
  - Many implementations for each operator
  - Many access paths for each relation
    - File scan or index + matching selection condition

- Cannot consider ALL plans
  - Heuristics: only partial plans with "low" cost