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

Relational Algebra

```sql
\pi_{\text{name}} (\sigma_{\text{scity} = 'Seattle' \land \text{sstate} = 'WA' \land \text{pno} = 2} (\text{Supplier} \bowtie \text{Supply}))
```

Give a relational algebra expression for this query

Relational Algebra

```sql
\pi_{\text{name}} (\sigma_{\text{scity} = 'Seattle' \land \text{sstate} = 'WA' \land \text{pno} = 2} (\text{Supplier} \bowtie \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?

Bill Howe -- 444 Fall 2010

---

Key Idea: Algebraic Optimization

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

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?

Bill Howe -- 444 Fall 2010

---

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: \( (n \times x + n \times y) = n \times (x + y) \)
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

Bill Howe -- 444 Fall 2010

---

Query Optimization Goal

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

Bill Howe -- 444 Fall 2010

---

Example

```
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 different relational algebra expression for this query

Bill Howe -- 444 Fall 2010
Physical Query Plan 1

(On the fly) \( \pi \text{sname} \) Selection and project on-the-fly
\( \sigma \text{scity} = 'Seattle' \land \text{sstate} = 'WA' \land \text{pno} = 2 \) -> No additional cost.

(On the fly) \( \pi \text{sname} \) (File scan)

(Block-nested loop)

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

Physical Query Plan 2

(On the fly) \( \pi \text{sname} \) (d) Total cost
\( \sigma \text{scity} = 'Seattle' \land \text{sstate} = 'WA' \land \text{pno} = 2 \)

(Sort-merge join)

\[\text{Total cost} = 100 + 100 \times 1/20 \times 1/10 \text{ (a)} + 100 + 100 \times 1/2500 \text{ (b)} + 2 \text{ (c)} + 0 \text{ (d)} \]

Total cost \( \approx 204 \text{ I/Os} \)

Physical Query Plan 3

(On the fly) \( \pi \text{sname} \) (d)

(On the fly) \( \sigma \text{scity} = 'Seattle' \land \text{sstate} = 'WA' \)

(Use index)

\( \sigma \text{pno} = 2 \) (Index nested loop) Total cost \( \approx 5 \text{ I/Os} \)

(Scan write to T1)

\( \sigma \text{scity} = 'Seattle' \land \text{sstate} = 'WA' \land \text{pno} = 2 \) (Scan write to T2)

Assume: clustered

Physical Query Plan 4

Physical Query Plan 5

Physical Query Plan 6

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

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?
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: \( \sigma_{c_1\cdot c_2}(R) \) 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

Commutativity, Associativity, Distributivity

\[
\begin{align*}
R \cup S &= S \cup R, \\
R \cup (S \cup T) &= (R \cup S) \cup T \\
R \bowtie S &= S \bowtie R, \\
R \bowtie (S \bowtie T) &= (R \bowtie S) \bowtie T \\
R \bowtie (S \cup T) &= (R \bowtie S) \cup (R \bowtie T)
\end{align*}
\]

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:

\[ \sigma_{C \land C'}(R) = \sigma_C(\sigma_{C'}(R)) = \sigma_C(R) \land \sigma_{C'}(R) \]
\[ \sigma_{C \lor C'}(R) = \sigma_C(R) \lor \sigma_{C'}(R) \]
\[ \sigma_C(R \bowtie S) = \sigma_C(R) \bowtie S \]

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: \( R(A, B, C, D) \), \( S(E, F, G) \)
  \[ \sigma_{F=3} (R \bowtie D=E S) = \]
  \[ \sigma_{A=5 \land G=9} (R \bowtie D=E S) = \]

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 \( R(A,B,C,D) \), \( S(E, F, G) \)
  \[ \Pi_{A,B,G}(R \bowtie_{D=E} S) = \Pi_{A} (\Pi_{C}(R) \bowtie_{D=E} \Pi_{F}(S)) \]

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 \( \text{agg} \) 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)

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

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

Example

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

SELECT pname, price FROM CheapProductCompany WHERE price < 100

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

SELECT pname, price FROM CheapProductCompany 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: $T(A_1, \ldots, A_n)$

Why Would We Do This?

- 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?

Why would we do this?

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:
  $R_{i1} = R_{i1} \bowtie R_{j1}$
  $R_{i2} = R_{i2} \bowtie R_{j2}$
  
  such that the query is equivalent to:
  $Q = R_{i1} \bowtie R_{i2} \bowtie \ldots \bowtie R_n$

- A full reducer is such that no dangling tuples remain

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

Example with Semijoins

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

View:

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

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.sal > V.avgsal
```

Example with Semijoins

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

Full reducer:

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

Example with Semijoins

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

New query:

```
SELECT P did, 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