Query Optimization

Introduction to Databases
CompSci 316 Spring 2017
Announcements (Mon., Mar. 27)

• Homework #3
  • 3.3, 3.4, and 3.5 due Wednesday, March 29

• Project
  • Milestone 2 due today

• Submit both on sakai
Query optimization (QO)

• One logical plan → “best” physical plan

• Questions
  • How to enumerate possible plans
  • How to estimate costs
  • How to pick the “best” one

• Often the goal is not getting the optimum plan, but instead avoiding the horrible ones. Why?
  • want to execute only one
  • need to estimate cost without executing the plan

Any of these will do

1 second  1 minute  1 hour
Steps for cost based QO

• Tasks:
  1. Estimate the cost of individual operators -- done
  2. Estimate the size of output of individual operators
  3. Combine costs of different operators in a plan – next lecture
  4. Efficiently search the space of plans
Plan enumeration in relational algebra

• Apply relational algebra equivalences

ά Join reordering: $\times$ and $\bowtie$ are associative and commutative (except column ordering, but that is unimportant)
More relational algebra equivalences

• Convert $\sigma_p \times$ to/from $\Join_p$: $\sigma_p (R \times S) = R \Join_p S$

• Merge/split $\sigma$’s: $\sigma_{p_1} (\sigma_{p_2} R) = \sigma_{p_1 \land p_2} R$

• Merge/split $\pi$’s: $\pi_{L_1} (\pi_{L_2} R) = \pi_{L_1} R$, where $L_1 \subseteq L_2$

• Push down/pull up $\sigma$:
  $\sigma_{p \land p_r \land p_s} (R \Join_{p'} S) = (\sigma_{p_r} R) \Join_{p \land p'} (\sigma_{p_s} S)$, where
  • $p_r$ is a predicate involving only $R$ columns
  • $p_s$ is a predicate involving only $S$ columns
  • $p$ and $p'$ are predicates involving both $R$ and $S$ columns

• Push down $\pi$: $\pi_L (\sigma_p R) = \pi_L \left( \sigma_p (\pi_{L'L'} R) \right)$, where
  • $L'$ is the set of columns referenced by $p$ that are not in $L$

• Many more (seemingly trivial) equivalences...
  • Can be systematically used to transform a plan to new ones
Relational query rewrite example

\[ \pi_{\text{Group.name}} \quad \sigma_{\text{User.name} = "Bart" \land \text{User.uid} = \text{Member.uid} \land \text{Member.gid} = \text{Group.gid}} \times \text{Group} \times \text{User} \]

Push down \( \sigma \)

Convert \( \sigma_p \times \) to \( \bowtie_p \)
Heuristics-based query optimization

• Start with a logical plan
• Push selections/projections down as much as possible (why/why not?)
  • Why? Reduce the size of intermediate results
  • Why not? May be expensive; maybe joins filter better
• Join smaller relations first, and avoid cross product (why/why not?)
  • Why? Reduce the size of intermediate results
  • Why not? Size depends on join selectivity too
• Convert the transformed logical plan to a physical plan (by choosing appropriate physical operators)
SQL query rewrite

• More complicated—subqueries and views divide a query into nested “blocks”
  • Processing each block separately forces particular join methods and join order
  • Even if the plan is optimal for each block, it may not be optimal for the entire query

• Unnest query: convert subqueries/views to joins
  We can just deal with select-project-join queries
  • Where the clean rules of relational algebra apply
SQL query rewrite example

• SELECT name
  FROM User
  WHERE uid = ANY (SELECT uid FROM Member);

• SELECT name
  FROM User, Member
  WHERE User.uid = Member.uid;
    • Wrong—consider two Bart’s, each joining two groups

• SELECT name
  FROM (SELECT DISTINCT User.uid, name
        FROM User, Member
        WHERE User.uid = Member.uid);
    • Right—assuming User.uid is a key

User(uid, name)
Member(uid, gid)
Dealing with correlated subqueries

• SELECT gid FROM Group
  WHERE name LIKE 'Springfield%'
  AND min_size > (SELECT COUNT(*) FROM Member
      WHERE Member.gid = Group.gid);

• SELECT gid
  FROM Group, (SELECT gid, COUNT(*) AS cnt
      FROM Member GROUP BY gid) t
  WHERE t.gid = Group.gid AND min_size > t.cnt
  AND name LIKE 'Springfield%';

• New subquery is inefficient (it computes the size for every group)
• Suppose a group is empty?
“Magic” decorrelation

- SELECT gid FROM Group
  WHERE name LIKE 'Springfield%'
  AND min_size > (SELECT COUNT(*) FROM Member
    WHERE Member.gid = Group.gid);

- WITH Supp_Group AS (SELECT * FROM Group WHERE name LIKE 'Springfield%'),
  Magic AS (SELECT DISTINCT gid FROM Supp_Group),
  DS AS ((SELECT Group.gid, COUNT(*) AS cnt
    FROM Magic, Member WHERE Magic.gid = Member.gid
    GROUP BY Member.gid) UNION
    (SELECT gid, 0 AS cnt
    FROM Magic WHERE gid NOT IN (SELECT gid FROM Member)))

SELECT Supp_Group.gid FROM Supp_Group, DS
  WHERE Supp_Group.gid = DS.gid
  AND min_size > DS.cnt;

  Process the outer query without the subquery
  Collect bindings
  Evaluate the subquery with bindings
  Finally, refine the outer query
Heuristics- vs. cost-based optimization

• Heuristics-based optimization
  • Apply heuristics to rewrite plans into cheaper ones

• Cost-based optimization
  • Rewrite logical plan to combine “blocks” as much as possible
  • Optimize query block by block
    • Enumerate logical plans (already covered)
    • Estimate the cost of plans
    • Pick a plan with acceptable cost
  • Focus: select-project-join blocks
Cost estimation

Physical plan example:

- We have: cost estimation for each operator
  - Example: SORT\((gid)\) takes \(O(B\text{ (input)} \times \log_M B\text{ (input)})\)
    - But what is \(B\text{ (input)}\)?
- We need: size of intermediate results
Cardinality estimation
Selections with equality predicates

- $Q$: $\sigma_{A=v} R$

Suppose the following information is available

- Size of $R$: $|R|$
- Number of distinct $A$ values in $R$: $V(R, A) = |\pi_A R|$

Assumptions

- Values of $A$ are uniformly distributed in $R$
- Values of $v$ in $Q$ are uniformly distributed over all $R.A$ values

- $|Q| \approx \frac{|R|}{|\pi_A R|}$
- Selectivity factor of $(A = v)$ is $\frac{1}{|\pi_A R|}$
 Conjunctive predicates

• Q: \( \sigma_{A=u} \land B=v^R \)

• Additional assumptions
  • \((A = u)\) and \((B = v)\) are independent
    • Counterexample: major and advisor
    • Counterexample: \(A = 10\) and \(A > 30\)
  • No “over”-selection
    • Counterexample: \(A\) is the key

• \(|Q| \approx \frac{|R|}{|\pi_A R| \cdot |\pi_B R|} \)
  • Reduce total size by all selectivity factors
Negated and disjunctive predicates

- \( Q: \sigma_{A \neq v} R \)
  - \( |Q| \approx |R| \cdot \left( 1 - \frac{1}{|\pi_A R|} \right) \)
  - Selectivity factor of \( \neg p \) is \((1 - \text{selectivity factor of } p)\)

- \( Q: \sigma_{A = u \lor B = v} R \)
  - \( |Q| \approx |R| \cdot \left( \frac{1}{|\pi_A R|} + \frac{1}{|\pi_B R|} \right) \)?
  - No! Tuples satisfying \( (A = u) \) and \( (B = v) \) are counted twice
  - \( |Q| \approx |R| \cdot \left( \frac{1}{|\pi_A R|} + \frac{1}{|\pi_B R|} - \frac{1}{|\pi_A R||\pi_B R|} \right) \)
  - Inclusion-exclusion principle
Range predicates

- \( Q: \sigma_{A>v}R \)
- Not enough information!
  - Just pick, say, \( |Q| \approx |R| \cdot \frac{1}{3} \)
- With more information
  - Largest R.A value: \( \text{high}(R.A) \)
  - Smallest R.A value: \( \text{low}(R.A) \)
  - \( |Q| \approx |R| \cdot \frac{\text{high}(R.A)-v}{\text{high}(R.A)-\text{low}(R.A)} \)
- In practice: sometimes the second highest and lowest are used instead. Why?
  - The highest and the lowest are often used by inexperienced database designer to represent invalid values!
Two-way equi-join

- \( Q: R(A, B) \bowtie S(A, C) \)

- Assumption: containment of value sets
  - Every tuple in the “smaller” relation (one with fewer distinct values for the join attribute) joins with some tuple in the other relation
  - That is, if \( |\pi_A R| \leq |\pi_A S| \) then \( \pi_A R \subseteq \pi_A S \)
  - Certainly not true in general
  - But holds in the common case of foreign key joins

- \(|Q| \approx \frac{|R| \cdot |S|}{\max(|\pi_A R|, |\pi_A S|)}\)
  - Selectivity factor of \( R. A = S. A \) is \( \frac{1}{\max(|\pi_A R|, |\pi_A S|)} \)
Multiway equi-join

• $Q: R(A, B) \bowtie S(B, C) \bowtie T(C, D)$

• What is the number of distinct $C$ values in the join of $R$ and $S$?

• Assumption: preservation of value sets
  • A non-join attribute does not lose values from its set of possible values
  • That is, if $A$ is in $R$ but not $S$, then $\pi_A(R \bowtie S) = \pi_A R$
  • Certainly not true in general
  • But holds in the common case of foreign key joins (for value sets from the referencing table)
Multiway equi-join (cont’d)

• \( Q: R(A, B) \bowtie S(B, C) \bowtie T(C, D) \)

• Start with the product of relation sizes
  • \(|R| \cdot |S| \cdot |T|\)

• Reduce the total size by the selectivity factor of each join predicate
  • \( R.B = S.B: \frac{1}{\max(|\pi_B R|, |\pi_B S|)} \)
  • \( S.C = T.C: \frac{1}{\max(|\pi_C S|, |\pi_C T|)} \)
  • \( |Q| \approx \frac{|R| \cdot |S| \cdot |T|}{\max(|\pi_B R|, |\pi_B S|) \cdot \max(|\pi_C S|, |\pi_C T|)} \)
Cost estimation: summary

• Using similar ideas, we can estimate the size of projection, duplicate elimination, union, difference, aggregation (with grouping)

• Lots of assumptions and very rough estimation
  • Accurate estimate is not needed
  • Maybe okay if we overestimate or underestimate consistently
  • May lead to very nasty optimizer “hints”

    SELECT * FROM User WHERE pop > 0.9;
    SELECT * FROM User WHERE pop > 0.9 AND pop > 0.9;

• Not covered: better estimation using histograms
Search strategy
Search space

• 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 \( n = 6 \)
  • Recurrence relation: \( f(n) = f(n-1) \times (4n-6) \), 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
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 $n = 6$)