Announcements

- Midterm grading is almost done!
  - All exams essentially graded
  - Final checking phase happening now
  - Back by Wednesday (if not before)
  - Average grade will be in the 80’s (nice! 😊)
- HW#5 is underway (due on Thursday)
  - First of 2-3 related HW assignments
  - Today we’ll continue with SQL…!
Example Data in MySQL

### Sailors

<table>
<thead>
<tr>
<th>sid</th>
<th>sid</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>3</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>1998-09-09</td>
</tr>
<tr>
<td>74</td>
<td>103</td>
<td>1998-09-08</td>
</tr>
<tr>
<td>NULL</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 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.

**SQL Queries**

SELECT S.sid
FROM Sailors S, Boats B1, Reserves R1, Boats B2, Reserves R2
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:

```sql
SELECT S.sname
FROM Sailors S
WHERE S.sid IN (SELECT R.sid
                   FROM Reserves R
                   WHERE R.bid=103)
```

- A very powerful feature of SQL: a WHERE clause can itself contain an SQL query! (Actually, so can SQL’s 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:

```sql
SELECT S.sname
FROM Sailors S
WHERE EXISTS (SELECT *
               FROM Reserves R
               WHERE R.bid=103 AND S.sid=R.sid)
```

- `EXISTS` is another set comparison operator, like `IN`.
- Illustrates why, in general, subquery must be recomputed 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’)
  ```

  So let’s try …
  … running w/ANY on MySQL
  … running w/ALL on MySQL

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’)
```

- Similarly, EXCEPT queries can be re-written using NOT IN.
- This is what you’ll need to do when using MySQL (but you can play with RelaX for the other set ops).
**Division, SQL Style**

Find sailors who’ve reserved all boats.

1. \[
\text{SELECT S.sname} \\
\text{FROM Sailors S} \\
\text{WHERE NOT EXISTS} \\
((\text{SELECT B.bid} \\
\text{FROM Boats B}) \\
\text{EXCEPT} \\
(\text{SELECT R.bid} \\
\text{FROM Reserves R} \\
\text{WHERE R.sid=S.sid}))
\]

This Sailor’s unreserved Boat ids...

Sailors S such that...

the set of all Boat ids...

minus...

this Sailor’s reserved Boat ids...

is empty!

2. \[
\text{SELECT S.sname} \\
\text{FROM Sailors S} \\
\text{WHERE NOT EXISTS} \\
(\text{SELECT B.bid} \\
\text{FROM Boats B}) \\
\text{WHERE NOT EXISTS} \\
(\text{SELECT R.bid} \\
\text{FROM Reserves R} \\
\text{WHERE R.bid=B.bid} \\
\text{AND R.sid=S.sid}))
\]

This way is not that non-easy to understand – right...? (☹️)

**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 Sailors S such that ...

   there is no boat B without ...

   a Reserves tuple saying that S reserved B
**Ordering and/or Limiting Query Results**

*Find the ratings, ids, names, and ages of the three best sailors*

```
SELECT S.rating, S.sid, S.sname, S.age
FROM Sailors S
ORDER BY S.rating DESC
LIMIT 3
```

- The general syntax for this:

```
SELECT [DISTINCT] expressions
FROM tables
[WHERE condition]
.....
[ORDER BY expression [ ASC | DESC ]]
LIMIT number_rows [ OFFSET offset_value ];
```

**Aggregate Operators**

- Significant extension of the relational algebra.

```
SELECT COUNT (*)
FROM Sailors S

SELECT AVG (S.age)
FROM Sailors S
WHERE S.rating = 10

SELECT COUNT (DISTINCT S.rating)
FROM Sailors S
WHERE S.sname = 'Bob'

SELECT COUNT ( [DISTINCT] A)
SUM ( [DISTINCT] A)
AVG ( [DISTINCT] A)
MAX (A)
MIN (A)
```

```
SELECT S.sname
FROM Sailors S
WHERE S.rating = (SELECT MAX(S2.rating)
FROM Sailors S2)
```

```
SELECT AVG (DISTINCT S.age)
FROM Sailors S
WHERE S.rating = 10
```
Find name and age of the oldest sailor(s)

- That first try is illegal! (We’ll see why shortly, when we do GROUP BY.)

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

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 (©):

    For $i = 1, 2, \ldots, 10$:
    
    ```sql
    SELECT MIN (S.age) 
    FROM Sailors S 
    WHERE S.rating = i 
    ```