Announcements

- HW#4 is now underway...!
  - Today we’ll finish the relevant algebra material
  - Then we’ll move on to the relational calculus
  - Deadline still Friday 5PM (as originally planned)
- I will also be trying out a “new toy” today
  - Suggested by one of you after the last lecture
  - Warning...
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 sailors who’ve reserved a **red** and a **green** boat

- Simple predicate approach won’t work! Must identify (i) sailors who have reserved **red** boats and (ii) sailors who’ve reserved **green** boats, then find their intersection: (Note: `sid` is a key for Sailors!)

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

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

\[
\pi_{\text{name}}(\left( \text{Tempred} \cap \text{Tempgreen} \right) \bowtie \text{Sailors})
\]
Find sailors who’ve reserved a red and a green boat:

1. TempRed = \( \pi_{sid} (\sigma_{color = 'red'}(Boats) \times Reserves) \)
2. TempGrn = \( \pi_{sid} (\sigma_{color = 'green'}(Boats) \times Reserves) \)
3. \( \pi_{sname} ((TempRed \cap TempGrn) \times Sailors) \)

Find the names of sailors who’ve reserved all boats:

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

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

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

- Use division to find sailors who’ve reserved all boats:
  
  $$\text{Tempsids} \leftarrow \text{sailor number} \times \text{Bid}$$

- To find sailors who’ve reserved all Interlake boats:
  
  $$\text{Boats.bname} = \text{Interlake}$$

- SQL query to find sailors who’ve reserved all boats:

  ```sql
  SELECT T.sname
  FROM Sailors AS T
  WHERE T.sid IN (SELECT R.sid
                   FROM Reserves AS R
                   WHERE R.bid IN (SELECT B.bid
                                      FROM Boats AS B
                                      WHERE B.bname = 'Interlake'))
  ```

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. (SQL “EXPLAIN”)
- 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
NEXT: 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:
  \[
  \{ t(\text{attrlist}) \mid P(t) \}
  \]

- **Answer** includes all tuples \( t \) with (optionally) specified schema (\text{attrlist}) that cause formula \( P(t) \) 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:**
  - \( r \in R \), or \( r \notin R \), or \( r \cdot a \text{ op } s \cdot b \), or \( r \cdot a \text{ op } \text{constant} \)
  - \( \text{op} \) 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 \( P(\ldots) \), or
  - \( \forall r \in R (P(r)) \), where variable \( r \) is free in \( P(\ldots) \), 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 \( P(\ldots) \).
- Let’s look at some examples!
**Find sailors with a rating above 7**

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

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

(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”**

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

- 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!
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.*, \( s \mid \neg (s \in \text{Sailors}) \)

- 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...