Query Processing and Advanced Queries

Query Processing (2)
Review: Query Processing

SQL query → parse → parse tree → convert → logical query plan → query rewrite → “improved” l.q.p → estimate result sizes → l.q.p. + sizes → consider physical plans

{P1, P2, …..} → estimate costs → {(P1, C1), (P2, C2), …..} → pick best → execute → answer

CMPT 454: Database Systems II – Query Processing (2)
The following grammar describes a simple subset of SQL.

**Queries**

\[ \text{<Query>} ::= \text{SELECT} \text{ <SelList>} \text{ FROM} \text{ <FromList>} \text{ WHERE} \text{ <Condition>} ; \]

**Selection lists**

\[ \text{<SelList>} ::= \text{ <Attribute> , <SelList>} \]
\[ \text{<SelList>} ::= \text{ <Attribute>} \]

**From lists**

\[ \text{<FromList>} ::= \text{ <Relation> , <FromList>} \]
\[ \text{<FromList>} ::= \text{ <Relation>} \]
Grammar for SQL

- Conditions
  - `<Condition>::= <Condition> AND <Condition>`
  - `<Condition>::= <Attribute> IN (<Query>)`
  - `<Condition>::= <Attribute> = <Attribute>`
  - `<Condition>::= <Attribute> LIKE <Pattern>`

- Syntactic categories Relation and Attribute are not defined by grammar rules, but by the database schema.
- Syntactic category Pattern defined as some regular expression.
Example: A SQL Query

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

Goal: find the movies with stars born in 1960

SELECT movieTitle
FROM StarsIn, MovieStar
WHERE starName = name AND birthdate LIKE ‘%1960’
A Parse Tree

```
SELECT <SelList> FROM <FromList> WHERE <Condition>

<Attribute> movieTitle <Relation> StarsIn <FromList>

<Attribute> MovieStar <Relation> <Condition>

<Attribute> AND <Attribute> <Pattern> LIKE '1960'

starName = name

birthdate

CMPT 454: Database Systems II – Query Processing (2)
```
Conversion to Logical Query Plan

- How to convert a parse tree into a logical query plan, i.e. a relational algebra expression?

- Queries with conditions without subqueries are easy:
  - Form **Cartesian product** of all relations in <FromList>.
  - Apply a **selection** $\sigma_c$ where C is given by <Condition>.
  - Finally apply a **projection** $\pi_L$ where L is the list of attributes in <SelList>.

- Queries involving subqueries are more difficult.
  - Remove subqueries from conditions and represent them by a two-argument selection in the logical query plan.
An Algebraic Expression Tree

\[ \pi_{\text{movieTitle}} \]
\[ \sigma_{\text{starName}=\text{name AND birthdate LIKE } '1960'} \]
\[ X \]

\[ \text{StarsIn} \quad \text{MovieStar} \]
Another SQL Query

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

Goal: find the movies with stars born in 1960

SELECT title
FROM StarsIn
WHERE starName IN (SELECT name FROM MovieStar
WHERE birthdate LIKE ‘%1960’);
Another Parse Tree

```
SELECT  <SelList>    FROM    <FromList>     WHERE     <Condition>
     <Attribute>              <Relation>
         title               StarsIn ( )
            IN
                        <Attribute> IN ( )
                           スターName

SELECT  <SelList>    FROM    <FromList>     WHERE     <Condition>
     <Attribute>              <Relation>
         name                MovieStar LIKE <Pattern>
                        birthDate LIKE ‘%1960’
```
Another Algebraic Expression Tree

\[ \pi_{\text{movieTitle}} \sigma \text{StarsIn \atop \langle \text{condition} \rangle} \langle \text{Attribute} \rangle \text{IN} \pi_{\text{name}} \sigma_{\text{starName}} \sigma_{\text{birthdate LIKE ‘%1960’}} \text{MovieStar} \]
Algebraic Laws for Query Plans

Introduction

- Algebraic laws allow us to transform a Relational Algebra (RA) expression into an equivalent one.
- Two RA expressions are equivalent if, for all database instances, they produce the same answer.
- The resulting expression may have a more efficient physical query plan.
- Algebraic laws are used in the query rewrite phase.
Introduction

- **Commutative law:**
  Order of arguments does not matter.
  \[ x + y = y + x \]

- **Associative law:**
  May group two uses of the operator either from the left or the right.
  \[ (x + y) + z = x + (y + z) \]

- Operators that are commutative and associative can be grouped and ordered arbitrarily.
Algebraic Laws for Query Plans

Natural Join, Cartesian Product and Union

\[ R \bowtie S = S \bowtie R \]
\[ (R \bowtie S) \bowtie T = R \bowtie (S \bowtie T) \]
\[ R \times S = S \times R \]
\[ (R \times S) \times T = R \times (S \times T) \]
\[ R \cup S = S \cup R \]
\[ R \cup (S \cup T) = (R \cup S) \cup T \]
Algebraic Laws for Query Plans

Selection

\[ \sigma_{p_1 \land p_2}(R) = \sigma_{p_1} \left[ \sigma_{p_2}(R) \right] \]

\[ \sigma_{p_1 \lor p_2}(R) = \left[ \sigma_{p_1}(R) \right] \cup \left[ \sigma_{p_2}(R) \right] \]

\[ \sigma_{p_1} \left[ \sigma_{p_2}(R) \right] = \sigma_{p_2} \left[ \sigma_{p_1}(R) \right] \]

- Simple conditions \( p_1 \) or \( p_2 \) may be pushed down further than the complex condition.
Bag Union

What about the union of relations with duplicates (bags)?

\[ R = \{a,a,b,b,b,c\} \]
\[ S = \{b,b,c,c,d\} \]
\[ R \cup S = ? \]

Number of occurrences either SUM or MAX of occurrences in the input relations.

SUM: \[ R \cup S = \{a,a,b,b,b,b,b,c,c,c,d\} \]
MAX: \[ R \cup S = \{a,a,b,b,b,c,c,d\} \]
**Selection**

- $\sigma_{p_1 \lor p_2}(R) = \sigma_{p_1}(R) \cup \sigma_{p_2}(R)$
- MAX implementation of union makes rule work.

- $R = \{a,a,b,b,b,c\}$
  - $p_1$ satisfied by $a,b$, $p_2$ satisfied by $b,c$
  - $\sigma_{p_1 \lor p_2}(R) = \{a,a,b,b,b,c\}$
  - $\sigma_{p_1}(R) = \{a,a,b,b,b\}$
  - $\sigma_{p_2}(R) = \{b,b,b,c\}$
  - $\sigma_{p_1}(R) \cup \sigma_{p_2}(R) = \{a,a,b,b,b,c\}$
Algebraic Laws for Query Plans

Selection

- \( \sigma_{p_1 \lor p_2}(R) = \sigma_{p_1}(R) \cup \sigma_{p_2}(R) \)

SUM implementation of union makes more sense.

- Senators (……..)
- Reps (……..)

\[
\begin{array}{ccc}
T1 & = & \pi_{yr, state} \text{ Senators}, \\
T2 & = & \pi_{yr, state} \text{ Reps}
\end{array}
\]

<table>
<thead>
<tr>
<th>T1 Year</th>
<th>T1 State</th>
<th>T2 Year</th>
<th>T2 State</th>
</tr>
</thead>
<tbody>
<tr>
<td>97</td>
<td>CA</td>
<td>99</td>
<td>CA</td>
</tr>
<tr>
<td>99</td>
<td>CA</td>
<td>99</td>
<td>CA</td>
</tr>
<tr>
<td>98</td>
<td>AZ</td>
<td>98</td>
<td>CA</td>
</tr>
</tbody>
</table>

Use SUM implementation, but then some laws do not hold.
Algebraic Laws for Query Plans

Selection and Set Operations

\[ \sigma_p(R \cup S) = \sigma_p(R) \cup \sigma_p(S) \]

\[ \sigma_p(R - S) = \sigma_p(R) - S = \sigma_p(R) - \sigma_p(S) \]
Algebraic Laws for Query Plans

Selection and Join

- $p$: predicate with only $R$ attributes
- $q$: predicate with only $S$ attributes
- $m$: predicate with attributes from $R$ and $S$

$$\sigma_p (R \bowtie S) = [\sigma_p (R)] \bowtie S$$

$$\sigma_q (R \bowtie S) = R \bowtie [\sigma_q (S)]$$
Selection and Join

\[ \sigma_{p \land q} (R \bowtie S) = [\sigma_p (R)] \bowtie [\sigma_q (S)] \]

\[ \sigma_{p \land q \land m} (R \bowtie S) = \]
\[ \sigma_m [(\sigma_p R) \bowtie (\sigma_q S)] \]

\[ \sigma_{p \lor q} (R \bowtie S) = \]
\[ [(\sigma_p R) \bowtie S] \cup [R \bowtie (\sigma_q S)] \]
Algebraic Laws for Query Plans

Projection

- X: set of attributes
- Y: set of attributes
- XY: X U Y

\[ \pi_{xy}(R) = \pi_x[\pi_y(R)] \]

May introduce projection anywhere in an expression tree as long as it eliminates no attributes needed by an operator above and no attributes that are in result
**Algebraic Laws for Query Plans**

*Projection and Selection*

- $X$: subset of $R$ attributes
- $Z$: attributes in predicate $P$ (subset of $R$ attributes)

\[ \pi_x (\sigma_p R) = \pi_x \{\sigma_p [ \pi_x (R) ]\} \]

- Need to keep attributes for the selection and for the result
Algebraic Laws for Query Plans

Projection and Selection

- X: subset of R attributes
- Y: subset of S attributes
- Z: intersection of R, S attributes

\[ \pi_{xy}(R \bowtie S) = \pi_{xy}\{[\pi_{xz}(R)] \bowtie [\pi_{yz}(S)]\} \]
Projection, Selection and Join

\[ \pi_{xy} \{\sigma_p (R \bowtie S)\} = \]

\[ \pi_{xy} \{\sigma_p [\pi_{xz'} (R) \bowtie \pi_{yz'} (S)]\} \]

\[ z' = z \cup \{\text{attributes used in } P\} \]
What Are Good Transformation?

- No transformation is always good
- Usually good: early selections/projections