Query Optimization

CPS 116
Introduction to Database Systems

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

Plan enumeration in relational algebra

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

More relational algebra equivalences

- Convert $\sigma_p \times$ to/from $\times$:
  $\sigma_p (R \times S) = R \sigma_p S$
- Merge/split $\sigma$: $\sigma_p (\sigma_{p'} R) = \sigma_{p \land p'} R$, where $L1 \subseteq L2$
- Join smaller relations first, and avoid cross product
- Convert the transformed logical plan to a physical plan (by choosing appropriate physical operators)

Relational query rewrite example

Heuristics-based query optimization

- Start with a logical plan
- Push selections/projections down as much as possible
  - 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? 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

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 a CPS class is empty?

“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)
SORT (Course)
SCAN (Enroll)
```

- We have: cost estimation for each operator
- Example: SORT(CID) takes $2 \times \text{Input}$
  - But what is $\text{Input}$?
- 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 $1 / |\pi_A R|$

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 $(1 - \text{selectivity factor of } p)$
- $Q: \sigma_A = a \text{ or } B = v R$
  - $|Q| \approx |R| \cdot (1 / |\pi_A R| + 1 / |\pi_B R|)$
  - Not tuples satisfying $(A = a)$ and $(B = v)$ are counted twice
  - $|Q| \approx |R| \cdot (1 - (1 - 1 / |\pi_A R|) \cdot (1 - 1 / |\pi_B R|))$
  - Intuition: $(A = a)$ or $(B = v)$ is equivalent to $\neg (\neg (A = a) \text{ AND } \neg (B = v))$

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 $|\pi_x R| \leq |\pi_x S|$ then $\pi_x R \subseteq \pi_x S$
  - 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 $1 / \max(|\pi_A R|, |\pi_A S|)$

Conjunctive predicates

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

Range predicates

- $Q: (A = a) \text{ and } (B = v)$
  - Not enough information!
    - Just pick, say, $|Q| \approx |R| \cdot 1/3$
  - With more information
    - Largest $R.A$ value: high($R.A$)
    - Smallest $R.A$ value: 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!

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
  - $R.B = S.B: 1/\max(|\pi_B R|, |\pi_B S|)$
  - $S.C = T.C: 1/\max(|\pi_C S|, |\pi_C T|)$
  - $|Q| \approx (|R| \cdot |S| \cdot |T|)/\left(\max(|\pi_B R|, |\pi_B S|) \cdot \max(|\pi_C S|, |\pi_C T|)\right)$

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”
- Not covered: better estimation using histograms

Search for the best plan

- Huge search space
- “Bushy” plan example:

  - Just considering different join orders, there are $(2^n - 2)/(n - 1)!$ 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 $R_1 \bowtie \cdots \bowtie R_6$?
  - Significantly fewer, but still lots—$n!$ ($720$ for $n = 6$)

A greedy algorithm

- $S_1, \ldots, S_n$
  - Say selections have been pushed down; i.e., $S_j = \sigma_{p_j} R_i$
- Start with the pair $S_j, S_i$ with the smallest estimated size for $S_j \bowtie S_i$
- 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