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
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
Relational Algebra

\[ \pi_{\text{sname}} \left( \sigma_{\text{scity} = 'Seattle' \land \text{sstate} = 'WA' \land \text{pno} = 2} \left( \text{Supplier} \bowtie_{\text{sid} = \text{sid}} \text{Supply} \right) \right) \]
Relational Algebra

\[ \pi_{\text{sname}} \sigma_{\text{scity}=\text{'Seattle'} \land \text{sstate}=\text{'WA'} \land \text{pno}=2} \]

\[ \text{sid} = \text{sid} \]

\[ \text{Supplier} \quad \text{Supply} \]
Key Idea: *Algebraic Optimization*

\[ N = \frac{(z \times 2) + ((z \times 3) + y)}{x} \]

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

What order did you perform the operations?
Key Idea: *Algebraic Optimization*

\[
N = \frac{(z\times 2)+(z\times 3)+0}{1}
\]

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?

Bill Howe -- 444 Fall 2010
Key Idea: Algebraic Optimization

\[ N = \frac{((z \times 2) + ((z \times 3) + 0))}{1} \]

Algebraic Laws:

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

Apply rules 1, 3, 4, 2:
\[ N = (2 + 3) \times z \]

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'

\[ \pi\text{ sname}\left(\sigma\text{ scity=‘Seattle’} \land \text{sstate=‘WA’} \land \text{pno=2}\left(\text{Supplier \Join_{sid = sid} Supply}\right)\right) \]

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

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

$M = 10$

```
SELECT sname 
FROM Supplier x, Supply y 
WHERE x.sid = y.sid 
  and y.pno = 2 
  and x.scity = 'Seattle'
  and x.sstate = 'WA'
```
Physical Query Plan 1

(On the fly)  \[ \pi_{sname} \]  Selection and project on-the-fly  
-> No additional cost.

(On the fly)  \[ \sigma_{\text{scity=’Seattle’} \land \text{sstate=’WA’} \land \text{pno=2}} \]

(Block-nested loop)  \[ \text{sid = sid} \]

Total cost of plan is thus cost of join:
= \[ B(\text{Supplier}) + B(\text{Supplier}) \times B(\text{Supply}) / M \]
= 100 + 10 * 100
= 1,100 I/Os
Physical Query Plan 2

(On the fly)

\( \pi \text{ sname} \)  
\( \text{sid} = \text{sid} \)

(Sort-merge join)

(Scan write to T1)

(Scan write to T2)

Total cost
\[ = 100 + 100 \times \frac{1}{20} \times \frac{1}{10} \]  
\[ + 100 + 100 \times \frac{1}{2500} \]  
\[ + 2 \]  
\[ + 0 \]  
Total cost \( \approx 204 \) I/Os

Bill Howe -- 444 Fall 2010
Physical Query Plan 3

Total cost
= 1 (a) + 4 (b) + 0 (c) + 0 (d) 
≈ 5 I/Os

(a) \( \sigma_{pno=2} \)  
[Index lookup on pno]  
Supply
Assume: clustered

(b) \( \sigma_{sid=sid} \)  
[Index lookup on sid]  
Supplier

(c) \( \sigma_{scity='Seattle'} \land sstate='WA' \)  
[Index nested loop]  

(d) \( \pi_{sname} \)

(On the fly)  

(On the fly)

Use index

B(Supplier) = 100  
V(Supplier, scity) = 20  
M = 10

B(Supply) = 100  
V(Supplier, state) = 10  
V(Supply, pno) = 2,500

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

4 tuples

Total cost
\approx 5 \text{ 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: $\sigma_{c_1}(\sigma_{c_2}(R))$ same as $\sigma_{c_2}(\sigma_{c_1}(R))$
  – Cascading: $\sigma_{c_1 \land c_2}(R)$ same as $\sigma_{c_2}(\sigma_{c_1}(R))$

• 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

- R3
- R1
- R4

Bushy plan

- R3
- R1
- R2
- R4
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: \( T(R \bowtie S) = 0.1 \times T(R) \times T(S) \) etc.
  – \( B(R)=100, B(S) = 50, B(T)=500 \)
  – All joins are main memory joins
  – All intermediate results are materialized
Laws involving selection:

\[\begin{align*}
\sigma_{C \text{ AND } C'}(R) &= \sigma_{C}(\sigma_{C'}(R)) = \sigma_{C}(R) \cap \sigma_{C'}(R) \\
\sigma_{C \text{ OR } C'}(R) &= \sigma_{C}(R) \cup \sigma_{C'}(R) \\
\sigma_{C}(R \bowtie S) &= \sigma_{C}(R) \bowtie S
\end{align*}\]

When C involves only attributes of R

\[\begin{align*}
\sigma_{C}(R - S) &= \sigma_{C}(R) - S \\
\sigma_{C}(R \cup S) &= \sigma_{C}(R) \cup \sigma_{C}(S) \\
\sigma_{C}(R \bowtie S) &= \sigma_{C}(R) \bowtie S
\end{align*}\]
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

\[
\Pi_M(R \Join S) = \Pi_M(\Pi_P(R) \Join \Pi_Q(S))
\]

\[
\Pi_M(\Pi_N(R)) = \Pi_M(R)
\]

/* note that M \subseteq N */

• Example \(R(A,B,C,D), S(E, F, G)\)

\[
\Pi_{A,B,G}(R \Join_{D=E} S) = \Pi ? (\Pi ?(R) \Join_{D=E} \Pi ?(S))
\]
Laws involving grouping and aggregation

\[ \delta(\gamma_A, \text{agg}(B)(R)) = \gamma_A, \text{agg}(B)(R) \]

\[ \gamma_A, \text{agg}(B)(\delta(R)) = \gamma_A, \text{agg}(B)(R) \]

*if agg is “duplicate insensitive”*

Which of the following are “duplicate insensitive”? sum, count, avg, min, max

\[ \gamma_A, \text{agg}(D)(R(A,B) \bowtie_{B=C} S(C,D)) = \gamma_A, \text{agg}(D)(R(A,B) \bowtie_{B=C} (\gamma_C, \text{agg}(D)S(C,D))) \]
Laws Involving Constraints

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

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

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

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:

\[ R \bowtie S = \Pi_{A_1, \ldots, A_n} (R \bowtie S) \]

• Where the schemas are:
  – Input: \( R(A_1, \ldots, A_n), \ S(B_1, \ldots, B_m) \)
  – Output: \( T(A_1, \ldots, A_n) \)
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

\[
\begin{align*}
R_{i1} &= R_{i1} \bowtie R_{j1} \\
R_{i2} &= R_{i2} \bowtie R_{j2} \\
\vdots & \\
R_{ip} &= R_{ip} \bowtie R_{jp}
\end{align*}
\]

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:

\[ Q = R(A,B) \Join S(B,C) \]

• A reducer is:

\[ R_1(A,B) = R(A,B) \bowtie S(B,C) \]

• The rewritten query is:

\[ Q = R_1(A,B) \Join S(B,C) \]

Why would we do this?
Why Would We Do This?

- Large attributes:
  \[ Q = R(A,B,D,E,F,...) \bowtie S(B,C,M,K,L,...) \]

- Expensive side computations
  \[ Q = \gamma_{A,B,\text{count}(\ast)} R(A,B,D) \bowtie \sigma_{C=\text{value}}(S(B,C)) \]

\[ R_1(A,B,D) = R(A,B,D) \bowtie \sigma_{C=\text{value}}(S(B,C)) \]
\[ Q = \gamma_{A,B,\text{count}(\ast)} R_1(A,B,D) \bowtie \sigma_{C=\text{value}}(S(B,C)) \]
Laws with Semijoins

• Example:

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

• A reducer is:

\[ R_1(A,B) = R(A,B) \bowtie S(B,C) \]

• 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:

\[
\begin{align*}
R_1(A,B) &= R(A,B) \bowtie S(B,C) \\
S_1(B,C) &= S(B,C) \bowtie R_1(A,B)
\end{align*}
\]

• 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:

\[
\begin{align*}
S'(B,C) & := S(B,C) \bowtie R(A,B) \\
T'(C,D,E) & := T(C,D,E) \bowtie S(B,C) \\
S''(B,C) & := S'(B,C) \bowtie T'(C,D,E) \\
R'(A,B) & := R(A,B) \bowtie S''(B,C)
\end{align*}
\]

\[ Q = R'(A,B) \bowtie S''(B,C) \bowtie T'(C,D,E) \]
Laws with Semijoins

• Example:

\[ Q = R(A,B) \bowtie S(B,C) \bowtie T(A,C) \]

• 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

**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 */

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.budget > 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

[Chaudhuri’ 98]
Example with Semijoins

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

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