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

- \( R_1 := \text{SELECT}_C (R_2) \)
  - \( C \) is a condition (as in "if" statements) that refers to attributes of \( R_2 \).
  - \( R_1 \) is all those tuples of \( R_2 \) that satisfy \( C \).
**Projection**

- \( R_1 := \text{PROJ}_L(R_2) \)
  - \( L \) is a list of attributes from the schema of \( R_2 \).
  - \( R_1 \) is constructed by looking at each tuple of \( R_2 \), extracting the attributes on list \( L \), in the order specified, and creating from those components a tuple for \( R_1 \).
  - Eliminate duplicate tuples, if any.

**Example**

Relation Sells:

\[
\begin{array}{|c|c|c|}
\hline
\text{Bar} & \text{Beer} & \text{Price} \\
\hline
\text{Joe's} & \text{Bud} & 2.50 \\
\text{Joe's} & \text{Miller} & 2.75 \\
\text{Sue's} & \text{Bud} & 2.50 \\
\text{Sue's} & \text{Miller} & 3.00 \\
\hline
\end{array}
\]

JoeMenu := \text{SELECT}_{\text{bar} = \text{Joe's}}(\text{Sells})

\[
\begin{array}{|c|c|c|}
\hline
\text{Bar} & \text{Beer} & \text{Price} \\
\hline
\text{Joe's} & \text{Bud} & 2.50 \\
\text{Joe's} & \text{Miller} & 2.75 \\
\hline
\end{array}
\]

**Example**

Relation Sells:

\[
\begin{array}{|c|c|c|}
\hline
\text{Bar} & \text{Beer} & \text{Price} \\
\hline
\text{Joe's} & \text{Bud} & 2.50 \\
\text{Joe's} & \text{Miller} & 2.75 \\
\text{Sue's} & \text{Bud} & 2.50 \\
\text{Sue's} & \text{Miller} & 3.00 \\
\hline
\end{array}
\]

Prices := \text{PROJ}_{\text{beer,price}}(\text{Sells})

\[
\begin{array}{|c|c|}
\hline
\text{Beer} & \text{Price} \\
\hline
\text{Bud} & 2.50 \\
\text{Miller} & 2.75 \\
\text{Miller} & 3.00 \\
\hline
\end{array}
\]

**Example: \( R_3 := R_1 \ast R_2 \)**

\[
\begin{array}{|c|c|}
\hline
\text{R1} & \text{R2} \\
\hline
\text{A} & \text{B} & \text{A} & \text{B} & \text{R1.B} & \text{R2.B} & \text{C} \\
1 & 2 & 1 & 2 & 5 & 8 \\
3 & 4 & 1 & 2 & 7 & 6 \\
\hline
\end{array}
\]

**Theta-Join**

- \( R_3 := R_1 \text{JOIN}_C R_2 \)
  - Take the product \( R_1 \ast R_2 \).
  - Then apply \( \text{SELECT}_C \) to the result.
  - As for \( \text{SELECT}_C \), \( C \) 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

Sells(bar, beer, price)
Joe's Bud 2.50
Joe's Miller 2.75
Sue's Bud 2.50
Sue's Coors 3.00

Bars(name, addr)
Joe's Maple St.
Joe's Maple St.
Sue's River Rd.
Sue's River Rd.

BarInfo := Sells JOIN_sbar = sname Bars

Example

Sells(bar, beer, price)
Joe's Bud 2.50
Joe's Miller 2.75
Sue's Bud 2.50
Sue's Coors 3.00

Bars(name, addr)
Joe's Maple St.
Joe's Maple St.
Sue's River Rd.
Sue's River Rd.

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

Example

Bars(name, addr)
Joe's Maple St.
Sue's River Rd.

R(bar, addr) := Bars

Example

Natural Join

◆ A frequent type of join connects two relations by:
  • Equating attributes of the same name, and
  • Projecting out one copy of each pair of equated attributes.
◆ Called natural join.
◆ Denoted R3 := R1 JOIN R2.

Example

Sells(bar, beer, price)
Joe's Bud 2.50
Joe's Miller 2.75
Sue's Bud 2.50
Sue's Coors 3.00

Bars(name, addr)
Joe's Maple St.
Joe's Maple St.
Sue's River Rd.
Sue's River Rd.

BarInfo := Sells JOIN_sbar = sname Bars

Example

Sells(bar, beer, price)
Joe's Bud 2.50
Joe's Miller 2.75
Sue's Bud 2.50
Sue's Coors 3.00

Bars(name, addr)
Joe's Maple St.
Joe's Maple St.
Sue's River Rd.
Sue's River Rd.

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

Example

Bars(name, addr)
Joe's Maple St.
Sue's River Rd.

R(bar, addr) := Bars

Example

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 JOINc R2 can be written:
  
  - R4 := R1 × R2
  - R3 := SELECTc(R4)

Expressions in a Single Assignment

- Example: the theta-join R3 := R1 JOINc R2 can be written: R3 := SELECTc (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:

```
  UNION
     |
  RENAME(name)
     |
  PROJECT_bar
     |
  SELECT beer = "Bud"
    |
  Sells
```

Example

- 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

- PROJECT
- SELECT
- JOIN
- RENAME

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

<table>
<thead>
<tr>
<th>R</th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>

S( | B | C |
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>8</td>
<td></td>
</tr>
</tbody>
</table>

SELECT_{A=B} (R) =

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>

Example: Bag Projection

<table>
<thead>
<tr>
<th>R</th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>

PROJECT_{A} (R) =

<table>
<thead>
<tr>
<th>A</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
</tr>
<tr>
<td>5</td>
</tr>
</tbody>
</table>

Example: Bag Product

<table>
<thead>
<tr>
<th>R</th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>

S( | B | C |
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>8</td>
<td></td>
</tr>
</tbody>
</table>

R * S =

<table>
<thead>
<tr>
<th>A</th>
<th>R.B</th>
<th>S.B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>7</td>
<td>8</td>
</tr>
</tbody>
</table>

Example: Bag Theta-Join

<table>
<thead>
<tr>
<th>R</th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>

S( | B | C |
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>8</td>
<td></td>
</tr>
</tbody>
</table>

R JOIN R.B=C.S =

<table>
<thead>
<tr>
<th>A</th>
<th>R.B</th>
<th>S.B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>7</td>
<td>8</td>
</tr>
</tbody>
</table>

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: \{1,2,2\} 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 \( S \text{ UNION } S != S \) 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

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

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

\[
\text{Sorting}
\]

\( R_1 := \text{TAU}_L(R_2). \)
  * \( L \) 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 on \( L \), then on the second attribute of \( L \), and so on.
  * Break ties arbitrarily.
  * \( \text{TAU} \) is the only operator whose result is neither a set nor a bag.

Example: Sorting

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

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

Extended Projection

\[
\text{Using the same PROJ}_L \text{ operator, we allow the list } L \text{ 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

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

\[
\text{PROJ}_{A,B,A_1,A_2}(R) = \begin{bmatrix}
A & B & A_1 & A_2 \\
3 & 1 & 1 & 3 \\
7 & 3 & 3 & 3 \\
\end{bmatrix}
\]

Aggregation Operators

\[
\text{Aggregation operators are not operators of relational algebra.}
\]

\[
\text{Rather, they apply to entire columns of a table and produce a single result.}
\]

\[
\text{The most important examples: SUM, AVG, COUNT, MIN, and MAX.}
\]
Example: Aggregation

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>

SUM(A) = 7
COUNT(A) = 3
MAX(B) = 4
AVG(B) = 3

Grouping Operator

R1 := GAMMA_L(R2). L 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 GAMMA_L(R)

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
<td>C</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>5</td>
</tr>
</tbody>
</table>

Within each group, compute AGG(A) for each aggregation on list L.
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

First, group R:

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
<td>C</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
</tbody>
</table>

Then, average C within groups:

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>AVG(C)</th>
</tr>
</thead>
</table>
| 4 | 5 | 6     

Example: Outerjoin

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
<td>C</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>

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

R OUTERJOIN S =

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
<td>C</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>NULL</td>
</tr>
<tr>
<td>NULL</td>
<td>6</td>
<td>7</td>
</tr>
</tbody>
</table>