Announcements

- HW and exams:
  - HW #4 info
    - Still ongoing, due tomorrow (6 PM)
  - Midterm Exam 1
    - Grading is still underway...
- This week’s discussion session plan:
  - Brief quiz on (all!) relational languages
  - Chance to ask Q’s about published HW solutions
- Today’s lecture plan:
  - More about SQL...!
**Example Data in MySQL**

### Sailors

<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>29</td>
<td>Brutus</td>
<td>1</td>
<td>33.0</td>
</tr>
<tr>
<td>31</td>
<td>Lubber</td>
<td>8</td>
<td>55.5</td>
</tr>
<tr>
<td>32</td>
<td>Andy</td>
<td>8</td>
<td>25.5</td>
</tr>
<tr>
<td>58</td>
<td>Rusty</td>
<td>10</td>
<td>35.0</td>
</tr>
<tr>
<td>64</td>
<td>Horatio</td>
<td>7</td>
<td>35.0</td>
</tr>
<tr>
<td>71</td>
<td>Zorba</td>
<td>10</td>
<td>16.0</td>
</tr>
<tr>
<td>74</td>
<td>Horatio</td>
<td>9</td>
<td>35.0</td>
</tr>
<tr>
<td>85</td>
<td>Art</td>
<td>4</td>
<td>25.5</td>
</tr>
<tr>
<td>95</td>
<td>Bob</td>
<td>3</td>
<td>63.5</td>
</tr>
<tr>
<td>101</td>
<td>Joan</td>
<td>3</td>
<td>NULL</td>
</tr>
<tr>
<td>107</td>
<td>Johan</td>
<td>5</td>
<td>35.0</td>
</tr>
</tbody>
</table>

### Reserves

<table>
<thead>
<tr>
<th>sid</th>
<th>bid</th>
<th>date</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>101</td>
<td>1998-10-10</td>
</tr>
<tr>
<td>22</td>
<td>102</td>
<td>1998-10-10</td>
</tr>
<tr>
<td>22</td>
<td>103</td>
<td>1998-10-08</td>
</tr>
<tr>
<td>22</td>
<td>104</td>
<td>1998-10-07</td>
</tr>
<tr>
<td>31</td>
<td>102</td>
<td>1998-11-10</td>
</tr>
<tr>
<td>31</td>
<td>103</td>
<td>1998-11-06</td>
</tr>
<tr>
<td>31</td>
<td>104</td>
<td>1998-11-12</td>
</tr>
<tr>
<td>64</td>
<td>101</td>
<td>1998-09-05</td>
</tr>
<tr>
<td>64</td>
<td>102</td>
<td>1998-09-08</td>
</tr>
<tr>
<td>74</td>
<td>103</td>
<td>1998-09-08</td>
</tr>
<tr>
<td>101</td>
<td>103</td>
<td>1998-09-09</td>
</tr>
<tr>
<td>1</td>
<td>NULL</td>
<td>2001-01-11</td>
</tr>
<tr>
<td>1</td>
<td>NULL</td>
<td>2002-02-02</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>

---

**Find sid’ s of sailors who ’ve reserved a red or a green boat**

- If we replace **OR** by **AND** in this first version, what do we get?
- **UNION**: Can be used to compute the union of any two *union-compatible* sets of tuples (which are themselves the result of SQL queries).
- Also available: **EXCEPT**

(Note: MySQL vs. RelaX – and why?)
**SQL vs. TRC**

*Find sid’s of sailors who’ve reserved a red or a green boat*

\[
\{ \text{t(sid)} \mid \exists s \in \text{Sailors} (t.sid = s.sid \land \exists r \in \text{Reserves} (r.sid = s.sid \land \exists b \in \text{Boats} (b.bid = r.bid \land (b.color = 'red' \lor b.color = 'green'))) \}
\]

```
SELECT  S.sid
FROM    Sailors S, Boats B, Reserves R
        AND (B.color= 'red' OR B.color= 'green')
```

```
SQL vs. TRC
Find sid’s of sailors who’ve reserved a red or a green boat

SELECT  S.sid
FROM    Sailors S, Boats B, Reserves R
        AND (B.color= 'red' OR B.color= 'green')

\{ t(sid) | \exists s \in \text{Sailors} (t.sid = s.sid \land \\
\exists r \in \text{Reserves} (r.sid = s.sid \land \\
\exists b \in \text{Boats} (b.bid = r.bid \land \\
(\text{b.color = 'red'} \lor \text{b.color = 'green'}))))) \}
```

---

**Find sid’s of sailors who’ve reserved a red and a green boat**

- **INTERSECT**: Can be used to compute the intersection of any two _union-compatible_ sets of tuples.
- Included in the SQL/92 standard, but not in all systems (incl. MySQL).
- Contrast symmetry of the UNION and INTERSECT queries with how much the other versions differ.

```
SELECT  S.sid
FROM    Sailors S, Boats B1, Reserves R1,
        Boats B2, Reserves R2
WHERE   S.sid=R1.sid AND R1.bid=B1.bid
        AND S.sid=R2.sid AND R2.bid=B2.bid
        AND (B1.color= 'red' AND B2.color= 'green')
```

```
SELECT  S.sid
FROM    Sailors S, Boats B, Reserves R
        AND B.color= 'red'
INTERSECT
SELECT  S.sid
FROM    Sailors S, Boats B, Reserves R
        AND B.color= 'green'
```
Nested Queries

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

\[
\begin{align*}
\text{SELECT} & \quad \text{sname} \\
\text{FROM} & \quad \text{Sailors S} \\
\text{WHERE} & \quad \text{s.sid} \ \text{IN} \ \left( \text{SELECT} \ \text{r.sid} \\
& \quad \quad \text{FROM} \ \text{Reserves R} \\
& \quad \quad \text{WHERE} \ \text{r.bid}=103 \right)
\end{align*}
\]

- A very powerful feature of SQL: a WHERE clause can itself contain an SQL query! (Actually, so can FROM and HAVING clauses!!)
- To find sailors who’ve not reserved #103, use NOT IN.
- To understand semantics (including cardinality) of nested queries, think **nested loops** evaluation: *For each Sailors tuple, check qualification by computing subquery.*

Nested Queries with Correlation

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

\[
\begin{align*}
\text{SELECT} & \quad \text{sname} \\
\text{FROM} & \quad \text{Sailors S} \\
\text{WHERE} & \quad \text{exists} \left( \text{SELECT} \ * \\
& \quad \quad \text{FROM} \ \text{Reserves R} \\
& \quad \quad \text{WHERE} \ \text{r.bid}=103 \ \text{AND} \ \text{s.sid}=\text{r.sid} \right)
\end{align*}
\]

- **EXISTS** is another set comparison operator, like IN.
- Illustrates why, in general, subquery must be re-computed for each Sailors tuple (conceptually).
  
  **NOTE:** Recall that there was a join way to express this query, too. Relational query optimizers will try to **unnest** queries into joins when possible to avoid nested loop query evaluation plans.
More on Set-Comparison Operators

- We’ve already seen IN and EXISTS. Can also use NOT IN and NOT EXISTS.
- Also available: op ANY, op ALL (for ops: >, <, =, ≥, ≤, ≠ )
- Find sailors whose rating is greater than that of some sailor called Horatio:
  
  ```sql
  SELECT *
  FROM Sailors S
  WHERE S.rating > ANY (SELECT S2.rating
                          FROM Sailors S2
                          WHERE S2.sname = 'Horatio')
  ```

Rewriting INTERSECT Queries Using IN

Find sid’s of sailors who’ve reserved both a red and a green boat:

```sql
SELECT S.sid
FROM Sailors S, Boats B, Reserves R
WHERE S.sid = R.sid AND R.bid = B.bid AND B.color = 'red'
  AND S.sid IN (SELECT S2.sid
                 FROM Sailors S2, Boats B2, Reserves R2
                 WHERE S2.sid = R2.sid AND R2.bid = B2.bid
                 AND B2.color = 'green')
```
**Division, SQL Style**

Find sailors who’ve reserved all boats.

(1) `SELECT S.sname
    FROM Sailors S
    WHERE NOT EXISTS
        ((SELECT B.bid
            FROM Boats B)
        EXCEPT
        (SELECT R.bid
            FROM Reserves R
            WHERE R.sid=S.sid))`

Sailors S such that...

the set of all Boat ids...

minus...

this Sailor’s reserved Boat ids...

is empty!

(2) `SELECT S.sname
    FROM Sailors S
    WHERE NOT EXISTS
        (SELECT B.bid
            FROM Boats B)
        EXCEPT
        (SELECT R.bid
            FROM Reserves R
            WHERE R.bid=B.bid
            AND R.sid=S.sid))`

Sailors S such that...

there is no boat B without...

a Reserves tuple showing S reserved B

**Division in SQL (cont.)**

Find sailors who’ve reserved all boats.

Let’s do it the hard(er) way, i.e., without EXCEPT:

(1) `SELECT S.sname
    FROM Sailors S
    WHERE NOT EXISTS
        ((SELECT B.bid
            FROM Boats B)
        EXCEPT
        (SELECT R.bid
            FROM Reserves R
            WHERE R.sid=S.sid))`

(2) `SELECT S.sname
    FROM Sailors S
    WHERE NOT EXISTS
        (SELECT B.bid
            FROM Boats B)
        EXCEPT
        (SELECT R.bid
            FROM Reserves R
            WHERE R.bid=B.bid
            AND R.sid=S.sid))`
Aggregate Operators

- Significant extension of relational algebra.

\[
\begin{align*}
\text{SELECT} & \quad \text{COUNT} (*) \\
& \quad \text{FROM} \quad \text{Sailors} \ S \\
\text{SELECT} & \quad \text{AVG} (\text{S.age}) \\
& \quad \text{FROM} \quad \text{Sailors} \ S \\
& \quad \text{WHERE} \quad \text{S.rating} = 10 \\
\text{SELECT} & \quad \text{COUNT} (\text{DISTINCT} \ S.\text{rating}) \\
& \quad \text{FROM} \quad \text{Sailors} \ S \\
& \quad \text{WHERE} \quad \text{S.sname} = \text{‘Bob’} \\
\end{align*}
\]

Find name and age of the oldest sailor(s)

- The first query is illegal! (We’ll look into the reason a bit later, when we discuss GROUP BY.)

- The third query is equivalent to the second one, and allowed in the SQL/92 standard, but not supported in all systems.

\[
\begin{align*}
\text{SELECT} & \quad \text{S.sname, MAX} (\text{S.age}) \\
& \quad \text{FROM} \quad \text{Sailors} \ S \\
\text{SELECT} & \quad \text{S.sname, S.age} \\
& \quad \text{FROM} \quad \text{Sailors} \ S \\
& \quad \text{WHERE} \quad \text{S.age} = \\
& \quad \quad (\text{SELECT} \quad \text{MAX} (\text{age}) \\
& \quad \quad \quad \text{FROM} \quad \text{Sailors}) \\
\text{SELECT} & \quad \text{S.sname, S.age} \\
& \quad \text{FROM} \quad \text{Sailors} \ S \\
& \quad \text{WHERE} \quad (\text{SELECT} \quad \text{MAX} (\text{S2.age}) \\
& \quad \quad \text{FROM} \quad \text{Sailors} \ S2) \\
& \quad = \text{S.age}
\end{align*}
\]
Motivation for Grouping

- So far, we’ve applied aggregate operators to all (qualifying) tuples. Sometimes, we want to apply them to each of several groups of tuples.
- Consider: Find the age of the youngest sailor for each rating level.
  - In general, we don’t know how many rating levels exist, and what the rating values for these levels are!
  - Suppose we know that rating values go from 1 to 10; we can write 10 queries that look like this (√):

\[
\text{SELECT MIN (S.age) FROM Sailors S WHERE S.rating = i}
\]

For \(i = 1, 2, \ldots, 10\):

Queries With GROUP BY and HAVING

- The target-list contains (i) attribute names and (ii) terms with aggregate operations (e.g., \(\text{MIN (S.age)}\)).
  - The attribute list (i) must be a subset of grouping-list. Intuitively, each answer tuple corresponds to a group, and these attributes must have a single value per group. (A group is a set of tuples that have the same value for all attributes in grouping-list.)
**Conceptual Evaluation**

- The cross-product of *relation-list* is computed, tuples that fail the *qualification* are discarded, ‘unnecessary’ fields are deleted, and the remaining tuples are partitioned into groups by the value of attributes in *grouping-list*.

- A *group-qualification* (HAVING) is then applied to eliminate some groups. Expressions in *group-qualification* must also have a **single value per group**!
  - In effect, an attribute in *group-qualification* that is not an argument of an aggregate op must appear in *grouping-list*.
  (Note: SQL does not consider primary key semantics here.)

- One answer tuple is generated per qualifying group.

---

**Find age of the youngest sailor with age \( \geq 18 \)**

for each rating with at least 2 such sailors.

\[
\text{SELECT S.rating, } \text{MIN(S.age)} \\
\text{AS minage} \\
\text{FROM Sailors S} \\
\text{WHERE S.age} \geq 18 \\
\text{GROUP BY S.rating} \\
\text{HAVING COUNT (*)} \geq 2
\]

**Sailors instance:**

<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>29</td>
<td>brutus</td>
<td>1</td>
<td>33.0</td>
</tr>
<tr>
<td>31</td>
<td>lubber</td>
<td>8</td>
<td>55.5</td>
</tr>
<tr>
<td>32</td>
<td>andy</td>
<td>8</td>
<td>25.5</td>
</tr>
<tr>
<td>58</td>
<td>rusty</td>
<td>10</td>
<td>35.0</td>
</tr>
<tr>
<td>64</td>
<td>horatio</td>
<td>7</td>
<td>35.0</td>
</tr>
<tr>
<td>71</td>
<td>zorba</td>
<td>10</td>
<td>16.0</td>
</tr>
<tr>
<td>74</td>
<td>horatio</td>
<td>9</td>
<td>35.0</td>
</tr>
<tr>
<td>85</td>
<td>art</td>
<td>3</td>
<td>25.5</td>
</tr>
<tr>
<td>95</td>
<td>bob</td>
<td>3</td>
<td>63.5</td>
</tr>
<tr>
<td>96</td>
<td>frodo</td>
<td>3</td>
<td>25.5</td>
</tr>
</tbody>
</table>

**Answer relation:**

<table>
<thead>
<tr>
<th>rating</th>
<th>minage</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>25.5</td>
</tr>
<tr>
<td>7</td>
<td>35.0</td>
</tr>
<tr>
<td>8</td>
<td>25.5</td>
</tr>
</tbody>
</table>
To Be Continued... 😊