An Overview of Query Optimization

Chapter 11

Query Evaluation

- **Problem**: An SQL query is declarative – does not specify a query execution plan.
- A relational algebra expression is procedural – there is an associated query execution plan.
- **Solution**: Convert SQL query to an equivalent relational algebra and evaluate it using the associated query execution plan.
  - *But which equivalent expression is best?*

Naive Conversion

SELECT DISTINCT TargetList FROM R1, R2, ..., RN WHERE Condition

is equivalent to

\[ \pi_{\text{TargetList}}(\sigma_{\text{Condition}}(R1 \times R2 \times \ldots \times RN)) \]

but this may imply a very inefficient query execution plan.

**Example**: \( \pi_{\text{Name}}(\sigma_{\text{Id=ProfId \& CrsCode=CS532}}(\text{Professor} \times \text{Teaching})) \)

- Result can be < 100 bytes
- But if each relation is 50K then we end up computing an intermediate result \( \text{Professor} \times \text{Teaching} \) of size 500M before shrinking it down to just a few bytes.

**Problem**: Find an equivalent relational algebra expression that can be evaluated “efficiently”.

Query Processing Architecture

- **Query Optimizer**
  - Uses heuristic algorithms to evaluate relational algebra expressions. This involves:
    - estimating the cost of a relational algebra expression
    - transforming one relational algebra expression to an equivalent one
    - choosing access paths for evaluating the subexpressions
  - Query optimizers do not “optimize” – just try to find “reasonably good” evaluation strategies

Equivalence Preserving Transformations

- To transform a relational expression into another equivalent expression we need transformation rules that preserve equivalence
- Each transformation rule
  - Is provably correct (i.e., does preserve equivalence)
  - Has a heuristic associated with it
Selection and Projection Rules

- Break complex selection into simpler ones:
  \[ \sigma_{\text{Cond1} \land \text{Cond2}}(R) = \sigma_{\text{Cond1}}(\sigma_{\text{Cond2}}(R)) \]
- Break projection into stages:
  \[ \pi_{\text{attr}}(R) = \pi_{\text{attr}'}(\pi_{\text{attr}}(R)), \text{ if attr} \subseteq \text{attr}' \]
- Commute projection and selection:
  \[ \pi_{\text{attr}}(\sigma_{\text{Cond}}(R)) = \sigma_{\text{Cond}}(\pi_{\text{attr}}(R)), \text{ if attr} \subseteq \text{all attributes in Cond} \]

Pushing Selections and Projections

- \[ \sigma_{\text{Cond}}(R \times S) \equiv R \land \sigma_{\text{Cond}} S \]
  - Cond relates attributes of both R and S
  - Reduces size of intermediate relation since rows can be discarded sooner
- \[ \sigma_{\text{Cond}}(R \times S) \equiv \sigma_{\text{Cond}}(R) \times S \]
  - Cond involves only the attributes of R
  - Reduces size of intermediate relation since rows of R are discarded sooner
- \[ \pi_{\text{attr}}(R \times S) \equiv \pi_{\text{attr}'}(\pi_{\text{attr}}(R) \times S), \]
  \[ \text{if attributes}(R) \supseteq \text{attr}' \equiv \text{attr} \cap \text{attributes}(R) \]
  - Reduces the size of an operand of product

Equivalence Example

\[ \sigma_{C1 \land C2 \land C3}(R \times S) \equiv \sigma_{C1}(\sigma_{C2}(\sigma_{C3}(R \times S))) \]
\[ \equiv \sigma_{C1}(\sigma_{C2}(R \times \sigma_{C3}(S))) \]
\[ \equiv \sigma_{C2}(\sigma_{C1}(\sigma_{C3}(S))) \]
assuming \( C2 \) involves only attributes of \( R \), \( C3 \) involves only attributes of \( S \), and \( C1 \) relates attributes of \( R \) and \( S \)

Cost - Example 1

\[ \text{SELECT P.Name} \]
\[ \text{FROM Professor P, Teaching T} \]
\[ \text{WHERE P.Id = T.ProfId} \]
\[ \text{AND P.DeptId = 'CS' AND T.Semester = 'F1994'} \]
\[ \pi_{\text{Name}}(\sigma_{\text{DeptId} = 'CS' \land \text{Semester} = 'F1994'}(\text{Professor} \bowtie \text{Teaching})) \]
\[ \text{Master query execution plan (noting projected)} \]
\[ \text{Professor} \bowtie \text{Teaching} \]

Metadata on Tables (in system catalogue)

- Professor (Id, Name, DeptId)
  - size: 200 pages, 1000 rows, 50 departments
  - indexes: clustered, 2-level B+tree on DeptId, hash on Id
- Teaching (ProfId, CrsCode, Semester)
  - size: 1000 pages, 10,000 rows, 4 semesters
  - indexes: clustered, 2-level B+tree on Semester; hash on ProfId
- Definition: Weight of an attribute — average number of rows that have a particular value
  - weight of Id = 1 (it is a key)
  - weight of ProfId = 10 (10,000 classes/1000 professors)
Estimating Cost - Example 1

- Join - block-nested loops with 52 page buffer (50 pages - input for Professor, 1 page - input for Teaching, 1 - output page
  - Scanning Professor (outer loop): 200 page transfers, (4 iterations, 50 transfers each)
  - Finding matching rows in Teaching (inner loop): 1000 page transfers for each iteration of outer loop
    - 250 professors in each 50 page chunk * 10 matching Teaching tuples per professor = 2500 tuple fetches = 2500 page transfers for Teaching (Why?)
    - By sorting the record Ids of these tuples we can get away with only 1000 page transfers (Why?)
  - total cost = 200+4*1000 = 4200 page transfers

Estimating Cost - Example 1 (cont’d)

- Selection and projection – scan rows of intermediate file, discard those that don’t satisfy selection, project on those that do, write result when output buffer is full.
- Complete algorithm:
  - do join, write result to intermediate file on disk
  - read intermediate file, do select/project, write final result
  - Problem: unnecessary I/O

Pipelining

- Solution: use pipelining:
  - join and select/project act as coroutines, operate as producer/consumer sharing a buffer in main memory.
    - When join fills buffer, select/project filters it and outputs result
    - Process is repeated until select/project has processed last output from join
  - Performing select/project adds no additional cost

Cost Example 2

\[
\text{SELECT Name FROM Professor P, Teaching T WHERE P.Id = T.ProfId AND P.DeptId = 'CS' AND T.Semester = 'F1994'}
\]

\[
\sigma_{\text{DeptId='CS'}}(\text{Professor}) \times \text{DeptId='CS'}(\text{T})
\]

- Compute \(\sigma_{\text{DeptId='CS'}}(\text{Professor})\) (to reduce size of one join table) using clustered, 2-level B+ tree on DeptId.
  - 50 departments and 1000 professors; hence weight of DeptId is 20 (roughly 20 CS professors). These rows are in ~4 consecutive pages in Professor.
  - Cost = 4 (to get rows) + 2 (to search index) = 6
  - keep resulting 4 pages in memory and pipe to next step

Cost Example 2 -- selection

- 50 departments and 1000 professors; hence weight of DeptId is 20 (roughly 20 CS professors). These rows are in ~4 consecutive pages in Professor.
  - Cost = 4 (to get rows) + 2 (to search index) = 6
  - keep resulting 4 pages in memory and pipe to next step
Cost Example 2 -- join

- Index-nested loops join using hash index on ProfId of Teaching and looping on the selected professors (computed on previous slide)
  - Since selection on Semester was not pushed, hash index on ProfId of Teaching can be used
  - *Note*: if selection on Semester were pushed, the index on ProfId would have been lost – an advantage of *not* using a fully pushed query execution plan

Cost Example 2 -- join (cont’d)

- Each professor matches ~10 Teaching rows. Since 20 CS professors, hence 200 teaching records.
- All index entries for a particular ProfId are in same bucket.
  - Assume ~1.2 I/Os to get a bucket.
  - Cost = 1.2 x 20 (to fetch index entries for 20 CS professors) + 200 (to fetch Teaching rows, since hash index is *unclustered*) = 224

Estimating Output Size

- It is important to estimate the size of the output of a relational expression – size serves as input to the next stage and affects the choice of how the next stage will be evaluated.
- Size estimation uses the following measures on a particular instance of R:
  - Tuples(R): number of tuples
  - Blocks(R): number of blocks
  - Values(R.A): number of distinct values of A
  - MaxVal(R.A): maximum value of A
  - MinVal(R.A): minimum value of A

Estimation of Reduction Factor

- Assume that reduction factors due to target list and query condition are independent
- Thus:
  \[
  reduction(Query) = reduction(TargetList) \times reduction(Condition)
  \]
Reduction Due to Simple Condition

- \( \text{reduction}(R.A = \text{val}) = \frac{1}{\text{Values}(R.A)} \)
- \( \text{reduction}(R.A = R_j.B) = \frac{1}{\text{max}(\text{Values}(R.A), \text{Values}(R_j.B))} \)

Assume that values are uniformly distributed, \( \text{Tuples}(R_i) < \text{Tuples}(R_j) \), and every row of \( R_i \) matches a row of \( R_j \). Then the number of tuples that satisfy Condition is:

\[
\frac{\text{Tuples}(R_i)}{\text{Tuples}(R_j)} \times \frac{\text{Values}(R_i.A)}{\text{Values}(R_j.A)} \times \frac{\text{Tuples}(R_j)}{\text{Values}(R_j.B)}
\]

- \( \text{reduction}(R.A > \text{val}) = \frac{\text{MaxVal}(R.A) - \text{val}}{\text{MaxVal}(R.A) - \text{MinVal}(R.A)} \)

Reduction Due to Complex Condition

- \( \text{reduction}(\text{Cond}_1 \text{ AND } \text{Cond}_2) = \text{reduction}(\text{Cond}_1) \times \text{reduction}(\text{Cond}_2) \)
- \( \text{reduction}(\text{Cond}_1 \text{ OR } \text{Cond}_2) = \min(1, \text{reduction}(\text{Cond}_1) + \text{reduction}(\text{Cond}_2)) \)

Estimating Weight of Attribute

\[
\text{weight}(R.A) = \frac{\text{Tuples}(R) \times \text{reduction}(R.A = \text{value})}{\text{number-of-attributes}(R.A)}
\]

Choosing Query Execution Plan

- Step 1: Choose a logical plan
- Step 2: Reduce search space
- Step 3: Use a heuristic search to further reduce complexity

Step 1: Choosing a Logical Plan

- Involves choosing a query tree, which indicates the order in which algebraic operations are applied
- \textit{Heuristic:} Pushed trees are good, but sometimes “nearly fully pushed” trees are better due to indexing (as we saw in the example)
- So: Take the initial “master plan” tree and produce a \textit{fully pushed} tree plus several \textit{nearly fully pushed} trees.
Step 2: Reduce Search Space

- Deal with associativity of binary operators (join, union, …)

A B C D
A B C D
A B C D

Logical query execution plan

Equivalent query tree

Step 2: Dealing With Associativity

- Too many trees to evaluate: settle on a particular shape: left-deep tree.
  - USED because it allows pipelining

P₁
P₂
P₃

A B
X

Y D

Step 2: Dealing with Associativity

- Consider the alternative: if we use the association ((A × B) × (C × D))

P₁
P₂
P₃

A B
X

C D

Y

Each row of X must be processed against all of Y. Hence all of Y (can be very large) must be stored in P₃, or P₂ has to recompute it several times.

Step 3: Heuristic Search

- The choice of left-deep trees still leaves open too many options (N! permutations):
  - (((A × B) × C) × D),
  - (((C × A) × D) × B), …
  - A heuristic (often dynamic programming based) algorithm is used to get a ‘good’ plan

Step 2 (cont’d)

- Two issues:
  - Choose a particular shape of a tree (like in the previous slide)
    - Equals the number of ways to parenthesize N-way join – grows very rapidly
  - Choose a particular permutation of the leaves
    - E.g., 4! permutations of the leaves A, B, C, D

Step 3: Dynamic Programming Algorithm

- Just an idea – see book for details
- To compute a join of E₁, E₂, …, Eₙ in a left-deep manner:
  - Start with 1-relation expressions (can involve ϕ, π)
  - Choose the best and “nearly best” plans for each (a plan is considered nearly best if its output has some “interesting” form, e.g., is sorted)
  - Combine these 1-relation plans into 2-relation expressions. Retain only the best and nearly best 2-relation plans
  - Do same for 3-relation expressions, etc.
Index-Only Queries

- A B* tree index with search key attributes $A_1, A_2, \ldots, A_n$ has stored in it the values of these attributes for each row in the table.
  - Queries involving a prefix of the attribute list $A_1, A_2, \ldots, A_n$ can be satisfied using *only the index* – no access to the actual table is required.

- **Example:** Transcript has a clustered B* tree index on StudId. A frequently asked query is one that requests all grades for a given CrsCode.
  - **Problem:** Already have a clustered index on StudId – cannot create another one (on CrsCode)
  - **Solution:** Create an unclustered index on $(\text{CrsCode, Grade})$
    - Keep in mind, however, the overhead in maintaining extra indices