Relational Algebra on Bags

• A \textit{bag} (or \textit{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.

- Practical considerations: Some operations, like projection, are much more efficient on bags than sets.
Operations on Bags

- **Selection** operate the same way on bags as on sets.

- **Projection** do not eliminate duplicates on bags.

- **Products** and **joins** operate the same way on bags as on sets.
Example: Bag Selection

\[ R( (A, B) ) \]

<table>
<thead>
<tr>
<th></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>

\[ \text{SELECT}_{A+B<5} (R) = \]

\[ \begin{array}{|c|c|}
\hline
A & B \\
\hline
1 & 2 \\
1 & 2 \\
\hline
\end{array} \]
Example: Bag Projection

\[ R( A, B ) = \]

<table>
<thead>
<tr>
<th></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>

\[ \text{PROJECT}_A (R) = \]

<table>
<thead>
<tr>
<th>A</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>1</td>
</tr>
</tbody>
</table>
Example: Bag Product

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

R( A, B ) = S( B, C )

R \times 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>3</td>
<td>4</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

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

\[ R \text{ JOIN}_{R.B < S.B} S = \]

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

- The number of times an element appears in the union of two bags = the sum of the it appears in each bag.

- Example: \( \{1,2,1\} \cup \{1,1,2,3,1\} = \{1,1,1,1,1,2,2,3\} \)
Bag Intersection

• The number of times 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

• The number of times an element appears in the difference $A - B = \text{the number of times it appears in } A \text{ minus the number of times it appears in } B$.  
  – But never less than 0 times.

• Example: $\{1,2,1,1\} - \{1,2,3\} = \{1,1\}$. 
Beware: Bag Laws != Set Laws

• Some, but not all algebraic laws that hold for sets also hold for bags.

• Example: commutative law for union \((R \cup S = S \cup R)\)

  – Does it hold for bags?
  – Yes

  – 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 Bag Laws != Set Laws

- Set union is *idempotent*, meaning that $S \text{ UNION } S = S$.
- Does this law hold on bags? Why?

- For bags, $S \text{ UNION } S \neq S$ in general.
- Because, if $x$ appears $n$ times in $S$, then it appears $2n$ times in $S \text{ UNION } S$. 
The Extended Algebra

1. **DELTA** = eliminate duplicates from bags.
2. **TAU** = sort tuples.
4. **GAMMA** = grouping and aggregation.
5. **Outerjoin**: avoids “dangling tuples” = tuples that do not join with anything.
Duplicate Elimination

- \( R_1 := \text{DELTA}(R_2) \).
  - \( R_1 \) consists of one copy of each tuple that appears in \( R_2 \) one or more times.
Example: Duplicate Elimination

\[ R = ( \begin{array}{cc} A & B \\ 1 & 2 \\ 3 & 4 \\ 1 & 2 \end{array} ) \]

\[ \text{DELTA}(R) = \begin{array}{cc} A & B \\ 1 & 2 \\ 3 & 4 \end{array} \]
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.
  - Descending vs. ascending (default) order.
  - Break ties arbitrarily.

- TAU is the only operator whose result is neither a set nor a bag.
Example: Sorting

\[ R = (A, B) \]

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

\[ TAU_B (R) = [(5,2), (1,2), (3,4)] \]
Extended Projection

- Using the same $\text{PROJ}_\mathcal{L}$ operator, we allow the list $\mathcal{L}$ 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

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

\[ \text{PROJ}_{A+B,A,A}(R) = \begin{array}{ccc} \text{A+B} & \text{A1} & \text{A2} \\ 3 & 1 & 1 \\ 7 & 3 & 3 \end{array} \]
Aggregation Operators

• Aggregation operators are not operators of relational algebra.
  – Rather than applying to each tuple the same operation 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

\[ R = (A, B) \]

<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>

\[
\begin{align*}
\text{SUM}(A) &= 7 \\
\text{COUNT}(A) &= 3 \\
\text{MAX}(B) &= 4 \\
\text{AVG}(B) &= 3
\end{align*}
\]
Grouping Operator

• \( R_1 := \text{GAMMA}_L (R2). \)

• \( L \) is a list of elements that are either:
  1. Individual \((\text{grouping})\) attributes.
  2. \( \text{AGG}(A) \), where AGG is one of the aggregation operators and \( A \) is a non-grouping attribute.
Applying $\text{GAMMA}_L(R)$

- Group $R$ according to all the grouping attributes on list $L$.

- Within each group, compute $\text{AGG}(A)$ for each aggregation on list $L$.

- Result has one tuple for each group, incl:
  1. The grouping attributes and
  2. Their group’s aggregations.
Example: Grouping/Aggregation

\[ R = (A, B, C) \]

<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>

Then, average \( C \) within groups:

\[ \text{GAMMA}_{A, B, \text{AVG}(C)}(R) = ?? \]

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>

<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</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
</tbody>
</table>
Outerjoin

- Suppose we join $R \ JOIN_C S$.

- *Dangling Tuple*: A tuple of $R$ that does not joins with any tuple in $S$.
  - Similarly for a tuple of $S$.

- Outerjoin preserves dangling tuples
  - by padding them with a special NULL symbol in the result.
Example: Outerjoin

\[ R = ( \begin{array}{cc}
A & B \\
1 & 2 \\
4 & 5 \\
\end{array} ) \quad S = ( \begin{array}{cc}
B & C \\
2 & 3 \\
6 & 7 \\
\end{array} ) \]

\[ R \text{ NATURALJOIN } S \]

\[ R \text{ OUTERJOIN } S = \]

\[
\begin{array}{ccc}
A & B & C \\
1 & 2 & 3 \\
4 & 5 & \text{NULL} \\
\text{NULL} & 6 & 7 \\
\end{array}
\]
### Example Instances

#### Sailors

<table>
<thead>
<tr>
<th>sid</th>
<th>sname</th>
<th>rating</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>dustin</td>
<td>7</td>
<td>45.0</td>
</tr>
<tr>
<td>31</td>
<td>lubber</td>
<td>8</td>
<td>55.5</td>
</tr>
<tr>
<td>58</td>
<td>rusty</td>
<td>10</td>
<td>35.0</td>
</tr>
</tbody>
</table>

#### Reserves

<table>
<thead>
<tr>
<th>sid</th>
<th>bid</th>
<th>day</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>101</td>
<td>10/10/96</td>
</tr>
<tr>
<td>58</td>
<td>103</td>
<td>11/12/96</td>
</tr>
</tbody>
</table>
Find names of sailors who’ve reserved boat #103

- **Solution 1:** \( \pi_{sname}((\sigma_{\text{bid}=103}\text{Reserves}) \bowtie \text{Sailors}) \)

- **Solution 2:** \( \rho(\text{Temp1}, \sigma_{\text{bid}=103}\text{Reserves}) \)

  \( \rho(\text{Temp2}, \text{Temp1} \bowtie \text{Sailors}) \)

  \( \pi_{sname}(\text{Temp2}) \)

- **Solution 3:** \( \pi_{sname}(\sigma_{\text{bid}=103}(\text{Reserves} \bowtie \text{Sailors})) \)
Find names of sailors who’ve reserved a red boat

- Information about boat color only available in Boats; so need an extra join:
  \[ \pi_{sname}((\sigma_{color='red'} Boats) \bowtie Reserves \bowtie Sailors) \]

- A more efficient solution:
  \[ \pi_{sname}(\pi_{sid}((\pi_{bid} \sigma_{color='red'} Boats) \bowtie Res) \bowtie Sailors) \]

A query optimizer can find this, given the first solution!
Find sailors who’ve reserved a red or a green boat

\[
\rho \ (Tempboats, (\sigma_{\text{color} = 'red' \lor \text{color} = 'green'} \ Boats))
\]

\[
\pi_{\text{sname}} (Tempboats \bowtie Reserves \bowtie Sailors)
\]

- What happens if \( \lor \) is replaced by \( \land \) in this query?
- Can also define Tempboats using union! (How?)
Find sailors who’ve reserved a red and a green boat

\[ \rho \left( \text{Tempred}, \pi_{\text{sid}} \left( \left( \sigma_{\text{color} = 'red'} \left( \text{Boats} \bowtie \text{Reserves} \right) \right) \right) \right) \]

\[ \rho \left( \text{Tempgreen}, \pi_{\text{sid}} \left( \left( \sigma_{\text{color} = 'green'} \left( \text{Boats} \bowtie \text{Reserves} \right) \right) \right) \right) \]

\[ \pi_{\text{sname}} \left( \left( \text{Tempred} \cap \text{Tempgreen} \right) \bowtie \text{Sailors} \right) \]
RA Summary

- The relational model has rigorously defined query languages that are simple and powerful.
- Relational algebra is more operational; useful as internal representation for query evaluation plans.
- Several ways of expressing a given query; a query optimizer should choose the most efficient version.