Relational Algebra

Operators
Expression Trees
Bag Model of Data

By Prof. Ullman @ Stanford

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 \( C \).
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>

JoeMenu := SELECT_{bar=Joe}(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>
</tbody>
</table>

Projection

R1 := PROJ_{L}(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.

Product

R3 := R1 \times 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 then R2, in order.
- But beware attribute A of the same name in R1 and R2: use R1.A and R2.A.

Example: R3 := R1 \times R2

<table>
<thead>
<tr>
<th>R1(</th>
<th>A</th>
<th>B</th>
<th>R3(</th>
<th>A</th>
<th>R1.B</th>
<th>R2.B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>4</td>
<td>1</td>
<td>2</td>
<td>5</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>7</td>
<td>8</td>
<td></td>
<td></td>
<td></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>2.75</td>
</tr>
<tr>
<td>Miller</td>
<td>3.00</td>
</tr>
</tbody>
</table>

Theta-Join

R3 := R1 JOIN_{C}(R2)
- Take the product R1 \times 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 is =, \text{<}, etc.; hence the name “theta-join.”
Example

Sells(bar, beer, price)
- Joe's Bud 2.90
- Joe's Miller 2.75
- Sue's Bud 2.50
- Sue's Coors 3.00

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

BarInfo := Sells JOIN Sells.bar = Bars.name
BarInfo(
  bar, beer, price, name, addr
)
- Joe's Bud 2.50 Joe's Maple St.
- Joe's Miller 2.75 Joe's Maple St.
- Sue's Bud 2.50 Sue's River Rd.
- Sue's Coors 3.00 Sue's River Rd.

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(bar, addr)
- Joe's Maple St.
- Sue's River Rd.

BarInfo := Sells JOIN Bars
Note Bars.name has become Bars.bar to make the natural join "work."
BarInfo(
  bar, beer, price, addr
)
- Joe's Bud 2.50 Maple St.
- Joe's Miller 2.75 Maple St.
- Sue's Bud 2.50 River Rd.
- Sue's Coors 3.00 River Rd.

Example

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

R(bar, addr) := Bars
R(bar, addr)
- Joe's Maple St.
- Sue's River Rd.

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.

Renaming

◆ The RENAME operator gives a new schema to a relation.
◆ R1 := RENAME R1(A1,...,An) := R2 makes R1 be a relation with attributes A1,...,An and the same tuples as R2.
◆ Simplified notation: R1(A1,...,An) := R2.

Building Complex Expressions

◆ Combine operators with parentheses and precedence rules.
◆ 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:
  \( R4 := R1 \ast R2 \)
  \( R3 := \text{SELECT}_{c}(R4) \)

Expressions in a Single Assignment

- Example: the theta-join \( R3 := R1 \ JOIN_c R2 \) can be written: \( R3 := \text{SELECT}_{c}(R1 \ast R2) \)
- Precedence of relational operators:
  1. [SELECT, PROJECT, RENAME] (highest).
  2. [PRODUCT, JOIN].
  3. INTERSECTION.
  4. [UNION, -]

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

As a Tree:

Example

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

Schemas for Results

- 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.

Schemas for Results --- (2)

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

Relational Algebra on Bags

- A bag (or multiset) is like a set, but an element may appear more than once.
- Example: \( \{1,2,1,3\} \) is a bag.
- Example: \( \{1,2,3\} \) is also a bag that happens to be a set.

Why Bags?

- SQL, the most important query language for relational databases, is actually a bag language.
- 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**

\[ R(\begin{array}{c}
A \\
1 \\
5 \\
1
\end{array}, \begin{array}{c}
B \\
2 \\
6 \\
2
\end{array}) \]

\[ \text{SELECT}_{A,B:S} (R) = \begin{array}{c}
A \\
1 \\
5 \\
1
\end{array}, \begin{array}{c}
B \\
2 \\
6 \\
2
\end{array} \]

**Example: Bag Projection**

\[ R(\begin{array}{c}
A \\
1 \\
5 \\
1
\end{array}) \]

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

**Example: Bag Product**

\[ R(\begin{array}{c}
A \\
1 \\
5 \\
1
\end{array}, \begin{array}{c}
B \\
2 \\
6 \\
2
\end{array}) \]

\[ S(\begin{array}{c}
B \\
3 \\
7
\end{array}, \begin{array}{c}
C \\
4 \\
8
\end{array}) \]

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

**Example: Bag Theta-Join**

\[ R(\begin{array}{c}
A \\
1 \\
5 \\
1
\end{array}) \]

\[ S(\begin{array}{c}
B \\
3 \\
7
\end{array}, \begin{array}{c}
C \\
4 \\
8
\end{array}) \]

\[ R \Join_{A,B:S} S = \begin{array}{c|c|c|c}
A & R.B & S.B & C \\
--- & --- & --- & --- \\
1 & 2 & 3 & 4 \\
1 & 2 & 7 & 8 \\
5 & 6 & 7 & 8 \\
1 & 2 & 3 & 4 \\
1 & 2 & 7 & 8
\end{array} \]

**Bag Union**

- 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\} = \{1,1,1,1,2,2,3\}

**Bag Intersection**

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

Beware: Bag Laws $\neq$ Set Laws

- Some, but not all algebraic laws that hold for sets also hold for bags.
- Example: the commutative law for union ($R \cup S = S \cup R$) 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 $S$.

Example of the Difference

- Set union is idempotent, meaning that $S \cup S = S$.
- However, for bags, if $x$ appears $n$ times in $S$ then it appears $2n$ times in $S \cup S$.
- Thus $S \cup S \neq S$ 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

- $R1 := \text{DELTA}(R2)$.
- $R1$ consists of one copy of each tuple that appears in $R2$ one or more times.

Example: Duplicate Elimination

$$
R = \begin{pmatrix}
1 & 2 \\
3 & 4 \\
1 & 2
\end{pmatrix}
$$

$$
\text{DELTA}(R) = \begin{pmatrix}
1 & 2 \\
3 & 4
\end{pmatrix}
$$
Sorting

◆ R1 := TAU_L (R2).
  ♦ L is a list of some of the attributes of R2.
  ♦ R1 is the list of tuples of R2 sorted first on the value of the first attribute on \( L \),
    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

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

TAU_L(R) = [(5,2), (1,2), (3,4)]

Extended Projection

◆ Using the same 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

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

PROJ_{A+B,A1,A2}(R) = \[
\begin{pmatrix}
  A+B & A1 & A2 \\
  3 & 1 & 1 \\
  7 & 3 & 3 \\
\end{pmatrix}
\]

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

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

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)

- Group R according to all the grouping attributes on list L.
  - That is: form one group for each distinct list of values for those attributes in R.
- Within each group, compute AGG(A) for each aggregation on list L.
- Result has one tuple for each group:
  1. The grouping attributes and
  2. Their group's aggregations.

Example: Grouping/Aggregation

R = 
<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<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>

GAMMA_A,B [AVG(C)] (R) = ??

First, group R by A and B:

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

Then, average C within groups:

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>AVG(C)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>4.0</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>6.0</td>
</tr>
</tbody>
</table>

Outerjoin

- Suppose we join R JOIN_L S.
- A tuple of R that has no tuple of S with which it joins is said to be dangling.
  - Similarly for a tuple of S.
- Outerjoin preserves dangling tuples by padding them with a special NULL symbol in the result.

Example: Outerjoin

R = 
<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
</tr>
</tbody>
</table>

S = 
<table>
<thead>
<tr>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>6</td>
<td>7</td>
</tr>
</tbody>
</table>

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

R OUTERJOIN S = 
<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<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>