Lecture 13: Query Optimization

15-445/645 Database Systems (Fall 2017)
Carnegie Mellon University
Prof. Andy Pavlo

Overview

1. SQL is declarative, the user tells the DBMS what answer they want, not how to get the answer
2. There can be massive differences in performance based on plans
3. IBM System R had the first implementation of a query optimizer
4. A lot of the concepts from System R's optimizer are still being used today
5. There are two ways to decide optimizations
   (a) Heuristics/Rules: Rewrite the query to remove inefficiencies. Does not require a cost model
   (b) Cost-based Search: Use a cost model to evaluate multiple equivalent plans and pick the one with the smallest cost

Heuristics and Rules

1. Two relational algebra expressions are equivalent if they generate the same set of tuples
2. The DBMS can identify better query plans without a cost model
3. This is often called **query rewriting**
4. Examples of query rewriting
   (a) Predicate pushdown: Perform predicate filtering before join to reduce size of join)
   (b) Projections: Perform projections early to create smaller tuples and reduce intermediate results. You can project out all attributes except the ones requested or required (e.g. join attributes)
   (c) Join Orderings: For an n-way join, there is Catalan number (approx 4^n) number of ways to do the join

Cost Estimation

1. How long will the query take? You can try to measure:
   (a) CPU: Small cost; tough to estimate

(b) Disk: Number of block transfer

(c) Memory: Amount of DRAM used

(d) Network: Number of messages

2. To accomplish this, the DBMS stores internal statistics about tables, attributes, and indexes in its internal catalog

3. Different systems update the statistics at different times

4. Commercial DBMS have way more robust and accurate statistics compared to the open source systems

Derivable Statistics

1. For a relation R, the DBMS stores the number of tuples (N_R) and distinct values per attribute ($V(A, R)$)

2. The selection cardinality ($SC(A, R)$) is the average number of records with a value for an attribute A given $N_R/V(A, R)$

Complex Predicates

1. The **selectivity** (sel) of a predicate P is the fraction of tuples that qualify

$$sel(A = constant) = SC(P)/V(A, R)$$

2. For a range query, we can use: $sel(A >= a) = (A_{max} - a/(A_{max} - A_{min}))$

3. For negations: $sel(notP) = 1 - sel(P)$

4. These are estimates and thus can sometimes be inaccurate

5. Observation: The selectivity is the probability that a tuple will satisfy the predicate

6. Thus, assuming predicates are independent, then $sel(P1P2) = sel(P1) * sel(P2)$

Join Estimation

1. Given a join of R and S, the estimated size of a join on non-key attribute A is approx

$$estSize \approx N_R * N_S/max(V(A, R), V(A, S))$$

Cost Estimation Reality

1. We assumed values were uniformly distributed
2. In Reality, values are not uniformly distributed, and thus maintaining a histogram is expensive
3. Thus, we can put values into buckets to reduce the size of the histograms. However, this can lead to inaccuracies as frequent values will sway the count of infrequent values
4. To counteract this, we can size the buckets such that their spread is the same. They each hold a similar amount of values

Sampling

1. Modern DBMSs also employ **sampling** to estimate predicate selectivities
2. Sampling: Randomly select and maintain a subset of tuples from a table and estimate the selectivity of the predicate by applying the predicate to the small sample

Query Optimization

1. **Overview**
   (a) Bring query in internal form into canonical form
   (b) Generate alternative plans
   (c) Generate costs for each plan
   (d) Select plan with smallest cost
2. It's important to pick the best access method (i.e sequential scan, binary search, index scan). Simple heuristics are usually good enough for this
3. Query planning for OLTP queries is easy because they are **sargable**
   (a) **S**earch **Arg**ument **Able**
   (b) It is usually just picking the best index
   (c) Joins are almost always on foreign key relationships with small cardinality
   (d) Can be implemented with simple heuristics
4. For multiple relation query planning, the number of alternative plans grows rapidly as number of joins increases
   (a) In System R, the system only considers left-deep join trees
   (b) Left-deep joins allow you to pipeline data, and only need to maintain a single join table in memory
5. Candidate plans algorithm

   (a) Step 1: Enumerate the orderings (Left deep tree 1, left deep tree 2, etc)
   (b) Step 2: Enumerate the plans for each operator (Hash, sort merge, etc)
   (c) Step 3: Enumerate the access paths for each table (Index 1, Index 2, Seq scan, etc)
   (d) Step 4: Build a search graph and walk through to find the lowest cost path

Nested Sub-Queries

1. The DBMS treats nested sub-queries in the where clause as functions that take parameters and return a single value or set of values
2. Two Approaches
   (a) Rewrite to decorrelate and/or flatten queries
   (b) Decompose nested query and store result in subtable

Conclusion

1. Filter as early as possible to reduce amount of data
2. Selectivity estimations using uniformity, independence, histograms, and join selectivity
3. Dynamic programing for join orderings
4. Rewrite nested queries
5. Query optimization is really hard