Lecture 19:
Query Optimization (1)

May 17, 2010
Announcements

• Homework 3 due on Wednesday in class
  – How is it going?

• Project 4 posted
  – Due on June 2\textsuperscript{nd}
  – Start early!
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
Query Optimization Goal

• For a query
  – There exists many logical and physical query plans
  – Query optimizer needs to pick 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
Example

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

• 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

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

Supplier (File scan)  

Supply (File scan)

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

\[
\begin{align*}
\pi_{\text{sname}} & \quad \text{(4)} \\
\sigma_{\text{scity}=\text{Seattle} \land \text{sstate}=\text{WA}} & \quad \text{(1)} \\
\sigma_{\text{pno}=2} & \quad \text{(2)} \\
\text{Scan to T1} & \quad \text{(Scan write to T1)} \\
\text{Scan to T2} & \quad \text{(Scan write to T2)} \\
\text{Supplier} & \quad \text{(File scan)} \\
\text{Supply} & \quad \text{(File scan)} \\
\end{align*}
\]

Total cost
\[
= 100 + 100 \times \frac{1}{20} \times \frac{1}{10} \quad (1) \\
+ 100 + 100 \times \frac{1}{2500} \quad (2) \\
+ 2 \quad (3) \\
+ 0 \quad (4)
\]

Total cost \( \approx 204 \) I/Os

\[
\begin{align*}
T(\text{Supplier}) &= 1000 \\
B(\text{Supplier}) &= 100 \\
V(\text{Supplier}, \text{scity}) &= 20 \\
M &= 10 \\
T(\text{Supply}) &= 10,000 \\
B(\text{Supply}) &= 100 \\
V(\text{Supplier}, \text{state}) &= 10 \\
V(\text{Supply}, \text{pno}) &= 2,500
\end{align*}
\]
Physical Query Plan 3

(On the fly) (4) \( \pi_{sname} \)

(On the fly)

(3) \( \sigma_{scity='Seattle' \land sstate='WA'} \)

(On the fly)

(2) \( \sigma_{pno=2} \)

(Use index)

(1) \( B(Suppliers) = 100 \)

(Use index)

(4) \( T(Suppliers) = 1000 \)

Total cost

= 1 (1) + 4 (2) + 0 (3) + 0 (3)

Total cost \( \approx \) 5 I/Os

(1) \( B(Supply) = 100 \)

(4) \( T(Supply) = 10,000 \)

V(Supplier, scity) = 20

V(Supplier, state) = 10

V(Supply, pno) = 2,500

M = 10

V(Supplier, scity) = 20

V(Supplier, state) = 10

V(Supply, pno) = 2,500

M = 10
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
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

\[
R \cup S = S \cup R, \quad R \cup (S \cup T) = (R \cup S) \cup T
\]

\[
R \Join S = S \Join R, \quad R \Join (S \Join T) = (R \Join S) \Join T
\]

\[
R \Join S = S \Join R, \quad R \Join (S \Join T) = (R \Join S) \Join T
\]

\[
R \Join (S \cup T) = (R \Join S) \cup (R \Join T)
\]
Example

Which plan is more efficient?

\( R \Join (S \Join T) \) or \( (R \Join S) \Join T \) ?

• Assumptions:
  – Every join selectivity is 10%
    • That is: \( T(R \Join S) = 0.1 \times T(R) \times T(S) \) etc.
  – \( B(R) = 100, B(S) = 50, B(T) = 500 \)
  – All joins are main memory joins
  – All intermediate results are materialized
Laws involving selection:

\[ \sigma_{C \text{ AND } C'}(R) = \sigma_C(\sigma_{C'}(R)) = \sigma_C(R) \cap \sigma_{C'}(R) \]
\[ \sigma_{C \text{ OR } C'}(R) = \sigma_C(R) \cup \sigma_{C'}(R) \]
\[ \sigma_C(R \bowtie S) = \sigma_C(R) \bowtie S \]

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

\[ \Pi_M(R \bowtie S) = \Pi_M(\Pi_P(R) \bowtie \Pi_Q(S)) \]
\[ \Pi_M(\Pi_N(R)) = \Pi_M(R) \quad /* \text{note that } M \subseteq N */ \]

- 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

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

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\((pid, p\text{name}, price, cid)\)
Company\((cid, c\text{name}, city, state)\)

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

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

\[
\begin{align*}
R_{i_1} &= R_{i_1} \bowtie R_{j_1} \\
R_{i_2} &= R_{i_2} \bowtie R_{j_2} \\
\ldots & \\
R_{i_p} &= R_{i_p} \bowtie R_{j_p}
\end{align*}
\]

such that the query is equivalent to:

\[ Q = R_{k_1} \bowtie R_{k_2} \bowtie \ldots \bowtie R_{k_n} \]

- 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:

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

• 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
  \[ Q = \gamma_{A,B,\text{count}}(R(A,B,D) \bowtie \sigma_{C=\text{value}}(S(B,C))) \]
  \[ R_1(A,B,D) = R(A,B,D) \bowtie \sigma_{C=\text{value}}(S(B,C)) \]
  \[ Q = \gamma_{A,B,\text{count}}(R_1(A,B,D) \bowtie \sigma_{C=\text{value}}(S(B,C))) \]
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:

\[ R_1(A,B) = R(A,B) \bowtie S(B,C) \]
\[ S_1(B,C) = S(B,C) \bowtie R_1(A,B) \]

• The rewritten query is:

\[ Q :\leftarrow R_1(A,B) \bowtie S_1(B,C) \]

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

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

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
Example with Semijoins

Emp(eid, ename, sal, did)
Dept(did, 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