Introduction to Data Management

Lecture #11
(Relational Languages I)

Instructor: Mike Carey
mjcarey@ics.uci.edu
Announcements

- Homework notes
  - HW #3 is due Friday (Saturday if one day late)
  - HW #4 will come out on Monday (after the exam)

- Exam notes (time flies!)
  - Midterm #1 is next Monday (in class)
  - We’ll use assigned seating – come early!
  - You may bring an 8.5” x 11” (2-sided) cheat sheet
  - Don’t bank it all on last-minute Piazza answers…!

- Today’s lecture plan:
  - Relational languages – the next frontier…
  - Note: Today’s material won’t be on Monday’s exam! 😊
But 1st: A Word on Learning….

- ”When *I* was your age…” (😊)
  - 8-bit μprocessors, LEDs, hex keypad, 16-bit for its address space, …
  - Unix first appeared, Unix/INGRES DB 64K process design point, and 1MB of memory was amazing!
  - FORTRAN, Pascal, C, Lisp, Snobol, APL, Quel …
  - No web! (Just the early ArpaNet)

- Fast forward 35 years…
  - Your cell phones dwarf our ~1980 computing platforms
  - Python, Java, Go, C++, Ruby, SQL, SQL++, …
  - Twitter… (😢)
  - WHAT THIS MEANS: It’s critical to learn how to read and learn, how to search for info/resources, etc…!!!
On to 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 their 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, or 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 applying the definitions of the query language’s constructs.

Positional vs. named-field notation:

- Positional notation easier for formal definitions, but named-field notation far more readable.
- Both used in SQL (but try to avoid positional stuff!)
Example Instances

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

<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>

<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>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>
Relational Algebra

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

- **Additional operations:**
  - Intersection, *join*, division, renaming: Not essential, but (very!) useful. (I.e., don’t add expressive power, but…)

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

- Removes 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.
- **Relational** projection operator has to eliminate *duplicates!* (Why??)
  - Note: real systems typically don’t do duplicate elimination unless the user explicitly asks for it. *(Q: Why not?)*
Selection

- Selects rows that satisfy a selection condition.
- No duplicates in result! (Why?)
- Schema of result identical to schema of its (only) input relation.
- Result relation can be the input for another relational algebra operation! (This is 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>58</td>
<td>rusty</td>
<td>10</td>
<td>35.0</td>
</tr>
</tbody>
</table>

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

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

<table>
<thead>
<tr>
<th>sname</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>
Union, Intersection, Set-Difference

- All of these operations take two input relations, which must be union-compatible:
  - Same number of fields.
  - “Corresponding” fields are of the same type.
- What is the schema of result?

\[
\begin{array}{cccc}
\text{sid} & \text{sname} & \text{rating} & \text{age} \\
22 & \text{dustin} & 7 & 45.0 \\
31 & \text{lubber} & 8 & 55.5 \\
58 & \text{rusty} & 10 & 35.0 \\
44 & \text{guppy} & 5 & 35.0 \\
28 & \text{yuppy} & 9 & 35.0 \\
\end{array}
\]

\[
S1 \cup S2
\]

\[
\begin{array}{cccc}
\text{sid} & \text{sname} & \text{rating} & \text{age} \\
31 & \text{lubber} & 8 & 55.5 \\
58 & \text{rusty} & 10 & 35.0 \\
\end{array}
\]

\[
S1 \cap S2
\]

\[
\begin{array}{cccc}
\text{sid} & \text{sname} & \text{rating} & \text{age} \\
31 & \text{lubber} & 8 & 55.5 \\
58 & \text{rusty} & 10 & 35.0 \\
\end{array}
\]

Q: Any issues w/duplicates?
Cross-Product

- $S1 \times R1$: Each $S$ row is paired with each $R1$ row.
- 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>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>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**: $\rho \left( C(1\rightarrow sid1, 5\rightarrow sid2), S1 \times R1 \right)$
Renaming

- **Conflict**: S1 and R1 both had *sid* fields, giving:

<table>
<thead>
<tr>
<th>(sid)</th>
<th>surname</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></td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</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>

- Several renaming notations available:

\[
\rho (S1 \bowtie R1 (1 \mapsto sid1), S1 \times R1)
\]

\[
\rho (\text{Temp}S1 (sid \mapsto sid1), S1)
\]

\[
\text{Temp}S1 \times R1
\]

\[
(\pi \text{sid} \mapsto \text{sid1}, \text{surname}, \text{rating}, \text{age}) (S1) \times R1
\]

- Positional renaming
- Name-based renaming
- Generalized projection

*(I like this notation best! 😊)*
Joins

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

<table>
<thead>
<tr>
<th>(sid)</th>
<th>surname</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>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>58</td>
<td>103</td>
<td>11/12/96</td>
</tr>
</tbody>
</table>

\( S_1 \bowtie_{S1.sid < R1.sid} R_1 \)

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

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

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

<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>58</td>
<td>rusty</td>
<td>10</td>
<td>35.0</td>
<td>103</td>
<td>11/12/96</td>
</tr>
</tbody>
</table>

\[
S_1 \bowtie_{S_1.sid = R_1.sid} R_1
\]

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

- Not a primitive operator, but extremely useful for expressing queries like:
  Find sailors who have reserved all boats.

- Let $A$ have 2 fields, $x$ and $y$, while $B$ has one field $y$, so we have relations $A(x,y)$ and $B(y)$:
  - $A/B$ contains the $x$ tuples (e.g., sailors) such that for every $y$ tuple (e.g., 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 $A$. 
Examples of Division A/B

A

\[
\begin{array}{c|c}
\text{sno} & \text{pno} \\
\hline
\text{s1} & \text{p1} \\
\text{s1} & \text{p2} \\
\text{s1} & \text{p3} \\
\text{s1} & \text{p4} \\
\text{s2} & \text{p1} \\
\text{s2} & \text{p2} \\
\text{s3} & \text{p2} \\
\text{s4} & \text{p2} \\
\text{s4} & \text{p4} \\
\end{array}
\]

A/B1

\[
\begin{array}{c}
\text{pno} \\
\hline
\text{p2} \\
\text{p2} \\
\text{p4} \\
\end{array}
\]

B1

\[
\begin{array}{c|c}
\text{sno} & \\
\hline
\text{s1} & \\
\text{s2} & \\
\text{s3} & \\
\text{s4} & \\
\end{array}
\]

A/B2

\[
\begin{array}{c}
\text{pno} \\
\hline
\text{p2} \\
\text{p4} \\
\end{array}
\]

B2

\[
\begin{array}{c|c}
\text{sno} & \\
\hline
\text{s1} & \\
\text{s4} & \\
\end{array}
\]

B3

\[
\begin{array}{c}
\text{pno} \\
\hline
\text{p1} \\
\text{p2} \\
\text{p4} \\
\end{array}
\]

A/B3
Expressing A/B Using Basic Operators
(Advanced Topic – Just FYI 😊)

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

- **Idea:** For $A(x,y)/B(y)$, compute all $x$ values that are not “disqualified” by some $y$ value in $B$.
  - $x$ value is disqualified if by attaching a $y$ value from $B$, we obtain an $xy$ tuple that does not appear in $A$.

  Disqualified $x$ values ($D$):

  $$\pi_x \left( (\pi_x (A) \times B) - A \right)$$

  $$A/B: \quad \pi_x (A) \ - \ D$$
### Wisconsin Sailing Club Database

<table>
<thead>
<tr>
<th>Sailors</th>
<th>Reserves</th>
<th>Boats</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>sid</strong></td>
<td><strong>bid</strong></td>
<td><strong>bid</strong></td>
</tr>
<tr>
<td><strong>sname</strong></td>
<td><strong>date</strong></td>
<td><strong>bname</strong></td>
</tr>
<tr>
<td><strong>rating</strong></td>
<td><strong>date</strong></td>
<td><strong>color</strong></td>
</tr>
<tr>
<td><strong>age</strong></td>
<td><strong>date</strong></td>
<td><strong>color</strong></td>
</tr>
<tr>
<td><strong>sid</strong></td>
<td><strong>bid</strong></td>
<td><strong>bid</strong></td>
</tr>
<tr>
<td><strong>sname</strong></td>
<td><strong>date</strong></td>
<td><strong>bname</strong></td>
</tr>
<tr>
<td><strong>rating</strong></td>
<td><strong>date</strong></td>
<td><strong>bname</strong></td>
</tr>
<tr>
<td><strong>age</strong></td>
<td><strong>date</strong></td>
<td><strong>bname</strong></td>
</tr>
</tbody>
</table>

#### Sailors
- **sid**: 22, Rating: 7, Age: 45.0
- **sid**: 29, Rating: 1, Age: 33.0
- **sid**: 31, Rating: 8, Age: 55.5
- **sid**: 32, Rating: 8, Age: 25.5
- **sid**: 58, Rating: 10, Age: 35.0
- **sid**: 64, Rating: 7, Age: 35.0
- **sid**: 71, Rating: 10, Age: 16.0
- **sid**: 74, Rating: 9, Age: 35.0
- **sid**: 85, Rating: 4, Age: 25.5
- **sid**: 95, Rating: 3, Age: 63.5

#### Reserves
- **sid**: 22, **bid**: 101, **date**: 10/10/98
- **sid**: 22, **bid**: 102, **date**: 10/10/98
- **sid**: 22, **bid**: 103, **date**: 10/8/98
- **sid**: 22, **bid**: 104, **date**: 10/7/98
- **sid**: 31, **bid**: 102, **date**: 11/10/98
- **sid**: 31, **bid**: 103, **date**: 11/6/98
- **sid**: 31, **bid**: 104, **date**: 11/12/98
- **sid**: 64, **bid**: 101, **date**: 9/5/98
- **sid**: 64, **bid**: 102, **date**: 9/8/98
- **sid**: 74, **bid**: 103, **date**: 9/8/93

#### Boats
- **bid**: 101, **bname**: Interlake, **color**: blue
- **bid**: 102, **bname**: Interlake, **color**: red
- **bid**: 103, **bname**: Clipper, **color**: green
- **bid**: 104, **bname**: Marine, **color**: red
Find names of sailors who’ve reserved boat #103

Sailors(sid, sname, rating, age)  Reserves(sid, bid, day)
Boats(bid, bname, color)

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

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

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

Sailors(sid, sname, rating, age)  Reserves(sid, bid, day)
Boats(bid, bname, color)

Solution 1:

\[
\pi_{sname}((\sigma_{bid=103} \text{Reserves}) \bowtie \text{Sailors})
\]

Solution 2:

Temp1 = \(\sigma_{bid=103} \text{Reserves}\)

Temp2 = Temp1 \bowtie \text{Sailors}

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

Solution 3:

\[
\pi_{sname}(\sigma_{bid=103}((\text{Reserves}\bowtie \text{Sailors}))
\]
### Ex: Wisconsin Sailing Club Database

**σ** \( bid=103 \) \( Reserves \)

<table>
<thead>
<tr>
<th>sid</th>
<th>bid</th>
<th>date</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>103</td>
<td>10/8/98</td>
</tr>
<tr>
<td>31</td>
<td>103</td>
<td>11/6/98</td>
</tr>
<tr>
<td>74</td>
<td>103</td>
<td>9/8/93</td>
</tr>
</tbody>
</table>

**π** \( sname \) \( ((σ \ biid=103 \ Reserves) \Join \ Sailors) \)

<table>
<thead>
<tr>
<th>sname</th>
<th>rating</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dustin</td>
<td>7</td>
<td>45.0</td>
</tr>
<tr>
<td>Lubber</td>
<td>8</td>
<td>55.5</td>
</tr>
<tr>
<td>Horatio</td>
<td>9</td>
<td>35.0</td>
</tr>
</tbody>
</table>

(Solution 1)
Find names of sailors who’ve reserved a red boat

Sailors(sid, sname, rating, age)  Reserves(sid, bid, day)
Boats(bid, bname, color)

- Information about boat color only available in Boats; so need to do another join:

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

- A more “efficient” solution:

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

A query optimizer will find the latter, given the 1st query!