Query Optimization
Introduction to Databases
CompSci 316 Spring 2017

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?

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

More relational algebra equivalences
• Convert $\sigma_p \times$ to/from $\mu_p$: $\sigma_p (R \times S) = R \mu_p S$
• Merge/split $\sigma$’s: $\sigma_{p_1} (\sigma_{p_2} R) = \sigma_{p_1 \lor p_2} R$
• Merge/split $\pi$’s: $\pi_{L_1} (\pi_{L_2} R) = \pi_{L_1} \pi_{L_2} R$, where $L_1 \subseteq L_2$
• Push down/pull up $\sigma$: $\sigma_{\mu_{p_1}, \mu_{p_2}}(R \times S) = (\sigma_{p_1} R) \mu_{\pi_{p_2}}(\sigma_{p_2} S)$, where
  • $p_1$ is a predicate involving only $R$ columns
  • $p_2$ is a predicate involving only $S$ columns
  • $p$ and $p'$ are predicates involving both $R$ and $S$ columns
• Push down $\pi$: $\pi_L (\sigma_p R) = \pi_{L'} (\sigma_p (\pi_{L'} R))$, 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

Heuristics-based query optimization

SQL query rewrite

Dealing with correlated subqueries

“Magic” decorrelation
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:

- **Input to SORT gid**:
  - **MERGE-JOIN (uid)**
  - **SORT (uid)**
  - **SCAN (Member)**
  - **FILTER (name = "Bart")**
- **Output**:
  - **PROJECT (Group.title)**
  - **MERGE-JOIN (gid)**
  - **SORT (gid)**
  - **SCAN (Group)**

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

Cardinality estimation

![Image](http://www.learningresources.com/product/estimation+station.do)

Selections with equality predicates

- \( Q: \sigma_{A=B} R \)
  - Suppose the following information is available
    - Size of \( R \): \(|R|\)
    - Number of distinct \( A \) values in \( R \): \( \text{V}(R, A) = |\pi_A R| \)
  - Assumptions
    - Values of \( A \) are uniformly distributed in \( R \)
    - Values of \( v \) in \( Q \) are uniformly distributed over all \( R \)
    - \( A \) values
  - **Selectivity factor of** \( A = v \) is \( \frac{1}{|\pi_A R|} \)

Conjunctive predicates

- \( Q: \sigma_{A=B = v} R \)
  - 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
  - \(|Q| \approx \frac{|R|}{|\pi_AR|} \cdot \frac{1}{|\pi_B R|} \)
  - Reduce total size by all selectivity factors

Negated and disjunctive predicates

- \( Q: \sigma_{A=B = v} R \)
  - \(|Q| \approx |R| \cdot (1 - \frac{1}{|\pi_AR|}) \)
  - Selectivity factor of \( \neg p \) is \((1 - \text{selectivity factor of } p)\)
- \( Q: \sigma_{A=B} R \)
  - \(|Q| \approx |R| \cdot \left(\frac{1}{|\pi_AR|} + \frac{1}{|\pi_B R|}\right)\)
Range predicates

- \( Q: \sigma_{A \geq p} R \)
- Not enough information!
  - Just pick, say, \(|Q| \approx |R| \cdot \frac{1}{3}\)
- With more information
  - Largest RA value: \(\text{high}(R, A)\)
  - Smallest RA value: \(\text{low}(R, A)\)
  - \(|Q| \approx |R| \cdot \frac{\text{high}(R, A)}{\text{low}(R, A)}\)
  - In practice: sometimes the second highest and lowest are used instead. Why?

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
  - \(R.B = S.B: \frac{1}{\max(|\pi_B R|, |\pi_B S|)}\)
  - \(S.C = T.C: \frac{1}{\max(|\pi_C S|, |\pi_C T|)}\)
  - \(|Q| \approx \frac{|R| \cdot |S| \cdot |T|}{\max(|\pi_B R|, |\pi_B S|, |\pi_C S|, |\pi_C T|)}\)

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 $\binom{2n-2}{n-1}$ bushy plans for $R_1 \bowtie \cdots \bowtie R_n$
- 30240 for $n = 6$
- Recurrence relation: $f(n) = f(n-1)(4n-6)$, 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

A greedy algorithm

- $S_2, \ldots, S_n$
- Say selections have been pushed down; i.e., $S_i = \sigma_{\pi_i}(R_i)$
- Start with the pair $S_1, S_j$ with the smallest estimated size for $S_1 \bowtie S_j$
- 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
  - Minimize expected size
  - Pick most efficient join method

A dynamic programming approach

- Selinger’s algorithm
  - 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
    - …
  - Rational: Any subplan of an optimal plan must also be optimal (otherwise, just replace the subplan to get a better overall plan)
- $\text{OPT}([R_1, R_2]) + \text{Join}([\text{OPT}([R_1, R_2]), R_3])$
- $\text{OPT}([R_2, R_3]) + \text{Join}([\text{OPT}([R_2, R_3]), R_1])$
- $\text{OPT}([R_3, R_1]) + \text{Join}([\text{OPT}([R_3, R_1]), R_2])$
- “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 (suppose it 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?

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
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