Relational Algebra

Chapter 4
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 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: 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

<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>

- “Sailors” and “Reserves” relations for our examples.
- We’ll use positional or named field notation, assume that names of fields in query results are ‘inherited’ from names of fields in query input relations.
Relational Algebra

- **Basic operations:**
  - *Selection* ($\sigma$) Selects a subset of rows from relation.
  - *Projection* ($\pi$) Deletes unwanted columns from relation.
  - *Cross-product* ($\times$) Allows us to combine two relations.
  - *Set-difference* ($\setminus$) Tuples in reln. 1, but not in reln. 2.
  - *Union* ($\cup$) 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!**
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.*)

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

<table>
<thead>
<tr>
<th>sid</th>
<th>sname</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>
Projection

- Deletes attributes that are not in projection list.

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

<table>
<thead>
<tr>
<th>sname</th>
<th>rating</th>
</tr>
</thead>
<tbody>
<tr>
<td>yummy</td>
<td>9</td>
</tr>
<tr>
<td>lubber</td>
<td>8</td>
</tr>
<tr>
<td>guppy</td>
<td>5</td>
</tr>
<tr>
<td>rusty</td>
<td>10</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>35.0</td>
</tr>
<tr>
<td>55.5</td>
</tr>
</tbody>
</table>

- 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?)
**Selection + Projection**

- Result relation can be the input for another relational algebra operation! *(Operator composition.)*

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

- Is it the same as:

\[ \pi_{\text{sname}, \text{rating}}(\sigma_{\text{rating} > 8}(S2)) \]
Union-compatibility

Two relations are considered to be union-compatible if the following conditions hold:

- Same number of fields.
- ‘Corresponding’ fields have the same type.

Notice that field names are not important. Two schemas can be union-compatible even if they have different field names.

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

<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>
Union, Intersection, Set-Difference

- All of these operations require **union-compatibility**

<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>

<table>
<thead>
<tr>
<th>sid</th>
<th>sname</th>
<th>rating</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>28</td>
<td>uppy</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>

- The schema of the result has the same names as of the first relation.
Cross-Product (Cartesian Product)

<table>
<thead>
<tr>
<th></th>
<th>sid</th>
<th>name</th>
<th>rating</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</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></th>
<th>sid</th>
<th>bid</th>
<th>day</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1</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>

- 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`.

- **Renaming operator**: \(\rho (C(1 \to \text{sid}1, 5 \to \text{sid}2), S1 \times R1)\)

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke
Joins

- **Condition Join:**
  \[ R \bowtie_c S = \sigma_c (R \times S) \]

\[ S_1 \bowtie S_1.sid < R_1.sid \]

<table>
<thead>
<tr>
<th>(sid)</th>
<th>sname</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>31</td>
<td>lubber</td>
<td>8</td>
<td>55.5</td>
<td>58</td>
<td>103</td>
<td>11/12/96</td>
</tr>
</tbody>
</table>

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

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

\[ S_1 \bowtie_{sid} R_1 \]

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

- **Result schema** similar to cross-product, but only one copy of fields for which equality is specified.

- **Natural Join**: Equijoin on all common fields.