Query Optimization

Introduction to Database Management
CS348 Fall 2022
Overview

• Many different ways of processing the same query
  • Scan? Sort? Hash? Use an index?
  • All have different performance characteristics and/or make different assumptions about data

• Best choice depends on the situation
  • Implement all alternatives
  • Let the query optimizer choose at run-time (this lecture)

As some materials (sorting/hashing-based algorithms) are made optional in this term, some part of the edited video may not be smooth.
Outline

• System view of query processing
  • Logical plan and physical plan

• Cost calculation of the physical plan
  • Cardinality estimation

• Search space and search strategy
  • Transformation rules
A query’s trip through the DBMS

SQL query

Parser

Parse tree

Validator

Logical plan

Optimizer

Physical plan

Executor

Result

SELECT name, uid
FROM Member, Group
WHERE Member.gid = Group.gid;
Parsing and validation

• **Parser**: SQL → parse tree
  • Detect and reject **syntax** errors

• **Validator**: parse tree → logical plan
  • Detect and reject **semantic** errors
    • Nonexistent tables/views/columns?
    • Insufficient access privileges?
    • Type mismatches?
      • Examples: AVG(name), name + pop, User UNION Member

• Also
  • Expand *
    • Expand view definitions

• Information required for semantic checking is found in **system catalog** (which contains all schema information)
Logical plan

• Nodes are logical operators (often relational algebra operators)
• There are many equivalent logical plans

An equivalent plan:
Physical (execution) plan

- A complex query may involve multiple tables and various query processing algorithms
  - E.g., table scan, index nested-loop join, sort-merge join, hash-based duplicate elimination... (Lecture 13)

- A **physical plan** for a query tells the DBMS query processor how to execute the query
  - A tree of **physical plan operators**
  - Each operator implements a query processing algorithm
  - Each operator accepts a number of input tables/streams and produces a single output table/stream
Examples of physical plans

SELECT Group.name
FROM User, Member, Group
WHERE User.name = 'Bart'
AND User.uid = Member.uid AND Member.gid = Group.gid;

• Many physical plans for a single query
  • Equivalent results, but different costs and assumptions!

 DBMS query optimizer picks the “best” possible physical plan
How to pick the “best” physical plan?

• One logical plan → “best” physical plan

• Questions
  • How to estimate costs
  • How to enumerate possible plans
  • 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
Cost estimation

Physical plan example:

We have: cost estimation for each operator

Example: \( \text{INDEX-NESTED-LOOP-JOIN}(uid) \) takes \( O(B(R) + |R| \cdot \text{index lookup}) \)

We need: size of intermediate results
Cardinality estimation
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 \( R \cdot A \) values

• \(|Q| \approx \frac{|R|}{|\pi_A R|}\)
  • Selectivity factor of \((A = v)\) is \( \frac{1}{|\pi_A R|} \)
Example

Physical plan example:

- \(|\text{User}|=1000, |\pi_{name}(\text{User})| = 50 \Rightarrow |\sigma_{name="Bart"}(\text{User})| = ?\)
- Assumptions:
  - Values of name are uniformly distributed in User
  - Values of \(v\) in \(\sigma_{name="Bart"}(\text{User})\) are uniformly distributed over all User.name values
- \(|\sigma_{name="Bart"}(\text{User})| = \frac{1000}{50} = 20\)
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: $\text{high}(R.A)$
  • Smallest R.A value: $\text{low}(R.A)$
  • $|Q| \approx |R| \cdot \frac{\text{high}(R.A) - v}{\text{high}(R.A) - \text{low}(R.A)}$

• 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!
Example

- Database:
  - User(uid, name, age, pop), Member(gid, uid, date), Group(gid, gname)
  - |User|=1000 rows, |Group|=100 rows, |Member|=50000 rows
  - $\pi_{\text{name}}(\text{User})| = 50$, $\pi_{\text{pop}}(\text{User}) = \{1,2,3,4,5\}$
  - $|\pi_{\text{uid}}(\text{Member})| = 900$

- Estimate size $|\text{User} \bowtie \text{Member}| = ?$
Two-way equi-join

- $Q$: $R(A, B) \Join S(A, C)$

- 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 $\frac{1}{\max(|\pi_A R|, |\pi_A S|)}$
Example

• Database:
  • User(uid, name, age, pop), Member(gid, uid, date), Group(gid, gname)
  • |User|=1000 rows, |Group|=100 rows, |Member|=50000 rows
  • |\pi_{name}(User)| = 50, \pi_{pop}(User) = \{1,2,3,4,5\}
  • |\pi_{uid}(Member)| = 500

• Estimate size |User \bowtie Member| =?
  • |\pi_{uid}(User)| = 1000
  • |\pi_{uid}(Member)| = 500
  • 1000*50000/max(500,1000)=50000
Other estimations

- 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”
    
    ```sql
    SELECT * FROM User WHERE pop > 0.9;
    SELECT * FROM User WHERE pop > 0.9 AND pop > 0.9;
    ```

- Not covered: better estimation using histograms
Case Study

Physical plan example:

- System requirements:
  - Each disk/memory block can hold up to 10 rows (from any table);
  - All tables are stored compactly on disk (10 rows per block);
  - 8 memory blocks are available for query processing: \( M = 8 \)

- Database:
  - User(\( uid, age, pop \)), Member(\( gid, uid, date \)), Group(\( gid, gname \))
  - \(|\text{User}| = 1000\) rows, \(|\text{Group}| = 100\) rows, \(|\text{Member}| = 50000\) rows
  - \# of blocks: \( B(\text{User}) = 1000/10 = 100 \); \( B(\text{Group}) = 100/10 = 10 \); \( B(\text{Member}) = 50000/10 = 5k \)
Case Study

Physical plan example:

- \(|\text{User}| = 1000, |\pi_{name}(\text{User})| = 50 \Rightarrow |\sigma_{name="Bart"}(\text{User})| = \frac{1000}{50} = 20\) records
- INDEX-SCAN on User
  - IO COST: index lookup (4 IOs, depending on the height of the tree)
Case Study

Physical plan example:

- \(|User|=1000, |\pi_{name}(User)| = 50 \Rightarrow |\sigma_{name=\text{"Bart"}}(User)| = \frac{1000}{50} = 20\) records
- INDEX-SCAN on User
  - IO COST: index lookup (4 IOs, depending on the height of the index tree)
- JOIN: For each record with name = “Bart”, probe the index on Member\((uid)\)
  - IO cost: \(B(R) + |R| \cdot \text{(index lookup)}\)
  - 20 rows are not clustered \(\Rightarrow\) at worst case, 20 blocks of data to be retrieved
  - \(20 + 20 \times (4 \text{ IOs for index lookup})\)
Case Study

Physical plan example:

- Given $|\pi_{uid}(\sigma_{name="Bart"}User)| = 20$, $|\pi_{uid}(Member)| = 500$
- $|JOIN(uid)| \approx \frac{|R| \cdot |S|}{\max(|\pi_A R|, |\pi_A S|)} = \frac{20 \cdot 50k}{\max(20, 500)} = \frac{1000k}{500} = 2k$
- Exercise: what is the IO cost for the next INDEX-NESTED-LOOP-JOIN(gid)?

2k rows ~ B(input) = 200 blocks
Outline

• System view of query processing
  • Logical plan and physical plan

• Cost calculation of the physical plan
  • Cardinality estimation

• Search space and search strategy
  • Transformation rules
  • Heuristic approach
Search space is huge

• Characterized by “equivalent” logical query plans
  • select E.Ename, W. Resp
  from Employee E, Projects P, Works W
  where E.ENo = W.Eno and W.Pno=P.Pno and W.Dur > 37
This gets complicated very quickly

• Each logical plan can have multiple physical plans

• Do we need to exam all the logical plans?
  • No. We can use apply heuristic transformation rules to find a cheaper logical plan
Transformation rules (a sample)

• Convert $\sigma_p \times$ to/from $\bowtie_p$: $\sigma_p (R \times S) = R \bowtie_p S$
  • Example: $\sigma_{\text{User.uid} = \text{Member.uid}} (\text{User} \times \text{Member}) = \text{User} \bowtie \text{Member}$

• Merge/split $\sigma$’s: $\sigma_{p_1} (\sigma_{p_2} R) = \sigma_{p_1 \land p_2} R$
  • Example: $\sigma_{\text{age} > 20} (\sigma_{\text{pop} = 0.8} \text{User}) = \sigma_{\text{age} > 20 \land \text{pop} = 0.8} \text{User}$

• Merge/split $\pi$’s: $\pi_{L_1} (\pi_{L_2} R) = \pi_{L_1} R$, where $L_1 \subseteq L_2$
  • Example: $\pi_{\text{age}} (\pi_{\text{age, pop}} \text{User}) = \pi_{\text{age}} \text{User}$
Transformation rules (a sample)

- Push down/pull up $\sigma$:
  $\sigma_{p \land p_r \land p_s}(R \bowtie_{p'} S) = (\sigma_{p_r} R) \bowtie_{p \land p'} (\sigma_{p_s} S)$, where
  - $p_r$ is a predicate involving only $R$ columns
  - $p_s$ is a predicate involving only $S$ columns
  - $p$ and $p'$ are predicates involving both $R$ and $S$ columns
  - Example:

  $\sigma_{U1.name=U2.name \land U1.pop>0.8 \land U2.pop>0.8}(\rho_{U1}User \bowtie_{U1.uid\neq U2.uid} \rho_{U2}User)$

  $= \sigma_{pop>0.8}(\rho_{U1}User) \bowtie_{U1.uid\neq U2.uid, U1.name=U2.name} (\sigma_{pop>0.8}(\rho_{U2}User))$
Transformation rules (a sample)

• 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 $L$
  • Example:
    $\pi_{age}(\sigma_{pop>0.8 User}) = \pi_{age}(\sigma_{pop>0.8}(\pi_{age, pop User}))$

• Many more (seemingly trivial) equivalences…
  • Can be systematically used to transform a plan to new ones
Relational query rewrite example

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

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? 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)
Search strategy

• **Heuristics-based optimization**
  • Apply heuristics to rewrite “logical plans” into cheaper ones

• **Cost-based optimization**
  • Need statistics to estimate sizes of intermediate results to find the best “physical plan”

→ Course CS448 “Database Systems Implementation”
Summary

• System view of query processing
  • Logical plan and physical plan

• Heuristics-based optimization
  • Apply heuristics to rewrite “logical plans” into cheaper ones

• Cost-based optimization
  • Need statistics to estimate sizes of intermediate results to find the best “physical plan”