Introduction to Data Management

Lecture #14
(Relational Languages IV)

Instructor: Mike Carey
mjcarey@ics.uci.edu

It’s time again for....

Friday Nights
With Databases

Brought to you by…
Announcements

- HW#4 is in flight...!
  - Due Monday (RelaX this weekend 😊)
  - Finished with the relational calculus
  - Our SQL adventure has finally begun
- HW#5 due out Monday ("Monday mode" for now)
  - First of a series of SQL-based HW assignments
  - Critical that you resolve any MySQL issues! (Take your machine to discussion section, post questions on Piazza, whatever it takes – else you won’t survive…!)
  - **We now have a project logo... 😊**

Example Data in MySQL

<table>
<thead>
<tr>
<th>Sailors</th>
<th>Reserves</th>
<th>Boats</th>
</tr>
</thead>
<tbody>
<tr>
<td>sid</td>
<td>bname</td>
<td>color</td>
</tr>
<tr>
<td>22</td>
<td>101</td>
<td>blue</td>
</tr>
<tr>
<td>22</td>
<td>102</td>
<td>red</td>
</tr>
<tr>
<td>22</td>
<td>103</td>
<td>red</td>
</tr>
<tr>
<td>22</td>
<td>104</td>
<td>green</td>
</tr>
<tr>
<td>31</td>
<td>102</td>
<td>red</td>
</tr>
<tr>
<td>31</td>
<td>103</td>
<td>green</td>
</tr>
<tr>
<td>31</td>
<td>104</td>
<td>blue</td>
</tr>
<tr>
<td>64</td>
<td>101</td>
<td>blue</td>
</tr>
<tr>
<td>64</td>
<td>102</td>
<td>red</td>
</tr>
<tr>
<td>64</td>
<td>103</td>
<td>green</td>
</tr>
<tr>
<td>74</td>
<td>103</td>
<td>blue</td>
</tr>
<tr>
<td>1</td>
<td>101</td>
<td>blue</td>
</tr>
<tr>
<td>1</td>
<td>102</td>
<td>red</td>
</tr>
<tr>
<td>1</td>
<td>103</td>
<td>green</td>
</tr>
<tr>
<td>1</td>
<td>104</td>
<td>blue</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Boats</th>
<th>Sailors</th>
<th>Reserves</th>
</tr>
</thead>
<tbody>
<tr>
<td>bid</td>
<td>bname</td>
<td>date</td>
</tr>
<tr>
<td>101</td>
<td>Interlake</td>
<td>1998-10-10</td>
</tr>
<tr>
<td>102</td>
<td>Interlake</td>
<td>1998-10-08</td>
</tr>
<tr>
<td>103</td>
<td>Clipper</td>
<td>1998-10-07</td>
</tr>
<tr>
<td>104</td>
<td>Marine</td>
<td>1998-11-10</td>
</tr>
<tr>
<td>105</td>
<td>Interlake</td>
<td>1998-09-05</td>
</tr>
<tr>
<td>106</td>
<td>Interlake</td>
<td>1998-09-08</td>
</tr>
<tr>
<td>107</td>
<td>Interlake</td>
<td>1998-09-09</td>
</tr>
<tr>
<td>108</td>
<td>Interlake</td>
<td>2001-01-11</td>
</tr>
<tr>
<td>109</td>
<td>Interlake</td>
<td>2002-02-02</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**
  (What would we get if we replaced **UNION** by **EXCEPT**?)

[Note: MySQL vs. RelaX – and why?]

---

**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) | ∃s ∈ Sailors (t.sid = s.sid ∧
  ∃r ∈ Reserves (r.sid = s.sid ∧
  ∃b ∈ Boats (b.bid = r.bid ∧
  (b.color = 'red' V 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:

```
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:

```
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:

```
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. 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)))

(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!)

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke
**Division in SQL (cont.)**

Find sailors who’ve reserved all boats.

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

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

**(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…? 😅

---

**Ordering and/or Limiting Query Results**

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

\[
\text{SELECT S.rating, S.sid, S.sname, S.age} \\
\text{FROM Sailors S} \\
\text{ORDER BY S.rating DESC} \\
\text{LIMIT 3}
\]

- The general syntax for this:

\[
\text{SELECT [DISTINCT] expressions} \\
\text{FROM tables} \\
\text{[WHERE condition]} \\
\text{...} \\
\text{[ORDER BY expression [ ASC | DESC ]]} \\
\text{LIMIT number_rows [ OFFSET offset_value ]};
\]
Aggregate Operators

- Significant extension of the relational algebra.

<table>
<thead>
<tr>
<th>COUNT(*)</th>
<th>COUNT([DISTINCT] A)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUM([DISTINCT] A)</td>
<td>AVG([DISTINCT] A)</td>
</tr>
<tr>
<td>MAX(A)</td>
<td>MIN(A)</td>
</tr>
</tbody>
</table>

**Example Queries**

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

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

Find name and age of the oldest sailor(s)

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

- **Nit:** The third version is equivalent to the second one, and is 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 (∈):
    
    \[
    \text{SELECT } \min(S.\text{age}) \text{ FROM Sailors S WHERE } S.\text{rating} = i
    \]
    
    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., \( \min(S.\text{age}) \)).
  - 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 doesn’t 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.

```
SELECT S.rating, MIN(S.age) AS minage
FROM Sailors S
WHERE S.age >= 18
GROUP BY S.rating
HAVING COUNT(*) >= 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>
Find age of the youngest sailor with age $\geq 18$ for each rating with at least 2 such sailors.