Query Optimization

Introduction to Databases
CompSci 316 Fall 2021

Announcements (Tue., Nov. 23)

• Homework 4 due in one week
  • Except Problem X2, which will be due next Thu.
• We are still wrapping up Homework 3 and Project Milestone 3 grading
• Two sample final exams (from 2019) with solutions released on Sakai

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 $\bowtie$ 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: $\sigma_{p_1} (\sigma_{p_2} R) = \sigma_{p_1 \land p_2} R$
- Merge/split $\pi$’s: $\pi_{L_1} (\pi_{L_2} R) = \pi_{L_1 \cup L_2} R$, where $L_1 \subseteq L_2$
- Push down/pull up $\sigma$:
  $$\sigma_{p_1 \land p_2} (R \bowtie_p S) = \sigma_{p_1} (\pi_{L_1} R) \bowtie_p \sigma_{p_2} (\pi_{L_2} S),$$
  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 $\pi$: $\pi_L (\sigma_p R) = \pi_{L \cup L'} (\sigma_{p\prime} (\pi_{L'} R))$, 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

$$\pi_{\text{Group.name}} \sigma_{\text{User.name} = "Bart" \land \text{User.uid} = \text{Member.uid} \land \text{Member.gid} = \text{Group.gid}} \times \text{Member} \bowtie \text{Group} \times \text{User}$$
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

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
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 Process the outer query without the subquery
  (SELECT * FROM Group WHERE name LIKE 'Springfield'),

  Magic AS Collect bindings
    (SELECT DISTINCT gid FROM Supp_Group),

  DS AS Evaluate the subquery with bindings
    ((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;

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:

1. PROJECT (Group.name)
2. MERGE-JOIN (gid)
3. SCAN (Group)
4. SORT (gid)
5. MERGE-JOIN (uid)
6. SCAN (User)
7. FILTER (name = "Bart")
8. SORT (uid)
9. SCAN (Member)

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

• \(|Q| = |R| \cdot \frac{1}{|\pi_A R| \cdot |\pi_B R|} \)
  • Reduce total size by all selectivity factors

Negated and disjunctive predicates

• \( Q: \sigma_{A \neq u \lor B \neq v} R \)
  • \(|Q| = |R| \cdot (1 - \frac{1}{|\pi_{\neg A} R|}) \)
    • Selectivity factor of \(\neg p\) is \((1 - \text{selectivity factor of } p)\)

• \( Q: \sigma_{A \neq u \lor B \neq v} R \)
  • \(|Q| = |R| \cdot (\frac{1}{|\pi_{\neg A} R|} + \frac{1}{|\pi_{\neg B} R|}) \)
    • Not tuples satisfying \((A = u)\) and \((B = v)\) are counted twice

  Consider \( Q': \sigma_{A \neq u, B \neq v} R \)
  • \(|Q'| = |R| \cdot (1 - \frac{1}{|\pi_{\neg A} R|} \cdot (1 - \frac{1}{|\pi_{\neg B} R|}) \)
  • \(|R| = |Q| + |Q'| \)
  • Therefore \(|Q| = |R| \cdot \frac{1}{|\pi_{\neg A} R|} + \frac{1}{|\pi_{\neg B} R|} - \frac{1}{|\pi_{\neg A} R| \cdot |\pi_{\neg B} R|} \)
    (inclusion-exclusion principle)

Range predicates

• \( Q: \sigma_{A \geq v} R \)

  • Not enough information!
    • Just pick, say, \(|Q| = |R| \cdot \frac{1}{\sqrt{3}}\)

  • With more information
    • Largest R.A value: \(\text{high}(R.A)\)
    • Smallest R.A value: \(\text{low}(R.A)\)
    • \(|Q| = |R| \cdot \frac{1}{\sqrt{\text{high}(R.A) - \text{low}(R.A)}}\)

  • 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 $|\pi_R| \leq |\pi_S|$ then $\pi_R \subseteq \pi_S$
  - Certainly not true in general
  - But holds in the common case of foreign key joins

  - $|Q| \approx \frac{|R||S|}{\max(|\pi_R||\pi_S|)}$
  - Intuitively, if $|\pi_R| \leq |\pi_S|$, each $R$ tuple joins with roughly $\frac{|S|}{\max(|\pi_R||\pi_S|)}$ tuples in $S$
  - Selectivity factor of $R.A = S.A$ is $\frac{1}{\max(|\pi_R||\pi_S|)}$

Examples of Two-way Equi-join

- **Q**: $R(A, B) \bowtie S(B, C)$
  - $|R| = 1000, |\pi_R| = 20$
  - $|S| = 2000, |\pi_S| = 50$
  - $|R \bowtie S| = \frac{|R||S|}{\max(|\pi_R||\pi_S|)} = \frac{1000 \times 2000}{50} = 40,000$

- **Q**: $R(A, B, D) \bowtie S(B, D, C)$
  - $|R| = 1000, |\pi_R| = 20, |\pi_D| = 100$
  - $|S| = 2000, |\pi_S| = 50, |\pi_D| = 50$
  - $|R \bowtie S| = \frac{|R||S|}{\max(|\pi_R||\pi_S|)} = \frac{1000 \times 2000}{100 \times 50} = 400$

Multiway equi-join

- **Q**: $R(A, B) \bowtie S(B, C) \bowtie T(C, D)$
- What information is needed for $(R \bowtie S) \bowtie T$?
  - For $(R \bowtie S)$: need $|R|, |S|, |\pi_R|, |\pi_S|$
  - For $(R \bowtie S) \bowtie T$: need $|R \bowtie S|, |T|, |\pi_T|, |\pi_{T'}|, |\pi_{R \bowtie S}|$
- Assumption: preservation of value sets
  - A non-join attribute does not lose values from its set of possible values
  - That is, if $C$ is in $S$ but not $R$, then $\pi_C(R \bowtie S) = \pi_C S$
  - 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 Cartesian products of relations
  \[ |R| \cdot |S| \cdot |T| \]
• Reduce the total size by the selectivity factor of each join predicate
  \[ |Q| = \frac{|R| \cdot |S| \cdot |T|} {\max(|R|R|, |S|S|) \cdot \max(|S|S|, |T|T|)} \]

Example of Multiway Equi-join

• \( Q: R(A, B) \bowtie S(B, C) \bowtie T(C, D) \)
  \[ |R| = 1000, |R|R| = 20 \]
  \[ |S| = 2000, |R|S|S| = 50, |R|S| = 100 \]
  \[ |T| = 5000, |R|T| = 500 \]
• Estimation method 1: \( (R \bowtie S) \bowtie T \)
  \[ |R| \bowtie S| = \frac{|R| \cdot |S|} {\max(|R|R|, |S|S|)} \frac{1000 \times 2000} {50} = 40,000 \]
  \[ |(R \bowtie S) \bowtie T| = \frac{|R| \bowtie S| \cdot |T|} {\max(|R| \bowtie S|, |T|T|)} \frac{40,000 \times 5000} {500} = 400,000 \]
• Estimation method 2: \( R \bowtie (S \bowtie T) \)
  \[ |S| \bowtie T| = \frac{|S| \cdot |T|} {\max(|S|S|, |T|T|)} \frac{2000 \times 5000} {500} = 20,000 \]
  \[ |R \bowtie (S \bowtie T)| = \frac{|R| \bowtie (S \bowtie T)|} {\max(|R| \bowtie (S \bowtie T)|, |T|T|)} \frac{1000 \times 2000} {50} = 400,000 \]

Cost estimation: summary

• Lots of assumptions and very rough estimation
• Estimation for projection, duplicate elimination, union, difference, aggregation (with grouping)
• 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...

• Histograms with heavy hitters, e.g.:
  • \[ |\sigma_{\text{sub}} R| = \frac{550 \times 5}{10} + 100 \]
  • \[ |\delta \bowtie S| = \frac{250 \times 100}{5} + 150 \times 100 + 100 \times 70 \]

• Machine learning tools
  • Learn from feedback from queries (even without accessing data directly)
  • E.g., given observations from past query executions:
    \[ |\sigma_{A \bowtie R} = 415|, |\sigma_{B \bowtie R} = 115|, |\sigma_{A \bowtie B \bowtie R} = 200|, |\sigma_{A \bowtie B \bowtie C \bowtie R} = 100|, ... \text{what is } |\sigma_{A \bowtie B \bowtie C \bowtie D \bowtie R}|? \]

Search strategy

[Image: cornMaze.jpg]

Search space

• Huge!
  • “Bushy” plan example:

  ![Diagram of search space]

  • Just considering different join orders, there are \( \frac{(2^n-2)}{(n-1)!} \) 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 $R_1 \bowtie \cdots \bowtie R_n$?
    - 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., $S_i = \sigma_{P_i}(R_i)$
  - 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

Example of a Greedy algorithm

- $Q: R(A, B) \bowtie S(B, C) \bowtie T(C, D) \bowtie U(D, A)$
  - $|R| = |S| = |T| = |U| = 1000$
  - $|\pi_A R| = 100$, $|\pi_B R| = 200$
  - $|\pi_C S| = 100$, $|\pi_D S| = 500$
  - $|\pi_D T| = 20$, $|\pi_E T| = 50$
  - $|\pi_U U| = 1000$, $|\pi_D U| = 50$

<table>
<thead>
<tr>
<th></th>
<th>R x S</th>
<th>R x T</th>
<th>R x U</th>
<th>S x T</th>
<th>S x U</th>
<th>T x U</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>5000</td>
<td>12000</td>
<td>10000</td>
<td>2000</td>
<td>10000</td>
<td>2000</td>
</tr>
</tbody>
</table>

- Step 1: choose $T, U$
- Step 2: choose $S$
- Step 3: choose $R$
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)

\( \text{Well, not quite...} \)

Example of A DP algorithm

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

- \( |R| = |S| = |T| = |U| = 1000 \)
- \( |\pi_R R| = 100, |\pi_S S| = 200 \)
- \( |\pi_T T| = 100, |\pi_C C| = 500 \)
- \( |\pi_D D| = 50, |\pi_U U| = 1000 \)

• Step 1:

<table>
<thead>
<tr>
<th>( {R,S} )</th>
<th>( {R,T} )</th>
<th>( {R,U} )</th>
<th>( {S,T} )</th>
<th>( {S,U} )</th>
<th>( {T,U} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( 5000 )</td>
<td>( 1000 )</td>
<td>( 10000 )</td>
<td>( 10000 )</td>
<td>( 10000 )</td>
<td>( 10000 )</td>
</tr>
<tr>
<td>( R \bowtie S )</td>
<td>( R \bowtie T )</td>
<td>( R \bowtie U )</td>
<td>( S \bowtie T )</td>
<td>( S \bowtie U )</td>
<td>( T \bowtie U )</td>
</tr>
</tbody>
</table>

Example of A DP algorithm

• Step 2:

| \( \{R,S,T\} \) | \( \{R,S\} \) | \( \{R,T\} \) | \( \{R,U\} \) | \( \{S,T\} \) | \( \{S,U\} \) | \( \{T,U\} \) |
|-----|-----|-----|-----|-----|-----|
| \( 10000 \) | \( 50000 \) | \( 10000 \) | \( 20000 \) | \( 5000 \) | \( 10000 \) | \( 10000 \) |
| \( S \bowtie T \bowtie R \) | \( R \bowtie S \bowtie T \bowtie U \) | \( \{S \bowtie T \} \bowtie R \bowtie S \bowtie T \bowtie U \) |

• \( \{R,S,T\} \bowtie U = (((S \bowtie T) \bowtie R) \bowtie U) \)
• \( (R,S,U) \bowtie T = (((R \bowtie S) \bowtie U) \bowtie T) \)
• \( (R,T,U) \bowtie S = (((T \bowtie U) \bowtie R) \bowtie S) \)
• \( (S,T,U) \bowtie R = (((S \bowtie T) \bowtie U) \bowtie R) \)
• \( (R,S) \bowtie T,U = (R \bowtie S) \bowtie (T \bowtie U) \)
• …

Exhausted search over all left-deep trees

Exhausted search over all non-left-deep trees
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, duplication 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
Beyond...

* No pairwise join ordering is good!
  - Example: $R(A, B) \bowtie S(B, C) \bowtie T(C, D)$
  - $(R \bowtie S) \bowtie T$: $|R \bowtie S| = 25 + 5 = 30$
  - $R \bowtie (S \bowtie T)$: $|S \bowtie T| = 25 + 5 = 30$
  - Data skew!

* A hybrid approach of multiple orderings
  - $R \bowtie (S \bowtie T)$: $|S \bowtie T| = 5$
  - $(R \bowtie S) \bowtie T$: $|R \bowtie S| = 5$