Query Optimization
Introduction to Databases
CompSci 316 Spring 2017

Announcements (Wed., Mar. 29)
• Homework #3
  • 3.3, 3.4, and 3.5 due today

Review QP and QO
• QO approaches
  • Cost-based QO
  • Heuristic (rule)-based QO
  • Query rewriting, unnesting, and decorrelation
• Equivalence of RA operators
• Estimate cost of individual operators (Lec 16 + 17)
• Estimate output size of individual operators (Lec 18)
  • Uniformity + Independence + Other assumptions
  • Selection
  • Join
• Today:
  • Explore search space for join only plans
  • Combine multiple operators

Search strategy
• Huge!
  • “Bushy” plan example:

  Just considering different join orders, there are \( \frac{(2n-2)}{(n-1)} \) bushy plans for \( R_1 \bowtie \cdots \bowtie R_n \)
  • 30240 for \( n = 6 \)
    • (FYI: Recurrence relation: \( f(n) = f(n-1) f(n-2) \), where \( 4n-6 \) comes from two ways to add the new node to one of \( n-1 \) leaves and \( n-2 \) internal nodes.)
  • And there are more if we consider:
    • Multiway joins
    • Different join methods
    • Placement of selection and projection operators

Search space

Left-deep plans
• Heuristic: consider only “left-deep” plans, in which only the left child can be a join
  • Tend to be better than plans of other shapes, because many join algorithms scan inner (right) relation multiple times—you will not want it to be a complex subtree
• How many left-deep plans are there for \( R_1 \bowtie \cdots \bowtie R_n \)?
  • Significantly fewer, but still lots—\( n! \) (720 for \( n = 6 \))
A greedy algorithm

• $S_1, ..., S_m$
  • Say selections have been pushed down; i.e., $S_i = s_i(R_i)$
  • Start with the pair $S_i, S_j$ with the smallest estimated size for $S_i \bowtie S_j$
  • Repeat until no relation is left:
    Pick $S_k$ from the remaining relations such that the join of $S_k$ and the current result yields an intermediate result of the smallest size

A dynamic programming approach

• Selinger's algorithm
  • IBM System R, frequently adapted and used
  • Generate optimal plans bottom-up
    • Pass 1: Find the best single-table plans (for each table)
    • Pass 2: Find the best two-table plans (for each pair of tables) by combining best single-table plans
    • ...
    • Pass $k$: Find the best $k$-table plans (for each combination of $k$ tables) by combining two smaller best plans found in previous passes
  • ...
  • Rationale: Any subplan of an optimal plan must also be optimal
    • otherwise, just replace the subplan to get a better overall plan

Principle of Optimality

Query: $R_1 \bowtie R_2 \bowtie R_3 \bowtie R_4 \bowtie R_5$

Then, what can you say about this sub-plan?

Suppose, this is an Optimal Plan for joining $R_1$...$R_5$:

This has to be the optimal plan for joining $R_3$, $R_2$, $R_4$, $R_1$

Principle of Optimality

Query: $R_1 \bowtie R_2 \bowtie R_3 \bowtie R_4 \bowtie R_5$

Then, what can you say about this sub-plan?

We are using the associativity and commutativity of joins

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

Suppose, this is an Optimal Plan for joining $R_1$...$R_5$:

This has to be the optimal plan for joining $R_3$, $R_2$, $R_4$

Exploiting Principle of Optimality

Query: $R_1 \bowtie R_2 \bowtie ... \bowtie R_n$

Both are giving the same result

$R_2 \bowtie R_3 \bowtie R_1 = R_3 \bowtie R_1 \bowtie R_2$

Optimal for joining $R_1$, $R_2$, $R_3$

Sub-Optimal for joining $R_1$, $R_2$, $R_3$
Exploiting Principle of Optimality

Suppose you chose the sub-optimal one

Leads to sub-Optimal for joining R1,...,Rn

A sub-optimal sub-plan cannot lead to an optimal plan

Selinger Algorithm:

\[
\text{OPT} \left( \{ R_1, R_2, R_3 \} \right):
\]\[
\begin{cases}
\text{OPT} \left( \{ R_1, R_2 \} \right) + \text{Join} (\{(R_1, R_2), R_3\}) \\
\text{Min}
\end{cases}
\]

Selinger Algorithm:

Query: \( R_1 \bowtie R_2 \bowtie R_3 \bowtie R_4 \)

\[
\begin{align*}
\text{OPT} \left( \{ R_1, R_2, R_3, R_4 \} \right) \\
= & \text{OPT} \left( \{ R_1, R_2, R_3 \} \right) + \text{Join} (\{(R_1, R_2, R_3), R_4\}) \\
= & \text{OPT} \left( \{ R_1, R_3, R_4 \} \right) + \text{Join} (\{(R_1, R_3), R_2\}) \\
= & \text{OPT} \left( \{ R_2, R_3, R_4 \} \right) + \text{Join} (\{(R_2, R_3), R_1\})
\end{align*}
\]

Selinger Algorithm:

Query: \( R_1 \bowtie R_2 \bowtie R_3 \bowtie R_4 \)

\[
\begin{align*}
\text{OPT} \left( \{ R_1, R_2, R_3, R_4 \} \right) \\
= & \text{OPT} \left( \{ R_1, R_3 \} \right) + \text{Join} (\{(R_1, R_3), R_2\}) \\
= & \text{OPT} \left( \{ R_1, R_4 \} \right) + \text{Join} (\{(R_1, R_4), R_3\}) \\
= & \text{OPT} \left( \{ R_2, R_4 \} \right) + \text{Join} (\{(R_2, R_4), R_1\})
\end{align*}
\]

Selinger Algorithm:

Query: \( R_1 \bowtie R_2 \bowtie R_3 \bowtie R_4 \)

\[
\begin{align*}
\text{OPT} \left( \{ R_1, R_2, R_3, R_4 \} \right) \\
= & \text{OPT} \left( \{ R_1, R_2, R_3 \} \right) + \text{Join} (\{(R_1, R_2, R_3), R_4\}) \\
= & \text{OPT} \left( \{ R_1, R_3, R_4 \} \right) + \text{Join} (\{(R_1, R_3), R_2\}) \\
= & \text{OPT} \left( \{ R_2, R_3, R_4 \} \right) + \text{Join} (\{(R_2, R_3), R_1\})
\end{align*}
\]

Selinger Algorithm:

Query: \( R_1 \bowtie R_2 \bowtie R_3 \bowtie R_4 \)

\[
\begin{align*}
\text{OPT} \left( \{ R_1, R_2, R_3, R_4 \} \right) \\
= & \text{OPT} \left( \{ R_1, R_2, R_3 \} \right) + \text{Join} (\{(R_1, R_2, R_3), R_4\}) \\
= & \text{OPT} \left( \{ R_1, R_3, R_4 \} \right) + \text{Join} (\{(R_1, R_3), R_2\}) \\
= & \text{OPT} \left( \{ R_2, R_3, R_4 \} \right) + \text{Join} (\{(R_2, R_3), R_1\})
\end{align*}
\]
Selinger Algorithm:

Query: \( R_1 \bowtie R_2 \bowtie R_3 \bowtie R_4 \)

Progress of algorithm

NOTE: There is a one-one correspondence between the permutation \((R_3, R_1, R_4, R_2)\) and the above left deep plan

Summary: A dynamic programming approach

- Selinger’s algorithm
- Generate optimal plans bottom-up
- Rationale: Any subplan of an optimal plan must also be optimal (otherwise, just replace the subplan to get a better overall plan)

Well, not quite…
- for physical plan

The need for “interesting order”

- Example: \( R(A, B) \bowtie S(A, C) \bowtie T(A, D) \)
- Best plan for \( R \bowtie S \): hash join (beats sort-merge join)
- Best overall plan: sort-merge join \( R \) and \( S \), and then sort-merge join with \( T \)
- Subplan of the optimal plan is not optimal!
- Why?
  - The result of the sort-merge join of \( R \) and \( S \) is sorted on \( A \)
  - This is an interesting order that can be exploited by later processing (e.g., join, dup elimination, GROUP BY, ORDER BY, etc.).
Dealing with interesting orders

When picking the best plan

• Comparing their costs is not enough
  • Plans are not totally ordered by cost anymore
• Comparing interesting orders is also needed
  • Plans are now partially ordered
  • Plan X is better than plan Y if
    • Cost of X is lower than Y, and
    • Interesting orders produced by X “subsume” those produced by Y
• Need to keep a set of optimal plans for joining every combination of k tables
  • At most one for each interesting order

Combine cost of different operators in a plan

• Given a Physical Plan
  • Size = #tuples, NOT #blocks
  • Cost = #page or block I/O
    • but, need to consider whether the intermediate relation fits in memory, is written back to disk (or on-the-fly goes to the next operator) etc.

Example Query

Student (sid, name, age, address)
Book(bid, title, author)
Checkout(sid, bid, date)

Query:

SELECT S.name
FROM Student S, Book B, Checkout C
WHERE S.sid = C.sid
AND B.bid = C.bid
AND B.author = 'Olden Fames'
AND S.age > 12
AND S.age < 20

Assumptions

• Student: S, Book: B, Checkout: C
  • Sid, bid foreign key in C referencing S and B resp.
  • There are 10,000 Student records stored on 1,000 blocks.
  • There are 50,000 Book records stored on 5,000 blocks.
  • There are 300,000 Checkout records stored on 15,000 blocks.
  • There are 500 different authors.
  • Student ages range from 7 to 24.

  • T(R) = no. of tuples
    • R(R) = total cost
  • V(R,A) = no. of distinct values of attributes A of R

Physical Query Plan – 1

(Tuple-based nested loop
B inner)
(Page-oriented
-nested loop,
S outer, C inner)

Q. Compute
1. the cost and cardinality in steps (a) to (d)
2. the total cost

Assumptions:
• Data is not sorted on any attributes
• For both in (a) and (b), outer relations fit in memory

Cost =

Cardinality =

(b)

T(C) = 300,000

• foreign key join, output pipelined to next join
• Can apply the formula as well

T(S) * T(C) * max (V(S, sid), V(C, sid))
T(S) since V(S, sid) >= V(S, sid)
T(C) and

(On the fly) (d) IT name

(a)
null
Summary

• Relational algebra equivalence
• SQL rewrite tricks
• Heuristics-based optimization
• Cost-based optimization
  • Need statistics to estimate sizes of intermediate results
  • Greedy approach
  • Dynamic programming approach
• Combination of cost and size estimation along a plan