By relieving the brain of all unnecessary work, a good notation sets it free to concentrate on more advanced problems, and, in effect, increases the mental power of the race.

-- Alfred North Whitehead (1861 - 1947)

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.

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 over any legal instance)
  – The schema for the result of a given query is also **fixed**. It is determined by the definitions of the query language constructs.
• **Positional vs. named-field notation:**
  – Positional notation easier for formal definitions, named-field notation more readable.
  – Both used in SQL
    • Though positional notation is not encouraged

Relational Algebra: 5 Basic Operations

• **Selection** (σ) Selects a subset of **rows** from relation (horizontal).
• **Projection** (π) Retains only wanted **columns** from relation (vertical).
• **Cross-product** (×) Allows us to combine two relations.
• **Set-difference** (−) Tuples in r1, but not in r2.
• **Union** (∪) Tuples in r1 or in r2.

Since each operation returns a relation, operations can be **composed**! (Algebra is "closed").

Example Instances

<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>

Boats

<table>
<thead>
<tr>
<th>bid</th>
<th>bname</th>
<th>color</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>Interlake</td>
<td>blue</td>
</tr>
<tr>
<td>102</td>
<td>Interlake</td>
<td>red</td>
</tr>
<tr>
<td>103</td>
<td>Clipper</td>
<td>green</td>
</tr>
<tr>
<td>104</td>
<td>Marine</td>
<td>red</td>
</tr>
</tbody>
</table>

<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>45.0</td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>lubber</td>
<td>55.5</td>
<td></td>
</tr>
<tr>
<td>58</td>
<td>rusty</td>
<td>35.0</td>
<td></td>
</tr>
</tbody>
</table>

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-procedural, declarative.)
  ➤ Understanding Algebra & Calculus is key to understanding SQL, query processing!
**Projection (\(\pi\))**

- Examples: \(\pi_{\text{age}}(S2)\), \(\pi_{\text{name}, \text{rating}}(S2)\)
- Retains only attributes that are in the "projection list".
- **Schema** of result:
  - exactly the fields in the projection list, with the same names that they had in the input relation.
- Projection operator has to **eliminate duplicates** (How do they arise? Why remove them?)
  - Note: real systems typically don’t do duplicate elimination unless the user explicitly asks for it. (Why not?)

![Example Projection](image)

**Selection (\(\sigma\))**

- Selects rows that satisfy **selection condition**.
- Result is a relation.
- **Schema** of result is same as that of the input relation.
- Do we need to do duplicate elimination?

![Example Selection](image)

**Union and Set-Difference**

- Both of these operations take two input relations, which must be **union-compatible**:
  - Same number of fields.
  - "Corresponding" fields have the same type.
- For which, if any, is duplicate elimination required?

![Example Union and Set-Difference](image)
**Cross-Product**

- S1 \times R1: Each row of S1 paired with each row of R1.
- Q: How many rows in the result?
- **Result schema** has one field per field of S1 and R1, with field names `inherited` if possible.
  - May have a naming conflict: Both S1 and R1 have a field with the same name.
  - In this case, can use the renaming operator:
    \[ \rho (C(1 \rightarrow sid1, S \rightarrow sid2), S1 \times R1) \]
    
**Cross Product Example**

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

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

**Compound Operator: Intersection**

- In addition to the 5 basic operators, there are several additional "Compound Operators"
  - These add no computational power to the language, but are useful shorthands.
  - Can be expressed solely with the basic ops.
  - Intersection takes two input relations, which must be union-compatible.
  - Q: How to express it using basic operators?
    \[ R \cap S = R - (R - S) \]

**Intersection**

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

**Compound Operator: Join**

- Joins are compound operators involving cross product, selection, and (sometimes) projection.
- Most common type of join is a "natural join" (often just called "join"). \( R \bowtie S \) conceptually is:
  - Compute \( R \times S \)
  - Select rows where attributes that appear in both relations have equal values
  - Project all unique attributes and one copy of each of the common ones.
- Note: Usually done much more efficiently than this.

**Natural Join Example**

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

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

- **Condition Join (or "theta-join")**
  \[ R \bowtie_c S = \sigma_c (R \times S) \]

  - Result schema same as that of cross-product.
  - May have fewer tuples than cross-product.

- **Equi-Join**: Special case: condition \( c \) contains only conjunction of equalities.

Find names of sailors who’ve reserved boat #103

- **Solution 1**: \( \pi_{\text{sname}}((\sigma_{\text{bid}_1=103} \text{Reserves}) \bowtie_R \text{Sailors}) \)
- **Solution 2**: \( \pi_{\text{sname}}(\sigma_{\text{bid}_1=103} (\text{Reserves} \bowtie_R \text{Sailors})) \)

Boats

<table>
<thead>
<tr>
<th>bid</th>
<th>bname</th>
<th>color</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>Interlake</td>
<td>Blue</td>
</tr>
<tr>
<td>102</td>
<td>Interlake</td>
<td>Red</td>
</tr>
<tr>
<td>103</td>
<td>Clipper</td>
<td>Green</td>
</tr>
<tr>
<td>104</td>
<td>Marine</td>
<td>Red</td>
</tr>
</tbody>
</table>

Find names of sailors who’ve reserved a red boat

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

  - A more efficient solution:
    \[ \pi_{\text{sname}}(\sigma_{\text{bid}_1=103} (\sigma_{\text{color} = \text{red}} (\text{Boats} \bowtie_R \text{Reserves} \bowtie_R \text{Sailors}))) \]

  - 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:
  \[ \rho(\text{Tempboats}, (\sigma_{\text{color} = \text{red}} \lor \text{color} = \text{green}, \text{Boats})) \]
  \[ \pi_{\text{sname}}(\text{Tempboats} \bowtie_R \text{Reserves} \bowtie_R \text{Sailors}) \]

Find sailors who’ve reserved a red and a green boat

- Cut-and-paste previous slide?
  \[ \rho(\text{Tempboats}, (\sigma_{\text{color} = \text{red}} \land \text{color} = \text{green}, \text{Boats})) \]
  \[ \pi_{\text{sname}}(\text{Tempboats} \bowtie_R \text{Reserves} \bowtie_R \text{Sailors}) \]
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):
  \[ \rho (\text{Tempred}, \pi_{\text{sid}} (\sigma_{\text{color=red}} (\text{Boats}))) \bowtie \text{Reserves}) \]
  \[ \rho (\text{Tempgreen}, \pi_{\text{sid}} (\sigma_{\text{color=green}} (\text{Boats}))) \bowtie \text{Reserves}) \]
  \[ \pi_{\text{name}} (\text{Tempred} \cap \text{Tempgreen}) \bowtie \text{Sailors}) \]

**Summary**

- Relational Algebra: a small set of operators mapping relations to relations
  - Operational, in the sense that you specify the explicit order of operations
  - A closed set of operators! Can mix and match.
- Basic ops include: \( \sigma, \pi, \times, \cup, \rho \)
- Important compound ops: \( \cap, \bowtie \)