Algebra of Relations

- Operands = relation (including attributes) or a variable representing a relation.
- Operators = union (\( \cup \)), difference (\( - \)), selection (\( \sigma \)), projection (\( \pi \)), product (\( \times \)).
  - Also, important operators intersection (\( \cap \)) and join (\( \bowtie \)) defined in terms of these.

Why Relational Algebra?

- Very expressive language with operators that "do a lot," e.g., \( R \cup S \) implies a complex algorithm with lots of details we don't have to specify.
- Like all algebras, the algebraic laws let us "optimize" expressions into equivalent forms that are cheaper to evaluate.
  - For relations, where data is large and operators powerful, this ability makes orders of magnitude difference in running time.

Union, Intersection, Difference

- As for sets.
  - But schemes must agree (or rename the attributes).
  - Result has same scheme as operands.
- Note intersection in terms of difference: \( R \cap S = R - (R - S) \).

Selection

\( \sigma_C(R) = \) relation of all tuples of \( R \) that satisfy condition \( C \).
  - \( C \) refers to attributes, representing components of the tuples.
  - Result has same scheme as \( R \).
**Example:** \( \sigma_{\text{Weight} \geq 400000}(\text{Classes}) = \) “find all those classes displacing at least 40,000 tons.”

<table>
<thead>
<tr>
<th>Class</th>
<th>Weight</th>
<th>Guns</th>
<th>Caliber</th>
<th>Type</th>
<th>Country</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hood</td>
<td>41000</td>
<td>8</td>
<td>15</td>
<td>BC</td>
<td>Gt. Br.</td>
</tr>
<tr>
<td>Iowa</td>
<td>46000</td>
<td>9</td>
<td>16</td>
<td>BB</td>
<td>USA</td>
</tr>
<tr>
<td>Yamato</td>
<td>65000</td>
<td>9</td>
<td>18</td>
<td>BB</td>
<td>Japan</td>
</tr>
</tbody>
</table>

**Projection**

\( \pi_S(R) = \) take from each tuple of relation \( R \) those components for the attributes in list \( S \).

- Scheme = attributes of \( S \).

**Example:** \( \pi_{\text{Guns}, \text{Caliber}}(\sigma_{\text{Country} = \text{“USA”}}(\text{Classes})) \) = “List the number of guns and calibers of the US capital ships.”

<table>
<thead>
<tr>
<th>Guns</th>
<th>Caliber</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>12</td>
</tr>
<tr>
<td>8</td>
<td>16</td>
</tr>
<tr>
<td>9</td>
<td>16</td>
</tr>
<tr>
<td>10</td>
<td>14</td>
</tr>
<tr>
<td>12</td>
<td>14</td>
</tr>
<tr>
<td>12</td>
<td>12</td>
</tr>
</tbody>
</table>

**Cartesian Product**

\( R \times S = \) take each tuple of \( R \) and pair it with each tuple of \( S \).

- Scheme = attributes of \( R \), then attributes of \( S \).

  - If attribute \( A \) appears in both schemes, use \( A.R \) and \( A.S \) in result scheme.

- Not commonly used; generally appears in a “join” = product followed by selection.

**Natural Join**

\( R \bowtie S = \)

1. Take \( R \times S \).
2. Select for equality between each pair of attributes with the same name.
3. Project out one of each pair of equated attributes.
Example: Ships $\bowtie$ Classes extends the Ships tuples with all the information about its class.

- Example tuples:

<table>
<thead>
<tr>
<th>Name</th>
<th>Lncd</th>
<th>Class</th>
<th>Wt.</th>
<th>Guns</th>
<th>Cal.</th>
<th>Type</th>
<th>Cntry</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alabama</td>
<td>1942</td>
<td>S. Dakota</td>
<td>37000</td>
<td>9</td>
<td>16 BB</td>
<td>USA</td>
<td></td>
</tr>
<tr>
<td>Alaska</td>
<td>1944</td>
<td>Alaska</td>
<td>28000</td>
<td>9</td>
<td>12 BC</td>
<td>USA</td>
<td></td>
</tr>
</tbody>
</table>

Join Algorithms

- Very expensive operation — number of tuples can be product of number in the two operand relations.
  - If almost all tuples agree on the shared attributes.

- Methods: Index-join and Sort-Join.

Index-Join

Compute $R \bowtie S$ by:

\[
\text{for } (r \in R) \{ \\
\text{find tuples } s \in S \text{ matching } r \text{ in shared attributes; } \\
\text{produce tuple from } r \text{ and } s; \\
\}
\]

- Helps greatly if there is an index for $S$ on one of the shared attributes.
- If no index, create temporary hash table (often called hash-join).
- Note that natural join is "sort of" commutative — the result scheme has a different order, but the information in $R \bowtie S$ is the same as in $S \bowtie R$.
  - Thus, an index for $R$ on a shared attribute is as useful as one on $S$.
- With maximum-efficiency index, time on relations of size $n$ is $O(n)$ plus big-oh of output size (possibly much larger than $O(n)$).

Sort-Join

- Sort $R$ and $S$ on their common attributes.
• Run through the sorted lists to group tuples from both relations that have the same values for shared attributes.

• Time is $O(n \log n)$ plus big-oh of output size.

Query Optimization

• Many algebraic equivalences.

• Major efficiency gains obtained by doing size-reducing operations (selection and projection) as early as possible.

• "Pushing selections down." $\sigma_C(R \bowtie S) \equiv \sigma_C(R) \bowtie S$, provided condition $C$ refers only to attributes present in the scheme of $S$.

  □ Similar push to $S$ if attributes of $C$ are there.

• "Splitting selections." $\sigma_C(\sigma_D(R)) \equiv \sigma_C(\sigma(D(R))$.

Example: “What ships launched after 1940 had guns of less than 15-inch caliber?” In SQL, the compiler would interpret it as the algebraic expression

$$\pi_{Name}(\sigma_{Launched \geq 1940 \text{ and Caliber} < 15}(Ships \bowtie Classes))$$

• Requires the join of two large relations.

• Split selection and push each part down the side where it makes sense:

$$\pi_{Name}(\sigma_{Launched \geq 1940}(Ships) \bowtie \sigma_{\text{Caliber} < 15}(Classes))$$

• Selections produce subsets of each of the two large relations.

• Answer to either query = \{Alaska, Anson, Duke of York, Guam, Howe, Prince of Wales\}, a 1-ary relation with scheme Name.