Database Systems I

Query Processing & Optimization (2)

**Instructor:** Ouldooz Baghban Karimi

CMPT 354 - Summer 2019
Query Processing

From Chapter 15 and 16, The complete book
The job of the parser is to take text written in a language such as SQL and convert it to a parse tree.

Parse tree
  • Atoms
    • No Children
    • Keywords, names, constants, parentheses, operators
  • Syntactic categories
    • Have Children
    • <Query>, <Condition>
Preprocessor

• Obtain Parse Tree for Views
  • If a relation used in the query is actually a virtual view replace it by a parse tree that describes the view
    (we use relational algebra here because it is more succinct than the parse trees)

• Perform Semantic Checking
  • Check relation uses
    • Every relation mentioned is a relation or view in the current schema
  • Check and resolve attribute uses
    • Every attribute used is an attribute of some relation in the current scope
  • Check types
    • Type appropriate to their uses
From Parse Trees to Logical Query Plans

• Logical Query Plan Generator
  • Conversion to Relational Algebra
  • Removing Subqueries From Conditions
  • Improving the Logical Query Plan
  • Grouping Associative & Commutative Operators
Conversion to Relational Algebra

• If a <Query> with a <Condition> has no subqueries
  • We may replace it by a RA expression consisting, from bottom to top
    • The product of all the relations mentioned in the <FromList>, which is the argument of
    • A selection $\sigma_C$, where $C$ is the <Condition> expression in the construct being replaced, which in turn is the argument of
    • A projection $\pi_L$, where $L$ is the list of attributes in the <SelList>
Example (1)

From Chapter 16, The complete book
Removing Subqueries from Conditions

• Two-argument selection
  • A node labeled \( \sigma \), with no parameter
    • **Left child**: the relation \( R \) upon which the selection is being performed
    • **Right child**: an expression for the condition applied to each tuple of \( R \)

A two-argument selection with a condition involving `IN`
**Example (1)**

\[
\begin{align*}
&\text{SELECT <SelList> FROM <FromList> WHERE <Condition>} \\
&\quad \text{movieTitle} \quad \text{StarsIn} \quad \text{starName} \quad \text{<Query>} \\
&\Rightarrow \\
&\text{SELECT <SelList> FROM <FromList> WHERE <Condition>} \\
&\quad \text{name} \quad \text{MovieStar} \quad \text{birthdate LIKE '1960'}
\end{align*}
\]
Example (1)
```
SELECT DISTINCT ml.movieTitle, ml.movieYear
FROM StarsIn ml
WHERE ml.movieYear - 40 <=
    ( SELECT AVG(birthdate) 
        FROM StarsIn m2, MovieStar s 
        WHERE m2.starName = s.name AND 
            ml.movieTitle = m2.movieTitle AND 
            ml.movieYear = m2.movieYear 
    );
```

From Chapter 16, The complete book
Example (2)
Improving the Logical Query Plan

• **Selections** can be **pushed down** the expression tree as far as they can go

• **Projections** can be **pushed down** the tree, or **new projections** can be added

• **Duplicate eliminations** can sometimes be **removed**, or **moved** to a more convenient position in the tree

• Certain **selections** can be **combined with a product below** to turn the pair of operations into an equijoin, which is generally much more efficient to evaluate than are the two operations separately
Example

\[
\delta \\
\pi_{m2.movieTitle, m2.movieYear} \\
\sigma_{m2.movieYear-40 < abd} \\
\gamma_{m2.movieTitle, m2.movieYear, \text{AVG}(s.birthday) \rightarrow abd} \\
m2.starName = s.name \\
\text{StarsIn m2} \quad \text{MovieStar s}
\]
Final Steps

• Grouping Associative/Commutative Operators

  • Replace the natural joins with theta-joins that equate the attributes of the same name
  
  • Must add a projection to eliminate duplicate copies of attributes involved in a natural join that has become a theta-join
  
  • The theta-join conditions must be associative
Final Step in producing the logical query plan:
group the associative and commutative operators
Query Processing

• Query Compiler
  • Parsing & Preprocessing
  • Logical Optimization

✓ Query Execution
  ✓ Physical Query Plan
Size Estimation

• $B(R)$: the number of blocks needed to hold relation $R$

• $T(R)$: is the number of tuples of relation $R$

• $V(R, a)$: is the value count for attribute $a$ of relation $R$, that is, the number of distinct values relation $R$ has in attribute $a$
Size Estimation

• Estimating size of a selection
  • Let $S = \sigma_{A=C} (R)$, where A is an attribute of R and C is a constant.

    An estimate $T(S) = T(R)/V(R,A)$

• Estimating size of a natural join
  • $T(R \bowtie S) = T(R)T(S)/\max (V(R, Y), V(S, Y))$
Size Estimation

• Union
  • As large as the sum of the sizes or as small as the larger of the two arguments
  • One approach: use an average

• Intersection
  • As few as 0 tuples or as many as the smaller of the two arguments
  • One approach: Take the average of the extremes, which is half the smaller

• Difference
  • When we compute $R - S$, the result can have between $T(R)$ and $T(R) - T(S)$ tuples
  • One approach: Average as an estimate: $T(R) - T(S)/2$
Logical Plan to Physical Plan

• Estimating the Cost of Operations
  • Cost-based enumeration
    • An order and grouping for associative-and-commutative operations (e.g. joins, unions, intersections)
    • An algorithm for each operator in the logical plan (e.g. deciding whether a nested-loop join or a hash-join should be used)
    • Additional operators (e.g. scanning, sorting) that are needed for the physical plan but that were not present explicitly in the logical plan
    • The way in which arguments are passed from one operator to the next (e.g. by storing the intermediate result on disk or by using iterators and passing an argument one tuple or one main-memory buffer at a time)

• Estimating Sizes of Intermediate Relations
Enumerating Physical Plans

• Top-down: work down the tree of the logical query plan from the root

• Bottom-up
  • Compute the costs of all possible ways to compute that subexpression
  • Combine them in all possible ways

• Heuristic Selection
• Branch-and-Bound Plan Enumeration
• Hill Climbing
• Dynamic Programming
• Selinger-Style Optimization
Acknowledgements

I have used materials from the following resources in preparation of this course:

• **Database Systems: The Complete Book**
• Database Systems (Kiefer, Bernstein, Lewis)
• Database System Concepts: [https://www.db-book.com](https://www.db-book.com)
• Course offerings
  • CMPT 354 (Jiannan Wang - SFU): [https://sfu-db.github.io/cmpt354/](https://sfu-db.github.io/cmpt354/)
  • W 4111 (Eugene Wu - Columbia): [https://w4111.github.io/](https://w4111.github.io/)
  • CS 186 (Joe Hellerstein - Berkeley): [https://sites.google.com/site/cs186fall17/](https://sites.google.com/site/cs186fall17/)