Query Optimization

CPS 116
Introduction to Database Systems

Announcements

- Homework #4 assigned today (Nov. 18); due in two weeks (Dec. 2)
- Student presentation on Dec. 2 on databases for small devices
  - Allows your lowest homework grade to be dropped
  - Need more 1-2 more volunteers

Query optimization

- 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

Any of these will do:

1 second 1 minute 1 hour
Plan enumeration in relational algebra

- Apply relational algebra equivalences
  - Join reordering: $\times$ and $\triangleright\triangleright$ are associative and commutative (except column ordering, but that is unimportant)

![Diagram of relational algebra operations]

More relational algebra equivalences

- Convert $\sigma_p \times \sigma_q$ to/from $\sigma_{p \land q}$: $\sigma_p(R \times S) = R \triangleright\triangleright_p S$
- Merge/split $\sigma$: $\sigma_p(R) = \sigma_{p_1 \land p_2} R$
- Merge/split $\pi$: $\pi_{L_1}(\pi_{L_2} R) = \pi_{L_1} R$, where $L_1 \subseteq L_2$
- Push down/pull up $\sigma$: $\pi_{L}(\sigma_p R) = \pi_{\pi(L')} \sigma_p \pi(R)$, 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

- Convert $\sigma_p \times \sigma_q$ to/from $\sigma_{p \land q}$
- Push down $\sigma$: $\pi_{E_{\text{Title}}}$
- Convert $\sigma_p \times \sigma_q$ to/from $\sigma_{p \land q}$
- Push down $\sigma$: $\sigma_{E_{\text{Name}}}$
- Convert $\sigma_p \times \sigma_q$ to/from $\sigma_{p \land q}$
Heuristics-based query optimization

- Start with a logical plan
- Push selections/projections down as much as possible
  - Why?
  - Why not?
- Join smaller relations first, and avoid cross product
  - Why?
  - Why not?
- 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 Student
  WHERE SID = ANY (SELECT SID FROM Enroll);
- SELECT name
  FROM Student, Enroll
  WHERE Student.SID = Enroll.SID;
  - Wrong
- SELECT name
  FROM (SELECT DISTINCT Student.SID, name
  FROM Student, Enroll
  WHERE Student.SID = Enroll.SID);
  - Right
Dealing with correlated subqueries

- SELECT CID FROM Course
  WHERE title LIKE 'CPS%'
  AND min_enroll > (SELECT COUNT(*) FROM Enroll
  WHERE Enroll.CID = Course.CID);

- SELECT CID
  FROM Course, (SELECT CID, COUNT(*) AS cnt
  FROM Enroll GROUP BY CID) t
  WHERE t.CID = Course.CID AND min_enroll > t.cnt
  AND title LIKE 'CPS%';

  - New subquery is inefficient (computes enrollment for all courses)
  - Suppose

“Magic” decorrelation

- SELECT CID FROM Course
  WHERE title LIKE 'CPS%'
  AND min_enroll > (SELECT COUNT(*) FROM Enroll
  WHERE Enroll.CID = Course.CID);

- CREATE VIEW Supp_Course AS
  SELECT * FROM Course WHERE title LIKE 'CPS%';

- CREATE VIEW Magic AS
  SELECT DISTINCT CID FROM Supp_Course;

- CREATE VIEW DS AS
  (SELECT Enroll.CID, COUNT(*) AS cnt
  FROM Magic, Enroll WHERE Magic.CID = Enroll.CID
  GROUP BY Enroll.CID) UNION
  (SELECT Magic.CID, 0 AS cnt FROM Magic
  WHERE Magic.CID NOT IN (SELECT CID FROM Enroll));

- SELECT Supp_Course.CID FROM Supp_Course, DS
  WHERE Supp_Course.CID = DS.CID
  AND min_enroll = 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:

- PROJECT (title)
- MERGE-JOIN (CID)
- MERGE-JOIN (SID)
- SCAN (Course)
- SCAN (Enroll)
- SCAN (Student)
- FILTER (name = "Bart")

- Example: SORT (GID) takes $2 \times B(input)$
- But what is $B(input)$?

- We need: size of intermediate results

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 $R$: $|\pi_A R|$
- Assumptions
  - Values of $A$ are uniformly distributed in $R$
  - Values of $v$ in $Q$ are uniformly distributed over all $RA$ values
- $|Q| \approx |R|/|\pi_A R|$
  - Selectivity factor of $(A = v)$ is $1/|\pi_A R|$

Conjunctive predicates

- $Q$: $\sigma_A = a$ and $B = v R$
- Additional assumptions
  - $(A = a)$ and $(B = v)$ are independent
    - Counterexample: major and advisor
  - No "over"-selection
    - Counterexample: $A$ is the key
- $|Q| \approx |R|/\left( |\pi_A R| \cdot |\pi_B R| \right)$
  - Reduce total size by all selectivity factors
Negated and disjunctive predicates

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

\( Q: \sigma_{A = u \text{ or } B = v} R \)
- \(|Q| \approx |R| \cdot (1/|\pi_A R| + 1/|\pi_B R|)\)
  - No! Tuples satisfying \((A = u)\) and \((B = v)\) are counted twice
  - Intuition: \((A = u)\) or \((B = v)\) is equivalent to \(\neg (\neg (A = u) \text{ AND } \neg (B = v))\)

Range predicates

\( Q: \sigma_{A > v} R \)
- Not enough information!
  - Just pick, say, \(|Q| \approx |R| \cdot 1/3\)
- With more information
  - Largest \( R.A \) value: \( \text{high}(R.A) \)
  - Smallest \( R.A \) value: \( \text{low}(R.A) \)
  - \(|Q| \approx |R| \cdot (\text{high}(R.A) - v) / (\text{high}(R.A) - \text{low}(R.A))\)
  - In practice: sometimes the second highest and lowest are used instead
    - 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
- \(|Q| \approx |R| \cdot |S| / \max(|\pi_A R|, |\pi_A S|)\)
  - Selectivity factor of \(R.A = S.A\) is \(1/\max(|\pi_A R|, |\pi_A S|)\)
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

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”

\[
\begin{align*}
\text{SELECT} & \quad \text{FROM Student WHERE GPA > 3.9;} \\
\text{SELECT} & \quad \text{FROM Student WHERE GPA > 3.9 AND GPA > 3.9;}
\end{align*}
\]
- Not covered: better estimation using histograms
Search for the best plan

- Huge search space
- "Bushy" plan example:
  - Just considering different join orders, there are close to \((n - 1)! \cdot 4^{n-1}\) bushy plans for \(R_1 \bowtie \cdots \bowtie R_n\)
    - 30240 for \(n = 6\)
- 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

A greedy algorithm

- \(S_1, \ldots, S_n\)
  - Say selections have been pushed down; i.e., \(S_j = \sigma_{R_i}\)
- Start with the pair \(S_j, S_k\) with the smallest estimated size for \(S_j \bowtie S_k\)
- Repeat until no relation is left:
  - Pick \(S_j\) from the remaining relations such that the join of \(S_j\) and the current result yields an intermediate result of the smallest size
  - Pick most efficient join method
  - Minimize expected size
  - Current subplan
  - Remaining relations to be joined
A dynamic programming approach

- 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)
  - Well, not quite…

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, duplicate 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 \( Y \)
      - 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