Database Systems I

Query Processing & Optimization (1)

Instructor: Ouldooz Baghban Karimi

CMPT 354 - Summer 2019
Query Processing

✓ Query Compiler
   ✓ Parsing & Preprocessing
   ✓ Logical Optimization

• Query Execution
  • Physical Query Plan
Query Processing

Query → Parse Tree → Logical Query Plan → Physical Query Plan → Query Execution

• Logical Query Plan
  • Logical Optimization: improvement through algebraic laws

• Physical Query Plan
  • Cost Estimation
  • Algorithm choice
  • Pipelining versus materialization
Query Processing
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>
Parsing Process Rules

- `<Query>` ::= SELECT `<SelList>` FROM `<FromList>` WHERE `<Condition>`
- Symbol ::= means “can be expressed as”

- `<Attribute>`
  Any string of characters that identifies an attribute of the current database schema

- `<Relation>`
  Can be replaced by any string of characters that makes sense as a relation in the current schema

- `<Pattern>`
  Can be replaced by any quoted string that is a legal SQL pattern

- `<SelList>`
  Either a single attribute or an attribute, a comma, and any list of one or more attributes.

- `<FromList>`
  Any comma-separated list of relations

- `<Condition>`
  Represents SQL conditions
Sample Parsing Process Rules

\(<\text{SelList}\> ::= \ <\text{Attribute}\> , \ <\text{SelList}\>
\(<\text{SelList}\> ::= \ <\text{Attribute}\>\\

\(<\text{FromList}\> ::= \ <\text{Relation}\> , \ <\text{FromList}\>
\(<\text{FromList}\> ::= \ <\text{Relation}\>\\

\(<\text{Condition}\> ::= \ <\text{Condition}\> \text{ AND } \ <\text{Condition}\>
\(<\text{Condition}\> ::= \ <\text{Attribute}\> \text{ IN } ( \ <\text{Query}\> )
\(<\text{Condition}\> ::= \ <\text{Attribute}\> = \ <\text{Attribute}\>
\(<\text{Condition}\> ::= \ <\text{Attribute}\> \text{ LIKE } \ <\text{Pattern}\>\)
Example

StarsIn(movieTitle, movieYear, starName)
MovieStar(name, address, gender, birthdate)

Query: find the titles of movies that have at least one star born in 1960

(A)
SELECT movieTitle FROM StarsIn
WHERE starName IN ( 
    SELECT name
    FROM MovieStar
    WHERE birthdate LIKE '%1960'
);

(B)
SELECT movieTitle
FROM StarsIn, MovieStar
WHERE starName = name AND
    birthdate LIKE '%1960';
Example

From Chapter 16, *The complete book*
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
Logical Query Optimization

• Conversion to Relational Algebra
  • Algebraic Laws for Improving Query Plans

• Removing Subqueries From Conditions

• Improving the Logical Query Plan (Rewriting)
Some Trivial Laws

• Any selection on an empty relation is empty

• If $C$ is an always-true condition
  (e.g. $x > 10$ OR $x < 10$ on a relation that forbids $x = NULL$)
  then $\sigma_C(R) = R$

• If $R$ is empty, then $R \cup S = S$

**Note**: $R$ and $S$ are relations and $C$ is a condition
Commutative and Associative Laws

• Order of operator and argument presentation
  • Associative (e.g. arithmetic \((x + y) + z = x + (y + z)\)
  • Commutative (e.g. arithmetic \(x + y = y + x\))

• In Relational Algebra
  • \(R \times S = S \times R\)
  • \((R \times S) \times T = R \times (S \times T)\)
  • \(R \bowtie S = S \bowtie R\)
  • \((R \bowtie S) \bowtie T = R \bowtie (S \bowtie T)\)
  • \(R \cup S = S \cup R\)
  • \((R \cup S) \cup T = R \cup (S \cup T)\)
  • \(R \cap S = S \cap R\)
  • \((R \cap S) \cap T = R \cap (S \cap T)\)
Laws Involving Selection

• Selections reduce the size of relations
  • Move the selections down the tree as far as they will go without changing what the expression does (Push down selection: Major tool for optimizer)
    • Push down laws

• When the condition of a selection is complex
  • Split laws
Split Laws

\[ \sigma_{C_1 \text{ AND } C_2} (R) = \sigma_{C_1} (\sigma_{C_2} (R)) \]

\[ \sigma_{C_1 \text{ OR } C_2} (R) = (\sigma_{C_1} (R)) \cup_S (\sigma_{C_2} (R)) \]

\[ \sigma_{C_1} (\sigma_{C_2} (R)) = \sigma_{C_2} (\sigma_{C_1} (R)) \]

*S says that it is only a set union*
Push Laws

• For a union the selection must be pushed to both arguments
  \[ \sigma_C (R \cup S) = \sigma_C (R) \cup \sigma_C (S) \]

• For a difference the selection must be pushed to the first argument and optionally may be pushed to the second
  \[ \sigma_C (R - S) = \sigma_C (R) - S \]
  \[ \sigma_C (R - S) = \sigma_C (R) - \sigma_C (S) \]

• For the other operators it is only required that the selection be pushed to one argument
  \[ \sigma_C (R \cap S) = \sigma_C (R) \cap S \]
Pushing Selections

Logical query plan constructed from definition of a query and view

Improving the query plan by moving selections down the tree

From Chapter 16, The complete book
Laws Involving Projection

• Introducing a new projection somewhere below an existing projection
  • We may introduce a projection anywhere in an expression tree, as long as it eliminates only attributes that are neither used by an operator above nor are in the result of the entire expression

\[
\pi_L(R \bowtie S) = \pi_L(\pi_M(R) \bowtie \pi_N(S))
\]
\[
\pi_L(R \bowtie_c S) = \pi_L(\pi_M(R) \bowtie_c \pi_N(S))
\]
\[
\pi_L(R \times S) = \pi_L(\pi_M(R) \times \pi_N(S))
\]

*M and N are the join attributes and the input attributes of L that are found among the attributes of R and S, respectively*

• Less useful than pushing selections
Laws About Joins and Products

• Commutative & associative laws

• Some additional laws
  • R \bowtie_C S = \sigma_C (R \times S)
  • R \bowtie S = \pi_L (\sigma_C (R \times S))

C is the condition that equates each pair of attributes from R and S with the same name, and L is a list that includes one attribute from each equated pair and all the other attributes of R and S.
Additional Laws

• Laws Involving Duplicate Elimination
  • Can be pushed through many, but not all operators
  • We can sometimes move it to where it can be eliminated completely

• Laws Involving Grouping and Aggregation
  • Depends on the details of the aggregate: Not many general rules
  • We may project useless attributes prior to applying
I have used materials from the following resources in preparation of this course:

• **Database Systems: The Complete Book**
• Database Systems (Kifer, 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/)