Announcements

- HW#4 is now underway...!
  - Today we’ll finish the relevant algebra material
  - Read ahead w.r.t. the relational calculus
  - Another lateness discount lies ahead.... (☹)
- Midterm exam is looming!
  - Tue, May 9th, from 12:30-1:50 PM
  - Here (with pre-assigned seating)
  - Left-handed? → E-mail Taewoo today!
  - See course wiki for exam scope (!)
- Today’s plan:
  - RELATIONAL ALGEBRA continued!
Reminder: Relational Algebra

- Five basic operations:
  - Selection ($\sigma$) Selects a subset of rows from relation.
  - Projection ($\pi$) Omits unwanted columns from relation.
  - Set-difference (-) Tuples in reln. 1, but not in reln. 2.
  - Union (U) Tuples in reln. 1 and in reln. 2.
  - Cross-product (X) Allows us to combine two relations.

- 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, all these operations can be composed! (Algebra is “closed”.)

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>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: $\rho (C(1 \rightarrow sid1, 5 \rightarrow sid2), S1 \times R1)$
Renaming

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

<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>...</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 options available:
  - \( \rho (S1R1(l \rightarrow sid1), S1 \times R1) \) **Positional renaming**
  - \( \rho (TempS1(sid \rightarrow sid1), S1) \) **Name-based renaming**
  - \( TempS1 \times R1 \)
  - \( \pi_{sid \rightarrow sid1, sname, rating, age} (S1) \times R1 \)**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>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>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>

- **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**.
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>

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

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

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/B, 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): \( \pi_x ((\pi_x (A) \times B) - A) \)

A/B: \( \pi_x (A) - D \)
**Ex: Wisconsin Sailing Club Database**

<table>
<thead>
<tr>
<th>Sailors</th>
<th>Reserves</th>
<th>Boats</th>
</tr>
</thead>
<tbody>
<tr>
<td>sid</td>
<td>name</td>
<td>rating</td>
</tr>
<tr>
<td>22</td>
<td>Dustin</td>
<td>7</td>
</tr>
<tr>
<td>29</td>
<td>Brutus</td>
<td>1</td>
</tr>
<tr>
<td>31</td>
<td>Lubber</td>
<td>8</td>
</tr>
<tr>
<td>32</td>
<td>Andy</td>
<td>8</td>
</tr>
<tr>
<td>58</td>
<td>Rusty</td>
<td>10</td>
</tr>
<tr>
<td>64</td>
<td>Horatio</td>
<td>7</td>
</tr>
<tr>
<td>71</td>
<td>Zorba</td>
<td>10</td>
</tr>
<tr>
<td>74</td>
<td>Horatio</td>
<td>9</td>
</tr>
<tr>
<td>85</td>
<td>Art</td>
<td>4</td>
</tr>
<tr>
<td>95</td>
<td>Bob</td>
<td>3</td>
</tr>
</tbody>
</table>

**Find names of sailors who’ve reserved boat #103**

- **Sailors** (sid, name, rating, age)
- **Reserves** (sid, bid, day)
- **Boats** (bid, bname, color)

- **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 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:
  \[ \text{Temp}1 = \sigma_{bid=103} \text{Reserves} \]
  \[ \text{Temp}2 = \text{Temp}1 \bowtie \text{Sailors} \]
  \[ \pi_{sname}(\text{Temp}2) \]
- Solution 3:
  \[ \pi_{sname}(\sigma_{bid=103}(\text{Reserves} \bowtie \text{Sailors})) \]

Ex: Wisconsin Sailing Club Database

\[ \sigma_{bid=103} \text{Reserves} \]

<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>

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

<table>
<thead>
<tr>
<th>sid</th>
<th>bid</th>
<th>date</th>
<th>sname</th>
<th>rating</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>103</td>
<td>10/8/98</td>
<td>Dustin</td>
<td>7</td>
<td>45.0</td>
</tr>
<tr>
<td>31</td>
<td>103</td>
<td>11/6/98</td>
<td>Lubber</td>
<td>8</td>
<td>55.5</td>
</tr>
<tr>
<td>74</td>
<td>103</td>
<td>9/8/93</td>
<td>Horatio</td>
<td>9</td>
<td>35.0</td>
</tr>
</tbody>
</table>
Find names of sailors who’ve reserved a red boat

- Information about boat color only available in Boats; so need to do another join:
  \[ \pi_{sname}((\sigma_{\text{color} = 'red'} \text{Boats}) \bowtie \text{Reserves} \bowtie \text{Sailors}) \]

- A more “efficient” solution:
  \[ \pi_{sname}(\pi_{\text{sid}}((\pi_{\text{bid}}\sigma_{\text{color} = 'red'} \text{Boats}) \bowtie \text{Res} \bowtie \text{Sailors}) \]

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

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

- Can also define Tempboats using union! (Q: 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 and sailors who’ve reserved green boats, then find their intersection (notice that sid is a key for Sailors!):

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

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

\[
\pi_{\text{sname}} \left( \left( \text{Tempred} \cap \text{Tempgreen} \right) \bowtie \text{Sailors} \right)
\]
Find the names of sailors who’ve reserved all boats

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

- Uses division; schemas of the input relations feeding the / operator must be carefully chosen:

\[ \rho (\text{Temp}sid, (\pi_{\text{sid}, \text{bid}} \text{Reserves}) / (\pi_{\text{bid}} \text{Boats})) \]
\[ \pi_{\text{sname}} (\text{Temp}sid \bowtie \text{Sailors}) \]

- To find sailors who’ve reserved all ‘Interlake’ boats:

\[ \ldots / \pi_{\text{bid}} (\sigma_{bname = 'Interlake'} \text{Boats}) \]
Find the names of sailors who’ve reserved all boats:

\[
\rho \pi \pi (\pi (\sigma \text{bid} \text{bname} \text{Interlake} \text{Boats}) = \ldots) / (\pi \text{bid} \text{Boats})) \bowtie \text{Sailors}
\]

To find sailors who’ve reserved all ‘Interlake’ boats:

\[
\text{PS: RelaX Renaming Example...}
\]

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke
Relational Algebra Summary

- The relational model has (several) rigorously defined query languages that are both simple and powerful in nature.
- Relational algebra is more operational; very useful as an internal representation for query evaluation plans.
- Several ways of expressing a given query; a query optimizer should choose the most efficient version. (Take CS122C...! 😊)
- We’ll add a few more operators later on…
- Next up for now: Relational Calculus

Relational Calculus

- Comes in two flavors: Tuple relational calculus (TRC) and Domain relational calculus (DRC).
- Calculus has variables, constants, comparison ops, logical connectives and quantifiers.
  - TRC: Variables range over (i.e., get bound to) tuples.
  - DRC: Variables range over domain elements (= field values).
  - Both TRC and DRC are simple subsets of first-order logic.
- Expressions in the calculus are called formulas. An answer tuple is essentially an assignment of constants to variables that make the formula evaluate to true.
- TRC is the basis for various query languages (Quel, SQL, OQL, XQuery, ...), while DRC is the basis for example-based relational query UIs. We’ll study TRC!
**Tuple Relational Calculus**

- *Query* in TRC has the form:
  \[ \{ t(\text{attrlist}) \mid \text{P(t)} \} \]

- *Answer* includes all tuples t with (optionally) specified schema (attrlist) that cause formula \text{P(t)} to be *true*.

- *Formula* is recursively defined, starting with simple *atomic formulas* (getting tuples from relations or making comparisons of values), and building up bigger and better Boolean formulas using *logical connectives*.

---

**TRC Formulas**

- *Atomic formula*:
  - \( r \in R, \text{ or } r \notin R, \text{ or } r.a \text{ op } s.b, \text{ or } r.a \text{ op } \text{constant} \)
  - \( \text{op} \) is one of \(<, >, \leq, \geq, \neq, =\)

- *Formula*:
  - an atomic formula, or
  - \( \neg P, P \land Q, P \lor Q \), where P and Q are formulas, or
  - \( \exists r \in R (P(r)) \), where variable r is *free* in \( P(...) \), or
  - \( \forall r \in R (P(r)) \), where variable r is *free* in \( P(...) \), or
  - \( P \Rightarrow Q \) (pronounced “implies”, equivalent to \( (\neg P) \lor Q) \)
Free and Bound Variables

- The use of a quantifier such as $\exists t \in T$ or $\forall t \in T$ in a formula is said to bind $t$.
  - A variable that is not bound is free.
- Now let us revisit the definition of a TRC query:
  - $\{ t(a_1, a_2, \ldots) \mid P(t) \}$
- There is an important restriction: the variable $t$ that appears to the left of the $\mid$ ("such that") symbol must be the only free variable in the formula $P(...)$.
- Let’s look at some examples!

Find sailors with a rating above 7

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

{ s \mid s \in Sailors \land s.rating > 7 }

- This is equivalent to the more general form:
  - $\{ t(id, nm, rtg, age) \mid \exists s \in Sailors$
    - ( t.id = s.sid \land t.nm = s.sname
      \land t.rtg = s.rating \land t.age = s.age
      \land s.rating > 7 ) \}$

(Q: See how each one specifies the answer’s schema and values...? Note that the second one’s schema is different, as we’ve specified it.)
Find ids of sailors who are older than 30.0 or who have a rating under 8 and are named “Horatio”

\[
\{ \text{t}(\text{sid}) \mid \exists s \in \text{Sailors} \ ( (s.\text{age} > 30.0 \lor (s.\text{rating} < 8 \land s.\text{sname} = \text{“Horatio”})) \\
\quad \land t.\text{sid} = s.\text{sid} ) \}
\]

- Things to notice:
  - Again, how result schema and values are specified
  - Use of Boolean formula to specify the query constraints
  - Highly declarative nature of this form of query language!

Ex: TRC Query Semantics
Unsafe Queries and Expressive Power

- It is possible to write syntactically correct calculus queries that have an infinite number of answers! Such queries are called unsafe.
  - E.g., \( s \mid s \notin \text{Sailors} \)
- It is known that every query that can be expressed in relational algebra can be expressed as a safe query in DRC / TRC; the converse is also true.
- **Relational Completeness**: Query language (e.g., SQL) can express every query that is expressible in relational algebra/calculus.

To Be Continued...

- We’ll finish the TRC material on Thursday!
- (Meanwhile, read the book’s TRC sections!)