Query Optimization

CompSci 316
Introduction to Database Systems

Announcements (Tue. Nov. 27)

- Homework #4 due in a week
- Sign up (via email) for a 30-minute slot in the project demo period, Dec. 10-12
  - “Public” demo slots available on Dec. 6
- Final exam 2-5pm Dec. 12
  - Open book, open notes
  - Focus on the second half of the course
  - Sample final available soon

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
Plan enumeration in relational algebra

- Apply relational algebra equivalences
- 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 \( \bowtie p \): \( \sigma_p (R \times S) = R \bowtie_p S \)
- Merge/split \( \sigma \)'s: \( \sigma_{p_1} (\sigma_{p_2} R) = \sigma_{p_1 \cup p_2} R \)
- Merge/split \( \pi \)'s: \( \pi_{L_1} (\pi_{L_2} R) = \pi_{L_1 \cup L_2} R \)
- Push down/pull up \( \sigma \):
  - \( \sigma_{p \cap p'} (R \bowtie_p S) = (\sigma_{p_1} R) \bowtie_{p \cap p'} (\sigma_{p_2} S) \), 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 \( \pi \):
  - \( \pi_{L_1} (\sigma_p R) = \pi_{L_1} (\sigma_{p \cap L'} R) \), 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

- 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?
- 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—
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%';

“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;

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 (role)
- MERGE-JOIN (CID)
- SCAN (Course)
- FILTER (name = "Bart")
- SCAN (Student)
- SORT (CID)
- MERGE-JOIN (SID)

Input to SORT(CID):
- SCAN (Enroll)
- SCAN (Course)
- SORT (SID)

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

We have: cost estimation for each operator

- Example: SORT(CID) takes $O(B(\text{input}) \times \log B(\text{input}))$
  - But what is $B(\text{input})$?

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 $R.A$ values
- $|Q| \approx |R|/|\pi_A R|$
  - Selectivity factor of $(A = v)$ is $1/|\pi_A R|$

Conjunctive predicates

- $Q: \sigma_{A=u \land B=v} R$
- Additional assumptions
  - $(A = u)$ and $(B = v)$ are independent
  - Counterexample: major and advisor
  - No "over"-selection
    - Counterexample: $A$ is the key
Negated and disjunctive predicates

\[ Q : \sigma_{A \neq \emptyset} R \]
- \[ |Q| \approx |R| \cdot \left(1 - \frac{1}{|\pi_A R|}\right) \]
  - Selectivity factor of \( \neg \phi \) is \( 1 - \) selectivity factor of \( \phi \)

\[ Q : \sigma_{A \cup B = \emptyset} R \]
- \[ |Q| \approx |R| \cdot \left(\frac{1}{|\pi_A R|} + \frac{1}{|\pi_B R|}\right)^2 \]

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: high\( (R.A) \)
  - Smallest \( R.A \) value: low\( (R.A) \)
  - \[ |Q| \approx |R| \cdot \frac{high(R.A) - v}{high(R.A) - low(R.A)} \]
  - In practice: sometimes the second highest and lowest are used instead

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

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

\( \blacklozenge \) What is the number of distinct \( C \) values in the join of \( R \) and \( S \) ?

\( \blacklozenge \) 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)

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

\( \blacklozenge \) Start with the product of relation sizes

\[ |R| \cdot |S| \cdot |T| \]

\( \blacklozenge \) Reduce the total size by the selectivity factor of each join predicate

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

Cost estimation: summary

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

\( \blacklozenge \) 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"  
  \[ \text{SELECT} * \text{FROM Student WHERE GPA} > 3.9; \]
  \[ \text{SELECT} * \text{FROM Student WHERE GPA} > 3.9 \text{ AND GPA} > 3.9; \]
- Not covered: better estimation using histograms
Search for the best plan

- Huge search space
- "Bushy" plan example:

  - Just considering different join orders, there are \((2n-1)^n\) 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 \(R_1 \bowtie \cdots \bowtie R_n\)?

A greedy algorithm

- \(S_1, \ldots, S_n\)
  - Say selections have been pushed down; i.e., \(S_i = \sigma_{p_i}(R_i)\)
  - 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
    - 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, 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
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