Relational Query Languages

- **Query languages**: Allow manipulation and retrieval of data from a database.

- Relational model supports simple, powerful QLs:
  - Strong formal foundation based on logic.
  - Allows for much optimization.

- **Query Languages != programming languages!**
  - QLs not expected to be “Turing complete”.
  - QLs not intended to be used for complex calculations.
  - QLs support easy, efficient access to large data sets.
Formal Relational Query Languages

- Two mathematical Query Languages form the basis for “real” languages (e.g. SQL), and for implementation:
  - **Relational Algebra**: More operational, very useful for representing execution plans.
  - **Relational Calculus**: Lets users describe what they want, rather than how to compute it. (Non-operational, *declarative*.)
Preliminaries

- A query is applied to *relation instances*, and the result of a query is also a relation instance.
  - *Schemas of input* relations for a query are fixed (but query will run regardless of instance!)
  - The *schema for the result* of a given query is also fixed! Determined by definition of query language constructs.
Example Instances

“Sailors” and “Reserves” relations for our examples.

<table>
<thead>
<tr>
<th>S1</th>
<th>sid</th>
<th>surname</th>
<th>rating</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>22</td>
<td>dustin</td>
<td>7</td>
<td>45.0</td>
</tr>
<tr>
<td></td>
<td>31</td>
<td>lubber</td>
<td>8</td>
<td>55.5</td>
</tr>
<tr>
<td></td>
<td>58</td>
<td>rusty</td>
<td>10</td>
<td>35.0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>S2</th>
<th>sid</th>
<th>surname</th>
<th>rating</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>28</td>
<td>yuppy</td>
<td>9</td>
<td>35.0</td>
</tr>
<tr>
<td></td>
<td>31</td>
<td>lubber</td>
<td>8</td>
<td>55.5</td>
</tr>
<tr>
<td></td>
<td>44</td>
<td>guppy</td>
<td>5</td>
<td>35.0</td>
</tr>
<tr>
<td></td>
<td>58</td>
<td>rusty</td>
<td>10</td>
<td>35.0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>R1</th>
<th>sid</th>
<th>bid</th>
<th>day</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>22</td>
<td>101</td>
<td>10/10/96</td>
</tr>
<tr>
<td></td>
<td>58</td>
<td>103</td>
<td>11/12/96</td>
</tr>
</tbody>
</table>
Relational Algebra

❖ Basic operations:
  ▪ **Selection** (σ) Selects a subset of rows from relation.
  ▪ **Projection** (Π) Deletes unwanted columns from relation.
  ▪ **Cross-product** (×) Allows us to combine two relations.
  ▪ **Set-difference** (−) Tuples in reln. 1, but not in reln. 2.
  ▪ **Union** (∪) Tuples in reln. 1 and in reln. 2.

❖ Additional operations:
  ▪ Intersection, **join**, division, renaming: Not essential, but (very!) useful.

❖ Since each operation returns a relation, **operations can be composed**! (Algebra is “closed”.)
Projection

- Deletes attributes that are not in projection list.
- *Schema* of result contains exactly the fields in the projection list, with the same names that they had in the (only) input relation.
- Projection operator has to eliminate duplicates! (Why??)
  - Note: real systems typically don’t do duplicate elimination unless the user explicitly asks for it. (Why not?)

\[
\pi_{\text{snames}, \text{rating}}(S2)
\]

\[
\begin{array}{|c|c|}
\hline
\text{sname} & \text{rating} \\
\hline
\text{yuppy lubber} & 9 \\
\text{guppy} & 8 \\
\text{rusty} & 10 \\
\hline
\end{array}
\]

\[
\pi_{\text{age}}(S2)
\]

\[
\begin{array}{|c|}
\hline
\text{age} \\
\hline
35.0 \\
55.5 \\
\hline
\end{array}
\]
Selection

- Selects rows that satisfy selection condition.
- No duplicates in result! (Why?)
- Schema of result identical to schema of (only) input relation.
- Result relation can be the input for another relational algebra operation! (Operator composition.)

<table>
<thead>
<tr>
<th>sid</th>
<th>surname</th>
<th>rating</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>28</td>
<td>yuppy</td>
<td>9</td>
<td>35.0</td>
</tr>
<tr>
<td>31</td>
<td>lubber</td>
<td>8</td>
<td>55.5</td>
</tr>
<tr>
<td>44</td>
<td>guppy</td>
<td>5</td>
<td>35.0</td>
</tr>
<tr>
<td>58</td>
<td>rusty</td>
<td>10</td>
<td>35.0</td>
</tr>
</tbody>
</table>

\[ \sigma_{\text{rating} > 8}(S2) \]

<table>
<thead>
<tr>
<th>surname</th>
<th>rating</th>
</tr>
</thead>
<tbody>
<tr>
<td>yuppy</td>
<td>9</td>
</tr>
<tr>
<td>rusty</td>
<td>10</td>
</tr>
</tbody>
</table>

\[ \pi_{\text{name, rating}}(\sigma_{\text{rating} > 8}(S2)) \]
All of these operations take two input relations, which must be union-compatible:
- Same number of fields.
- `Corresponding’ fields have the same type.

What is the schema of result?

### $S1 \cup S2$

<table>
<thead>
<tr>
<th>sid</th>
<th>name</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>

### $S1 \cap S2$

<table>
<thead>
<tr>
<th>sid</th>
<th>name</th>
<th>rating</th>
<th>age</th>
</tr>
</thead>
<tbody>
<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>

### $S1 - S2$

<table>
<thead>
<tr>
<th>sid</th>
<th>name</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>
</tbody>
</table>
### Cross-Product

- Each row of S1 is paired with each row of R1.
- **Result schema** has one field per field of S1 and R1, with field names `inherited' if possible.
  - Conflict: Both S1 and R1 have a field called `sid'.

<table>
<thead>
<tr>
<th>sid</th>
<th>name</th>
<th>rating</th>
<th>age</th>
<th>sid</th>
<th>bid</th>
<th>day</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>dustin</td>
<td>7</td>
<td>45.0</td>
<td>22</td>
<td>101</td>
<td>10/10/96</td>
</tr>
<tr>
<td>22</td>
<td>dustin</td>
<td>7</td>
<td>45.0</td>
<td>58</td>
<td>103</td>
<td>11/12/96</td>
</tr>
<tr>
<td>31</td>
<td>lubber</td>
<td>8</td>
<td>55.5</td>
<td>22</td>
<td>101</td>
<td>10/10/96</td>
</tr>
<tr>
<td>31</td>
<td>lubber</td>
<td>8</td>
<td>55.5</td>
<td>58</td>
<td>103</td>
<td>11/12/96</td>
</tr>
<tr>
<td>58</td>
<td>rusty</td>
<td>10</td>
<td>35.0</td>
<td>22</td>
<td>101</td>
<td>10/10/96</td>
</tr>
<tr>
<td>58</td>
<td>rusty</td>
<td>10</td>
<td>35.0</td>
<td>58</td>
<td>103</td>
<td>11/12/96</td>
</tr>
</tbody>
</table>

- **Renaming operator:** $\rho \ (C(1 \rightarrow sid_1, 5 \rightarrow sid_2), S1 \times R1)$
Joins

❖ **Condition Join**: 

\[
R \bowtie_c S = \sigma_c(R \times S)
\]

\[
\begin{array}{|c|c|c|c|}
\hline
\text{sid} & \text{name} & \text{rating} & \text{age} \\
\hline
22 & dustin & 7 & 45.0 \\
31 & lubber & 8 & 55.5 \\
58 & rusty & 10 & 35.0 \\
\hline
\end{array}
\]

\[
\begin{array}{|c|c|c|}
\hline
\text{sid} & \text{bid} & \text{day} \\
\hline
22 & 101 & 10/10/96 \\
58 & 103 & 11/12/96 \\
\hline
\end{array}
\]

\[
R \bowtie_{R_1.\text{sid}<S_1.\text{sid}} S
\]

❖ **Result schema** same as that of cross-product.
❖ Fewer tuples than cross-product, might be able to compute more efficiently
❖ Sometimes called a *theta-join*. 

\[
R \bowtie_c S = \sigma_c(R \times S)
\]
Joins

- **Equi-Join**: A special case of condition join where the condition $c$ contains only **equalities**.

  \[
  R \bowtie_{sid} S
  \]

  - **Result schema** similar to cross-product, but only one copy of fields for which equality is specified.
  
  - **Natural Join**: Equijoin on all common fields.
Division

- Not supported as a primitive operator, but useful for expressing queries like:

  Find sailors who have reserved all boats.

- Let $A$ have 2 fields, $x$ and $y$; $B$ have only field $y$:
  - $A/B = \left\{ \langle x \rangle \mid \exists \langle x, y \rangle \in A \quad \forall \langle y \rangle \in B \right\}$
  - i.e., $A/B$ contains all $x$ tuples (sailors) such that for every $y$ tuple (boat) in $B$, there is an $xy$ tuple in $A$.
  - Or: If the set of $y$ values (boats) associated with an $x$ value (sailor) in $A$ contains all $y$ values in $B$, the $x$ value is in $A/B$.

- In general, $x$ and $y$ can be any lists of fields; $y$ is the list of fields in $B$, and $x \cup y$ is the list of fields of $A$. 
Find sailors who have reserved all boats?

**Examples of Division A/B**

<table>
<thead>
<tr>
<th>sno</th>
<th>pno</th>
<th>pno</th>
<th>pno</th>
<th>pno</th>
</tr>
</thead>
<tbody>
<tr>
<td>s1</td>
<td>p1</td>
<td>p2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>s1</td>
<td>p2</td>
<td></td>
<td>p2</td>
<td>p4</td>
</tr>
<tr>
<td>s1</td>
<td>p3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>s1</td>
<td>p4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>s2</td>
<td>p1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>s2</td>
<td>p2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>s3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>s4</td>
<td>p2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>s4</td>
<td>p2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>s4</td>
<td>p4</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>sno</th>
<th>pno</th>
<th>pno</th>
<th>pno</th>
<th>pno</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

A

A/B1

A/B2

A/B3
Find sailors who have reserved all boats? 

\((A/B)\)

\[\pi_{sno}(A) \times B - A\]

\[\pi_{sno}(A) - \pi_{sno}(\pi_{sno}(A) \times B - A)\]

\(A/B = A - \text{disqualified tuples}\)
Expressing A/B Using Basic Operators

- Division is not essential op; just a useful shorthand.
  - (Also true of joins, but joins are so common that systems implement joins specially.)

- Idea: For A/B, compute all x values that are not `disqualified’ by some y value in B.
  - x value is disqualified if by attaching y value from B, we obtain an xy tuple that is not in A.

Disqualified x values: \( \pi_x ((\pi_x (A) \times B) - A) \)

\( A/B: \pi_x (A) \) – all disqualified tuples
Find names of sailors who’ve reserved boat #103

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

Solution 2: \( \rho(\text{Temp1},\sigma_{\text{bid}=103}\text{Reserves}) \)
\[ \rho(\text{Temp2},\text{Temp1} \bowtie \text{Sailors}) \]
\[ \pi_{\text{name}}(\text{Temp2}) \]

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

- Information about boat color only available in Boats; so need an extra join:

$$\pi_{\text{sname}}((\sigma_{\text{color} = 'red'} \text{Boats}) \bowtie \text{Reserves} \bowtie \text{Sailors})$$

- A more efficient solution:

$$\pi_{\text{sname}}(\pi_{\text{sid}}(\pi_{\text{bid}}(\sigma_{\text{color} = 'red'} \text{Boats}) \bowtie \text{Reserves}) \bowtie \text{Sailors}))$$

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

- Can identify all red or green boats, then find sailors who’ve reserved one of these boats:

\[
\rho(Tempboats, (\sigma_{\text{color}='red' \lor \text{color}='green'} Boats)) \ni_{\text{sname}} (Tempboats \bowtie Reserves \bowtie Sailors)
\]

- Can also define Tempboats using union! (How?)

- What happens if \( \lor \) is replaced by \( \land \) in this query?
Find sailors who’ve reserved a red and a green boat

- Previous approach won’t work! Must identify sailors who’ve reserved red boats, sailors who’ve reserved green boats, then find the intersection (note that \( sid \) is a key for Sailors):

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

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

\[
\pi_{sname}((Tempred \cap Tempgreen) \bowtie Sailors)
\]

\[
\begin{array}{|c|c|c|c|} 
\hline
\text{sid} & \text{sname} & \text{rating} & \text{age} \\
\hline
22 & dustin & 7 & 45.0 \\
31 & lubber & 8 & 55.5 \\
58 & rusty & 10 & 35.0 \\
\hline
\end{array}
\]

\[
\begin{array}{|c|c|c|} 
\hline
\text{sid} & \text{bid} & \text{day} \\
\hline
22 & 101 & 10/10/96 \\
58 & 103 & 11/12/96 \\
\hline
\end{array}
\]
Find the names of sailors who’ve reserved all boats

- Uses division; schemas of the input relations to / must be carefully chosen:
  \[ \rho(Tempsids, (\pi_{sid,bid} Reserves)/(\pi_{bid} Boats)) \]
  \[ \pi_{sname} (Tempsids \bowtie Sailors) \]

- To find sailors who’ve reserved all ‘Interlake’ boats:
  \[ \ldots / \pi_{} bid (\sigma_{bname = 'Interlake'} Boats) \]
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.