CSE 544
Principles of Database Management Systems

Alvin Cheung
Fall 2015
Lecture 7 - Query optimization
Announcements

• HW1 due tonight at 11:45pm

• HW2 will be due in two weeks
  – You get to implement your own DBMS!

• We will meet with each project teams next week
  – Will send out doodle
References

• **Access path selection in a relational database management system.**
  Selinger. et. al. SIGMOD 1979

• **Database management systems.**
  Ramakrishnan and Gehrke.
  Third Ed. **Chapter 15.**
Query Optimization Motivation

- Parse & Rewrite Query
- Select Logical Plan
- Select Physical Plan
- Query Execution

Query optimization

- SQL query
- Declarative query
  - Recall physical and logical data independence

Disk

Logical plan

Physical plan
What We Already Know…

Supplier(sno, sname, scity, sstate)
Part(pno, pname, psize, pcolor)
Supply(sno, pno, price)

For each SQL query….
SELECT S.sname
FROM Supplier S, Supply U
WHERE S.scity='Seattle' AND S.sstate='WA'
AND S.sno = U.sno
AND U.pno = 2

There exist many logical query plan…
Example Query: Logical Plan 1

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

Supplier \rightarrow \sigma_{\text{sscity}='Seattle' \land \text{sstate}='WA' \land \text{pno}=2} \rightarrow \pi_{\text{sname}} \]

Supply
Example Query: Logical Plan 2

\[
\begin{align*}
\pi_{\text{sname}} \quad &
\quad \sigma_{\text{sno} = \text{sno}} \\
\sigma_{\text{sscity}='Seattle' \land \text{sstate}='WA'} \\
\sigma_{\text{pno}=2} \\
\text{Supplier} \\
\text{Supply}
\end{align*}
\]
What We Also Know

• For each logical plan…

• There exist many physical plans
Example Query: Physical Plan 1

(On the fly) \[ \pi_{\text{sname}} \]

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

(Nested loop) \[ \text{sno} = \text{sno} \]

Supplier (File scan)

Supply (File scan)
Example Query: Physical Plan 2

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

(On the fly) \( \sigma \text{scity='Seattle' \& sstate='WA' \& pno=2} \)

(Index nested loop) \( \text{sno = sno} \)

Supplier (File scan)

Supply (Index scan)
Query Optimization Algorithm

- For a query
  - There exists many physical query plans
  - Query optimizer needs to pick a good one

- Basic query optimization algorithm
  - Enumerate alternative plans
  - Compute estimated cost of each plan
    - Compute number of I/Os
    - Optionally take into account other resources
  - Choose plan with lowest cost
  - This is called cost-based optimization
Query Optimization

**Three major components:**

1. Cardinality and cost estimation
2. Search space
3. Plan enumeration algorithms
Estimating Cost of a Query Plan

• We already know how to
  – Compute the cost of different operations in terms of number IOs

• We still need to
  – Compute cost of retrieving tuples from disk with different access paths (for more sophisticated predicates than equality)
  – Compute cost of a complete plan
Access Path

• **Access path**: a way to retrieve tuples from a table
  – A file scan
  – An index *plus* a matching selection condition

• Index matches selection condition if it can be used to retrieve just tuples that satisfy the condition
  – Example: `Supplier(sid, sname, scity, sstate)`
  – B+-tree index on `(scity, sstate)`
    • matches `scity='Seattle'`
    • does not match `sid=3`, does not match `sstate='WA'`
Access Path Selection

- Supplier(sid,sname,scity,sstate)

- Selection condition: $\text{sid} > 300 \land \text{scity} = \text{‘Seattle’}$

- Indexes: B+-tree on sid and B+-tree on scity

- Which access path should we use?

- We should pick the most selective access path
Access Path Selectivity

• **Access path selectivity is the number of pages retrieved if we use this access path**
  – Most selective retrieves fewest pages

• As we saw earlier, **for equality predicates**
  – Selection on equality: $\sigma_{a=v}(R)$
  – $V(R, a) = \# \text{ of distinct values of attribute } a$
  – $1/V(R,a)$ is thus the reduction factor
  – Clustered index on $a$: cost $B(R)/V(R,a)$
  – Unclustered index on $a$: cost $T(R)/V(R,a)$
  – (we are ignoring I/O cost of index pages for simplicity)
Selectivity for Range Predicates

Selection on range: $\sigma_{a>v}(R)$

- How to compute the selectivity?
- Assume values are uniformly distributed
- Reduction factor $X$
- $X = \frac{(\text{Max}(R,a) - v)}{(\text{Max}(R,a) - \text{Min}(R,a))}$

- Clustered index on $a$: cost $B(R) \times X$
- Unclustered index on $a$: cost $T(R) \times X$
Back to Our Example

• Selection condition: \( \text{sid > 300} \land \text{scity='Seattle'} \)
  – Index I1: B+-tree on sid clustered
  – Index I2: B+-tree on scity unclustered

• Let’s assume
  – \( V(\text{Supplier}, \text{scity}) = 20 \)
  – \( \text{Max(Supplier, sid)} = 1000, \text{Min(Supplier, sid)} = 1 \)
  – \( B(\text{Supplier}) = 100, T(\text{Supplier}) = 1000 \)

• Cost I1: \( B(R) \times (\text{Max-v})/(\text{Max-Min}) = 100 \times 700/999 \approx 70 \)
• Cost I2: \( T(R) \times 1/V(\text{Supplier}, \text{scity}) = 1000/20 = 50 \)
Selectivity with Multiple Conditions

What if we have an index on multiple attributes?
• Example selection $\sigma_{a=v_1 \land b=v_2}(R)$ and index on $<a,b>$

How to compute the selectivity?
• Assume attributes are independent
• $X = \frac{1}{V(R,a) \cdot V(R,b)}$
  
  • Clustered index on $<a,b>$: cost $B(R) \cdot X$
  • Unclustered index on $<a,b>$: cost $T(R) \cdot X$
Back to Estimating Cost of a Query Plan

• We already know how to
  – Compute the cost of different operations
  – Compute cost of retrieving tuples from disk with different access paths

• We still need to
  – Compute cost of a complete plan
Computing the Cost of a Plan

• Collect statistical summaries of stored data

• Compute cost in a bottom-up fashion

• For each operator compute
  – Estimate cost of executing the operation
  – Estimate statistical summary of the output data
Statistics on Base Data

- Collected information for each relation
  - Number of tuples (cardinality)
  - Indexes, number of keys in the index
  - Number of physical pages, clustering info
  - Statistical information on attributes
    - Min value, max value, number distinct values
    - Histograms
  - Correlations between columns (hard)

- Collection approach: periodic, using sampling
Computing Cost of an Operator

• The cost of executing an operator depends
  – On the operator implementation
  – On the input data

• We learned how to compute this in the previous lecture
Statistics on the Output Data

• Most important piece of information
  – **Size of operator result**
  – I.e., the number of output tuples

• **Projection**: output size same as input size
• **Selection**: multiply input size by reduction factor
  – Similar to what we did for estimating access path selectivity
  – Assume independence between conditions in the predicate
  – (use product of the reduction factors for the terms)
Estimating Result Sizes

• For joins $R \bowtie S$
  
  – Take product of cardinalities of relations $R$ and $S$
  – Apply reduction factors for each term in join condition
  – Terms are of the form: $column1 = column2$
  – Reduction: $1/ (\text{MAX}(V(R,column1), V(S,column2)))$
  – Assumes each value in smaller set has a matching value in the larger set
Assumptions

• **Containment of values**: if $V(R,A) \leq V(S,B)$, then the set of $A$ values of $R$ is included in the set of $B$ values of $S$
  − Note: this indeed holds when $A$ is a foreign key in $R$, and $B$ is a key in $S$

• **Preservation of values**: for any other attribute $C$, $V(R \bowtie_{A=B} S, C) = V(R, C)$ (or $V(S, C)$)
Selectivity of $R \bowtie_{A=B} S$

Assume $V(R,A) \leq V(S,B)$

- Each tuple $t$ in $R$ joins with $T(S)/V(S,B)$ tuple(s) in $S$

- Hence $T(R \bowtie_{A=B} S) = T(R) \frac{T(S)}{V(S,B)}$

In general: $T(R \bowtie_{A=B} S) = T(R) \frac{T(S)}{\max(V(R,A),V(S,B))}$

CSE 544 - Fall 2015
Complete Example

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

- $M = 11$

```
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'
```
Computing the Cost of a Plan

• Estimate **cardinality** in a bottom-up fashion
  – Cardinality is the **size** of a relation (nb of tuples)
  – Compute size of *all* intermediate relations in plan

• Estimate **cost** by using the estimated cardinalities
Physical Query Plan 1

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

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

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

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

Supplier
(File scan)

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

Supply
(File scan)

(3) \( \pi_{\text{sname}} \)

(4) (On the fly)

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

\begin{align*}
T(\text{Supplier}) &= 1000 \\
B(\text{Supplier}) &= 100 \\
V(\text{Supplier, scity}) &= 20 \\
M &= 11 \\
T(\text{Supply}) &= 10,000 \\
B(\text{Supply}) &= 100 \\
V(\text{Supplier, state}) &= 10 \\
V(\text{Supply, pno}) &= 2,500
\end{align*}
Plan 2 with Different Numbers

What if we had:
- 10K pages of Suppliers
- 10K pages of Supplies

(Sort-merge join)

\( \pi_{sname} \)

\( \sigma_{sno = sno} \)

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

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

Total cost
\[= 10000 + 50 \] (1)
\[+ 10000 + 4 \] (2)
\[+ 4*50 + 2*4 + 4 + 50 \] (3)
\[+ 0 \] (4)

Total cost \( \approx 20,316 \) I/Os

Assuming naive two-pass sort algorithm

CSE 544 - Fall 2015
Physical Query Plan 3

(On the fly) (1) \( \sigma_{pno=2} \)

(2) \( \sigma_{sno = sno} \) (Index nested loop)

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

(4) \( \pi_{sname} \)

Total cost
= 1 (1) + 4 (2) + 0 (3) + 0 (3)
Total cost \( \approx 5 \) I/Os

\( T(\text{Supplier}) = 1000 \)
\( B(\text{Supplier}) = 100 \)
\( V(\text{Supplier}, \text{scity}) = 20 \)
\( M = 11 \)

\( T(\text{Supply}) = 10,000 \)
\( B(\text{Supply}) = 100 \)
\( V(\text{Supplier}, \text{state}) = 10 \)
\( V(\text{Supply}, \text{pno}) = 2,500 \)

Supply

(Hash index on pno)
Assume: clustered

Supplier

(Hash index on sno)
Clustering does not matter
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
Different Cost Models

• In previous examples, we considered IO costs

• Typically, want IO+CPU

• For parallel/distributed queries, add network bandwidth

• If need to compare *logical* plans
  – Compute the cardinality of each *intermediate* relation
  – Sum up all the cardinalities
Summary

• **What we know**
  – Different types of physical query plans
  – How to compute the cost of a query plan
  – Although it is hard to compute the cost accurately

• **We can now compare query plans**

• Let’s now consider how the query optimizer searches through the space of possible plans
Query Optimization

Three major components:

1. Cardinality and cost estimation
2. Search space
3. Plan enumeration algorithms
Relational Algebra Laws

• **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**
  – Cascading

• **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
Relational Algebra Laws

- Selects, projects, and joins
  - We can commute and combine all three types of operators
  - We just have to be careful that the fields we need are available when we apply the operator
  - Relatively straightforward. See book 15.3.

- More info in optional paper (by Chaudhuri), Section 4.
Group-by and Join

\[ R(A, B), \; S(C, D) \]
These are very powerful laws. They were introduced only in the 90’s.
Search Space Challenges

• **Search space is huge!**
  – Many possible equivalent trees (logical)
  – Many implementations for each operator (physical)
  – Many access paths for each relation (physical)

• Cannot consider ALL plans
• Want a search space that includes low-cost plans
Query Optimization

Three major components:

1. Cardinality and cost estimation
2. Search space
3. Plan enumeration algorithms
Two Types of Optimizers

• **Heuristic-based optimizers:**
  – Apply greedily rules that always improve plan
    • Typically: push selections down
  – Very limited: no longer used today

• **Cost-based optimizers:**
  – Use a cost model to estimate the cost of each plan
  – Select the “cheapest” plan
  – We focus on cost-based optimizers
Three Approaches to Search Space Enumeration

• Complete plans

• Bottom-up plans

• Top-down plans
**Complete Plans**

```
SELECT * 
FROM R, S, T 
WHERE R.B=S.B and S.C=T.C and R.A<40
```

Why is this search space inefficient?

R(A,B)  
S(B,C)  
T(C,D)
Bottom-up Partial Plans

\[
\begin{align*}
R(A, B) & \quad SELECT * \\
S(B, C) & \quad FROM R, S, T \\
T(C, D) & \quad WHERE R.B = S.B \text{ and } S.C = T.C \text{ and } R.A < 40
\end{align*}
\]

Why is this better?
Top-down Partial Plans

R(A,B)  SELECT *
S(B,C)   FROM R, S, T
T(C,D)   WHERE R.B=S.B and S.C=T.C and R.A<40

σA<40

SELECT R.A, T.D
FROM R, S, T
WHERE R.B=S.B
and S.C=T.C

.....

SELECT *
FROM R
WHERE R.A < 40

SELECT *
FROM R, S
WHERE R.B=S.B
and R.A < 40
Two Types of Plan Enumeration Algorithms

• Dynamic programming (in class)
  – Based on System R (aka Selinger) style optimizer[1979]
  – Limited to joins: *join reordering algorithm*
  – Bottom-up

• Rule-based algorithm *(will not discuss)*
  – Database of rules (=algebraic laws)
  – Usually: dynamic programming
  – Usually: *top-down*
System R Search Space

- Only left-deep plans
  - Enable dynamic programming for enumeration
  - Facilitate tuple pipelining from outer relation
- Consider plans with all “interesting orders”
- Perform cross-products after all other joins (heuristic)
- Only consider nested loop & sort-merge joins
- Consider both file scan and indexes
- Try to evaluate predicates early
Plan Enumeration Algorithm

- Idea: use dynamic programming
- For each subset of \{R_1, \ldots, R_n\}, compute the best plan for that subset
- In increasing order of set cardinality:
  - Step 1: for \{R_1\}, \{R_2\}, \ldots, \{R_n\}
  - Step 2: for \{R_1,R_2\}, \{R_1,R_3\}, \ldots, \{R_{n-1}, R_n\}
  - ...
  - Step n: for \{R_1, \ldots, R_n\}
- It is a bottom-up strategy
- A subset of \{R_1, \ldots, R_n\} is also called a subquery
Dynamic Programming Algo.

- For each subquery $Q \subseteq \{R_1, \ldots, R_n\}$ compute the following:
  - $\text{Size}(Q)$
  - A best plan for $Q$: $\text{Plan}(Q)$
  - The cost of that plan: $\text{Cost}(Q)$
Dynamic Programming Algo.

• **Step 1:** Enumerate all single-relation plans
  
  – Consider selections on attributes of relation
  – Consider all possible access paths
  – Consider attributes that are not needed
  
  – Compute cost for each plan
  
  – Keep cheapest plan per “interesting” output order
Dynamic Programming Algo.

- **Step 2**: Generate all two-relation plans
  - For each single-relation plan from step 1
  - Consider that plan as outer relation
  - Consider every other relation as inner relation
  - Compute cost for each plan
  - Keep cheapest plan per “interesting” output order
Dynamic Programming Algo.

• **Step 3**: Generate all three-relation plans
  – For each each two-relation plan from step 2
  – Consider that plan as outer relation
  – Consider every other relation as inner relation
  – Compute cost for each plan
  – Keep cheapest plan per “interesting” output order

• **Steps 4 through n**: repeat until plan contains all the relations in the query
Commercial Query Optimizers

DB2, Informix, Microsoft SQL Server, Oracle 8

• Inspired by System R
  – Left-deep plans and dynamic programming
  – Cost-based optimization (CPU and IO)

• Go beyond System R style of optimization
  – Also consider right-deep and bushy plans (e.g., Oracle and DB2)
  – Variety of additional strategies for generating plans (e.g., DB2 and SQL Server)
Other Query Optimizers

• **Randomized plan generation**
  – Genetic algorithm
  – PostgreSQL uses it for queries with many joins

• **Rule-based**
  – *Extensible* collection of rules
  – Rule = Algebraic law with a direction
  – Algorithm for firing these rules
    • Generate many alternative plans, in some order
    • Prune by cost
  – Startburst (later DB2) and Volcano (later SQL Server)