Relational Algebra

Operators
Expression Trees

What is an "Algebra"
◆ Mathematical system consisting of:
  • Operands --- variables or values from which new values can be constructed.
  • Operators --- symbols denoting procedures that construct new values from given values.

What is Relational Algebra?
◆ An algebra whose operands are relations or variables that represent relations.
◆ Operators are designed to do the most common things that we need to do with relations in a database.
  • The result is an algebra that can be used as a query language for relations.

Roadmap
◆ There is a core relational algebra that has traditionally been thought of as the relational algebra.
◆ But there are several other operators we shall add to the core in order to model better the language SQL --- the principal language used in relational database systems.

Core Relational Algebra
◆ Union, intersection, and difference.
  • Usual set operations, but require both operands have the same relation schema.
◆ Selection: picking certain rows.
◆ Projection: picking certain columns.
◆ Products and joins: compositions of relations.
◆ Renaming of relations and attributes.

Selection
◆ \( R1 := \text{SELECT}_C(R2) \)
  • \( C \) is a condition (as in "if" statements) that refers to attributes of \( R2 \).
  • \( R1 \) is all those tuples of \( R2 \) that satisfy \( C \).
Example

Relation Sells:

<table>
<thead>
<tr>
<th>bar</th>
<th>beer</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Joe's</td>
<td>Bud</td>
<td>2.50</td>
</tr>
<tr>
<td>Joe's</td>
<td>Miller</td>
<td>2.75</td>
</tr>
<tr>
<td>Sue's</td>
<td>Bud</td>
<td>2.50</td>
</tr>
<tr>
<td>Sue's</td>
<td>Miller</td>
<td>3.00</td>
</tr>
</tbody>
</table>

JoelMenu := SELECT_{bar='Joel'}(Sells):  

<table>
<thead>
<tr>
<th>bar</th>
<th>beer</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Joel</td>
<td>Bud</td>
<td>2.50</td>
</tr>
<tr>
<td>Joel</td>
<td>Miller</td>
<td>2.75</td>
</tr>
</tbody>
</table>

Projection

◆ R1 := PROJ_r (R2)  
  • L is a list of attributes from the schema of R2.  
  • R1 is constructed by looking at each tuple of R2, extracting the attributes on list L in the order specified, and creating from those components a tuple for R1.  
  • Eliminate duplicate tuples, if any.

Example

Relation Sells:

<table>
<thead>
<tr>
<th>bar</th>
<th>beer</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Joe's</td>
<td>Bud</td>
<td>2.50</td>
</tr>
<tr>
<td>Joe's</td>
<td>Miller</td>
<td>2.75</td>
</tr>
<tr>
<td>Sue's</td>
<td>Bud</td>
<td>2.50</td>
</tr>
<tr>
<td>Sue's</td>
<td>Miller</td>
<td>3.00</td>
</tr>
</tbody>
</table>

Prices := PROJ_{beer=price}(Sells):

<table>
<thead>
<tr>
<th>beer</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bud</td>
<td>2.50</td>
</tr>
<tr>
<td>Miller</td>
<td>3.00</td>
</tr>
</tbody>
</table>

Product

◆ R3 := R1 * R2  
  • Pair each tuple t1 of R1 with each tuple t2 of R2.  
  • Concatenation t1t2 is a tuple of R3.  
  • Schema of R3 is the attributes of R1 and R2, in order.  
  • But beware attribute A of the same name in R1 and R2: use R1.A and R2.A.

Example: R3 := R1 * R2

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>5</td>
<td>8</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>1</td>
<td>2</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>1</td>
<td>2</td>
<td>9</td>
<td>10</td>
</tr>
<tr>
<td>7</td>
<td>8</td>
<td>3</td>
<td>4</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>9</td>
<td>10</td>
<td>3</td>
<td>4</td>
<td>9</td>
<td>10</td>
</tr>
</tbody>
</table>

Theta-Join

◆ R3 := R1 JOIN_C R2  
  • Take the product R1 * R2.  
  • Then apply SELECT_C to the result.  
  • As for SELECT, C can be any boolean-valued condition.  
  • Historic versions of this operator allowed only A theta B, where theta was =, <, etc.; hence the name "theta-join."
Example

<table>
<thead>
<tr>
<th>Bar</th>
<th>Beer</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Joe's</td>
<td>Bud</td>
<td>2.50</td>
</tr>
<tr>
<td>Joe's</td>
<td>Miller</td>
<td>2.75</td>
</tr>
<tr>
<td>Sue's</td>
<td>Bud</td>
<td>2.50</td>
</tr>
<tr>
<td>Sue's</td>
<td>Coors</td>
<td>3.00</td>
</tr>
</tbody>
</table>

BarInfo := Sells JOIN Sells.bar = Bars.bar

Example

<table>
<thead>
<tr>
<th>Bar</th>
<th>Beer</th>
<th>Name</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Joe's</td>
<td>Bud</td>
<td>Joe's Maple St.</td>
<td>2.50</td>
</tr>
<tr>
<td>Joe's</td>
<td>Miller</td>
<td>Joe's Maple St.</td>
<td>2.75</td>
</tr>
<tr>
<td>Sue's</td>
<td>Bud</td>
<td>Sue's River Rd.</td>
<td>2.50</td>
</tr>
<tr>
<td>Sue's</td>
<td>Coors</td>
<td>Sue's River Rd.</td>
<td>3.00</td>
</tr>
</tbody>
</table>

BarInfo := Sells JOIN Bars
Note Bars.name has become Bars.bar to make the natural join "work."

Example

<table>
<thead>
<tr>
<th>Bar</th>
<th>Beer</th>
<th>Name</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Joe's</td>
<td>Bud</td>
<td>Joe's Maple St.</td>
<td>2.50</td>
</tr>
<tr>
<td>Joe's</td>
<td>Miller</td>
<td>Joe's Maple St.</td>
<td>2.75</td>
</tr>
<tr>
<td>Sue's</td>
<td>Bud</td>
<td>Sue's River Rd.</td>
<td>2.50</td>
</tr>
<tr>
<td>Sue's</td>
<td>Coors</td>
<td>Sue's River Rd.</td>
<td>3.00</td>
</tr>
</tbody>
</table>

Example

<table>
<thead>
<tr>
<th>Bar</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Joe's</td>
<td>2.50</td>
</tr>
<tr>
<td>Sue's</td>
<td>2.50</td>
</tr>
<tr>
<td>Sue's</td>
<td>3.00</td>
</tr>
</tbody>
</table>

Building Complex Expressions

• Algebras allow us to express sequences of operations in a natural way.
  • Example: in arithmetic --- \((x + 4)^*(y - 3)\).
  • Relational algebra allows the same.
  • Three notations, just as in arithmetic:
    1. Sequences of assignment statements.
    2. Expressions with several operators.
    3. Expression trees.
Sequences of Assignments

◆ Create temporary relation names.
◆ Renaming can be implied by giving relations a list of attributes.
◆ Example: R3 := R1 JOIN C R2 can be written:
  R4 := R1 * R2
  R3 := SELECT C(R4)

Expressions in a Single Assignment

◆ Example: the theta-join R3 := R1 JOIN C R2 can be written: R3 := SELECT C(R1 * R2)
◆ Precedence of relational operators:
  1. Unary operators --- select, project, rename --- have highest precedence, bind first.
  2. Then come products and joins.
  3. Then intersection.
  4. Finally, union and set difference bind last.
◆ But you can always insert parentheses to force the order you desire.

Expression Trees

◆ Leaves are operands --- either variables standing for relations or particular, constant relations.
◆ Interior nodes are operators, applied to their child or children.

Example

◆ Using the relations Bars(name, addr) and Sells(bar, beer, price), find the names of all the bars that are either on Maple St. or sell Bud for less than $3.

As a Tree:

◆ Using Sells(bar, beer, price), find the bars that sell two different beers at the same price.
◆ Strategy: by renaming, define a copy of Sells, called S(bar, beer1, price). The natural join of Sells and S consists of quadruples (bar, beer, beer1, price) such that the bar sells both beers at this price.
The Tree

Schemas for Interior Nodes

- An expression tree defines a schema for the relation associated with each interior node.
- Similarly, a sequence of assignments defines a schema for each relation on the left of the := sign.

Schema-Defining Rules 1

- For union, intersection, and difference, the schemas of the two operands must be the same, so use that schema for the result.
- Selection: schema of the result is the same as the schema of the operand.
- Projection: list of attributes tells us the schema.

Schema-Defining Rules 2

- Product: the schema is the attributes of both relations.
  - Use R.A, etc., to distinguish two attributes named A.
- Theta-join: same as product.
- Natural join: use attributes of both relations.
  - Shared attribute names are merged.
- Renaming: the operator tells the schema.

Relational Algebra on Bags

- A bag is like a set, but an element may appear more than once.
  - Multiset is another name for “bag.”
- Example: \( \{1,2,1,3\} \) is a bag. \( \{1,2,3\} \) is also a bag that happens to be a set.
- Bags also resemble lists, but order in a bag is unimportant.
  - Example: \( \{1,2,1\} = \{1,1,2\} \) as bags, but \( [1,2,1] \neq [1,1,2] \) as lists.

Why Bags?

- SQL, the most important query language for relational databases is actually a bag language.
  - SQL will eliminate duplicates, but usually only if you ask it to do so explicitly.
- Some operations, like projection, are much more efficient on bags than sets.
Operations on Bags

- Selection applies to each tuple, so its effect on bags is like its effect on sets.
- Projection also applies to each tuple, but as a bag operator, we do not eliminate duplicates.
- Products and joins are done on each pair of tuples, so duplicates in bags have no effect on how we operate.

Example: Bag Selection

\[
\begin{array}{c|c}
R & S \\hline
\begin{array}{c}
A \text{ } B \\
1 \text{ } 2 \\
5 \text{ } 6 \\
1 \text{ } 2 \\
\end{array} \\
& \begin{array}{c}
B \text{ } C \\
3 \text{ } 4 \\
7 \text{ } 8 \\
\end{array} \\
\end{array}
\]

\[
\text{SELECT}_{\text{A} 	ext{ } \text{B} 	ext{ } \text{C}} (R) = \begin{array}{c|c|c}
\text{A} & \text{B} & \text{C} \\
1 & 2 & 4 \\
\end{array}
\]

Example: Bag Projection

\[
\begin{array}{c|c|c}
R & S \\hline
\begin{array}{c|c}
A & B \\
1 & 2 \\
5 & 6 \\
1 & 2 \\
\end{array} \\
& \begin{array}{c|c|c}
B & C \\
3 & 4 \\
7 & 8 \\
\end{array} \\
\end{array}
\]

\[
\text{PROJECT}_{\text{A}} (R) = \begin{array}{c|c}
\text{A} & \\
1 & 5 \\
1 & 1 \\
\end{array}
\]

Example: Bag Product

\[
\begin{array}{c|c|c|c|c}
R \times S & A & B \text{ } B \text{ } C \\
\begin{array}{c|c|c|c}
R & A & B & \text{B} & \text{C} \\
1 & 2 & 3 & 4 \\
1 & 2 & 7 & 8 \\
5 & 6 & 3 & 4 \\
5 & 6 & 7 & 8 \\
1 & 2 & 3 & 4 \\
1 & 2 & 7 & 8 \\
\end{array} \\
& \begin{array}{c|c|c|c}
\text{B} & \text{C} \\
3 & 4 \\
7 & 8 \\
\end{array} \\
\end{array}
\]

Example: Bag Theta-Join

\[
\begin{array}{c|c|c}
R \text{ } \text{JOIN}_{A 	ext{ } B 	ext{ } C} S & A & \text{R.B} \text{ } \text{S.B} \text{ } \text{C} \\
\begin{array}{c|c|c|c|c}
R & A & B & \text{B} & \text{C} \\
1 & 2 & 3 & 4 \\
1 & 2 & 7 & 8 \\
5 & 6 & 7 & 8 \\
1 & 2 & 3 & 4 \\
1 & 2 & 7 & 8 \\
\end{array} \\
& \begin{array}{c|c|c|c}
\text{B} & \text{C} \\
3 & 4 \\
7 & 8 \\
\end{array} \\
\end{array}
\]

Bag Union

- Union, intersection, and difference need new definitions for bags.
- An element appears in the union of two bags the sum of the number of times it appears in each bag.
- Example: \{1,2,1\} UNION \{1,1,2,3,1\} = \{1,1,1,1,1,2,2,3\}
Bag Intersection

◆ An element appears in the intersection of two bags the minimum of the number of times it appears in either.
◆ Example: \( \{1,2,1\} \text{ INTER } \{1,2,3\} = \{1,2\} \).

Bag Difference

◆ An element appears in the difference \( A - B \) of bags as many times as it appears in \( A \) minus the number of times it appears in \( B \).
  • But never less than 0 times.
◆ Example: \( \{1,2,1\} - \{1,2,3\} = \{1\} \).

Beware: Bag Laws \(!=\) Set Laws

◆ Not all algebraic laws that hold for sets also hold for bags.
◆ For one example, the commutative law for union \( (R \text{ UNION } S = S \text{ UNION } R) \) does hold for bags.
  • Since addition is commutative, adding the number of times \( x \) appears in \( R \) and \( S \) doesn't depend on the order of \( R \) and \( S \).

An Example of Inequivalence

◆ Set union is idempotent, meaning that \( S \text{ UNION } S = S \).
◆ However, for bags, if \( x \) appears \( n \) times in \( S \), then it appears \( 2n \) times in \( S \text{ UNION } S \).
◆ Thus \( S \text{ UNION } S \neq S \) in general.

The Extended Algebra

1. DELTA = eliminate duplicates from bags.
2. TAU = sort tuples.
3. Extended projection : arithmetic, duplication of columns.
4. GAMMA = grouping and aggregation.
5. OUTERJOIN: avoids "dangling tuples" = tuples that do not join with anything.

Duplicate Elimination

◆ R1 := DELTA(R2).
◆ R1 consists of one copy of each tuple that appears in R2 one or more times.
Example: Duplicate Elimination

\[ R = \begin{array}{c|c}
A & B \\
1 & 2 \\
3 & 4 \\
\end{array} \]

\[ \text{DELTA}(R) = \begin{array}{c|c}
A & B \\
1 & 2 \\
3 & 4 \\
\end{array} \]

Sorting

\[ R_1 := \text{TAU}_L(R_2). \]
- \( L \) is a list of some of the attributes of \( R_2 \).
- \( R_1 \) is the list of tuples of \( R_2 \) sorted first on the value of the first attribute of \( L \), then on the second attribute of \( L \), and so on.
- Break ties arbitrarily.
- TAU is the only operator whose result is neither a set nor a bag.

Example: Sorting

\[ R = \begin{array}{c|c}
A & B \\
1 & 2 \\
3 & 4 \\
5 & 2 \\
\end{array} \]

\[ \text{TAU}_L(R) = [(5,2), (1,2), (3,4)] \]

Extended Projection

- Using the same \( \text{PROJ}_L \) operator, we allow the list \( L \) to contain arbitrary expressions involving attributes, for example:
  1. Arithmetic on attributes, e.g., \( A + B \).
  2. Duplicate occurrences of the same attribute.

Example: Extended Projection

\[ R = \begin{array}{c|c}
A & B \\
1 & 2 \\
3 & 4 \\
\end{array} \]

\[ \text{PROJ}_{A+B,A_1,A_2}(R) = \begin{array}{c|c|c}
& A+B & A_1 & A_2 \\
\hline
3 & 7 & 1 & 1 \\
1 & 1 & & \\
\end{array} \]

Aggregation Operators

- Aggregation operators are not operators of relational algebra.
- Rather, they apply to entire columns of a table and produce a single result.
- The most important examples: SUM, AVG, COUNT, MIN, and MAX.
Example: Aggregation

\[
R = \begin{array}{cc}
A & B \\
1 & 3 \\
3 & 4 \\
3 & 3 \\
\end{array}
\]

\[
\text{SUM}(A) = 7 \\
\text{COUNT}(A) = 3 \\
\text{MAX}(B) = 4 \\
\text{AVG}(B) = 3 \\
\]

Grouping Operator

\[ R_1 := \text{GAMMA}_L(R_2). \]
\[ L \text{ is a list of elements that are either:} \]
1. Individual (grouping) attributes.
2. AGG(A), where AGG is one of the aggregation operators and A is an attribute.

Applying \( \text{GAMMA}_L(R) \)

\[ \text{Group } R \text{ according to all the grouping attributes on list } L. \]
\[ \text{That is, form one group for each distinct list of values for those attributes in } R. \]
\[ \text{Within each group, compute AGG(A) for each aggregation on list } L. \]
\[ \text{Result has grouping attributes and aggregations as attributes. One tuple for each list of values for the grouping attributes and their group's aggregations.} \]

Example: Grouping/Aggregation

\[
R = \begin{array}{ccc}
A & B & C \\
1 & 2 & 3 \\
4 & 5 & 6 \\
1 & 2 & 5 \\
\end{array}
\]

Then, average C within groups:

\[
\text{GAMMA}_{A,B}\text{AVG}(C)(R) = \{ \\
\quad \begin{array}{ccc}
A & B & \text{AVG}(C) \\
1 & 2 & 3 \\
4 & 5 & 6 \\
\end{array} \\
\}
\]

Example: Outerjoin

\[
R = \begin{array}{cc}
A & B \\
1 & 2 \\
4 & 5 \\
\end{array}
\]

\[
S = \begin{array}{cc}
B & C \\
2 & 3 \\
5 & 7 \\
\end{array}
\]

(1,2) joins with (2,3), but the other two tuples are dangling.

\[
R \text{ OUTERJOIN } S = \begin{array}{ccc}
A & B & C \\
1 & 2 & \text{null} \\
4 & 5 & 7 \\
\end{array}
\]