Database Applications (15-415)

SQL-Part II
Lecture 8, February 3, 2015

Mohammad Hammoud
Today...

- **Last Session:**
  - Standard Query Language (SQL)- Part I

- **Today’s Session:**
  - Standard Query Language (SQL)- Part II

- **Announcements:**
  - PS2 is due on Sunday Feb 08, 2015 by midnight
  - Quiz I is on Thursday Feb 12, 2015 (all topics covered so far are included)
  - Project I is now posted. It is due on Tuesday Feb 17 by midnight
Outline

- Nested Queries
- Insertions, Deletions and Updates
- NULL values and Join Variants
A Join Query

- Find the names of sailors who have reserved boat 101

<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>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Sid</th>
<th>Bid</th>
<th>Day</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>101</td>
<td>10/10/2013</td>
</tr>
<tr>
<td>22</td>
<td>102</td>
<td>10/10/2013</td>
</tr>
</tbody>
</table>

\[
\text{select } S.\text{snname} \\
\text{from } \text{Sailors } S, \text{Reserves } R \\
\text{where } S.\text{sid} = R.\text{sid} \\
\text{and } R.\text{bid} = 101
\]
Nested Queries

- Find the names of sailors who have reserved boat 101

<table>
<thead>
<tr>
<th>Sailors</th>
<th>Reserves</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sid</td>
<td>Sname</td>
</tr>
<tr>
<td>22</td>
<td>Dustin</td>
</tr>
<tr>
<td>29</td>
<td>Brutus</td>
</tr>
</tbody>
</table>

```sql
SELECT S.sname
FROM Sailors S
WHERE S.sid IN (SELECT R.sid
                 FROM Reserves R
                 WHERE R.bid = 101)
```

OR...

IN compares a value with a set of values
Nested Queries

- Find the names of sailors who have not reserved boat 101

<table>
<thead>
<tr>
<th>Sailors</th>
<th>Reserves</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sid</td>
<td>Sname</td>
</tr>
<tr>
<td>22</td>
<td>Dustin</td>
</tr>
<tr>
<td>29</td>
<td>Brutus</td>
</tr>
</tbody>
</table>

```
SELECT S.sname
FROM Sailors S
WHERE S.sid NOT IN (SELECT R.sid
FROM Reserves R
WHERE R.bid=101)
```
Deeply Nested Queries

- Find the names of sailors who have reserved a red boat

<table>
<thead>
<tr>
<th>Sailors</th>
<th>Reserves</th>
<th>Boats</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sid</td>
<td>Sname</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>
</tbody>
</table>

SQL:

```
SELECT S.sname
FROM Sailors S
WHERE S.sid IN (SELECT R.sid
    FROM Reserves R
    WHERE R.bid IN (SELECT B.bid
        FROM Boats B
        WHERE B.color = 'red'))
```

In principle, queries with very deeply nested structures are possible!
### 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>

### Reserves instance:

<table>
<thead>
<tr>
<th>sid</th>
<th>bid</th>
<th>day</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>101</td>
<td>10/10/98</td>
</tr>
<tr>
<td>22</td>
<td>102</td>
<td>10/10/98</td>
</tr>
<tr>
<td>22</td>
<td>103</td>
<td>10/8/98</td>
</tr>
<tr>
<td>22</td>
<td>104</td>
<td>10/7/98</td>
</tr>
<tr>
<td>31</td>
<td>102</td>
<td>11/10/98</td>
</tr>
<tr>
<td>31</td>
<td>103</td>
<td>11/6/98</td>
</tr>
<tr>
<td>31</td>
<td>104</td>
<td>11/12/98</td>
</tr>
<tr>
<td>64</td>
<td>101</td>
<td>9/5/98</td>
</tr>
<tr>
<td>64</td>
<td>102</td>
<td>9/8/98</td>
</tr>
<tr>
<td>74</td>
<td>103</td>
<td>9/8/98</td>
</tr>
</tbody>
</table>

### Boats instance:

<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>

```sql
SELECT S.sname FROM Sailors S
WHERE S.sid IN (SELECT R.sid FROM Reserves R
WHERE R.bid IN (SELECT B.bid FROM Boats B
WHERE B.color = 'red'))
```
Deeply Nested Queries

- Find the names of sailors who have **not** reserved a red boat

<table>
<thead>
<tr>
<th>Sailors</th>
<th>Reserves</th>
<th>Boats</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sid</td>
<td>Sname</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>
</tbody>
</table>

```sql
SELECT S.sname
FROM Sailors S
WHERE S.sid NOT IN (SELECT R.sid
FROM Reserves R
WHERE R.bid IN (SELECT B.bid
FROM Boats B
WHERE B.color = 'red'))
```
This returns the names of sailors who have not reserved a red boat!
### 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>

### Reserves instance:

<table>
<thead>
<tr>
<th>sid</th>
<th>bid</th>
<th>day</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>101</td>
<td>10/10/98</td>
</tr>
<tr>
<td>22</td>
<td>102</td>
<td>10/10/98</td>
</tr>
<tr>
<td>22</td>
<td>103</td>
<td>10/8/98</td>
</tr>
<tr>
<td>22</td>
<td>104</td>
<td>10/7/98</td>
</tr>
<tr>
<td>31</td>
<td>102</td>
<td>11/10/98</td>
</tr>
<tr>
<td>31</td>
<td>103</td>
<td>11/6/98</td>
</tr>
<tr>
<td>31</td>
<td>104</td>
<td>11/12/98</td>
</tr>
<tr>
<td>64</td>
<td>101</td>
<td>9/5/98</td>
</tr>
<tr>
<td>64</td>
<td>102</td>
<td>9/8/98</td>
</tr>
<tr>
<td>74</td>
<td>103</td>
<td>9/8/98</td>
</tr>
</tbody>
</table>

### Boats instance:

<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>

**SELECT S.sname FROM Sailors S WHERE S.sid IN (SELECT R.sid FROM Reserves R WHERE R.bid NOT IN (SELECT B.bid FROM Boats B WHERE B.color = 'red'))**

This returns the names of sailors who have reserved a boat that is **not** red.

The previous one returns the names of sailors who have **not** reserved a red boat!
This returns the names of sailors who have not reserved a boat that is not red!

As such, it returns names of sailors who have reserved only red boats (if any)
**Correlated Nested Queries**

- Find the names of sailors who have reserved boat 101

<table>
<thead>
<tr>
<th>Sailors</th>
<th>Reserves</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sid</td>
<td>Bid</td>
</tr>
<tr>
<td>22</td>
<td>101</td>
</tr>
<tr>
<td>22</td>
<td>102</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Sailors</th>
<th>Reserves</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sid</td>
<td>Bid</td>
</tr>
<tr>
<td>22</td>
<td>101</td>
</tr>
<tr>
<td>22</td>
<td>102</td>
</tr>
</tbody>
</table>

- A correlation allows us to test whether a set is “nonempty”

- Compares a value with a set of values

```sql
SELECT s.sname
FROM Sailors s
WHERE s.sid IN (SELECT r.sid
                 FROM Reserves r
                 WHERE r.bid=101)
```

```sql
SELECT s.sname
FROM Sailors s
WHERE EXISTS (SELECT *
               FROM Reserves r
               WHERE r.bid=101
               AND r.sid = s.sid)
```
Correlated Nested Queries

- Find the names of sailors who have not reserved boat 101

<table>
<thead>
<tr>
<th>Sailors</th>
<th>Reserves</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sid</td>
<td>Sname</td>
</tr>
<tr>
<td>22</td>
<td>Dustin</td>
</tr>
<tr>
<td>29</td>
<td>Brutus</td>
</tr>
</tbody>
</table>

SELECT S.sname
FROM Sailors S
WHERE S.sid NOT IN (SELECT R.sid
FROM Reserves R
WHERE R.bid=101)

SELECT S.sname
FROM Sailors S
WHERE NOT EXISTS (SELECT *
FROM Reserves R
WHERE R.bid=101 AND R.sid = S.sid)
Nested Queries with Set-Comparison Operators

- Find sailors whose rating is better than some sailor called Dustin

<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>
</tbody>
</table>

Q: What if there were no sailors called Dustin?

A: An empty set is returned!
Nested Queries with Set-Comparison Operators

- Find sailors whose rating is better than every sailor called Dustin

<table>
<thead>
<tr>
<th>Sailors</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sid</td>
</tr>
<tr>
<td>-------</td>
</tr>
<tr>
<td>22</td>
</tr>
<tr>
<td>29</td>
</tr>
</tbody>
</table>

SELECT S.sname
FROM Sailors S
WHERE S.rating > ALL (SELECT S2. rating 
FROM Sailors S2 
WHERE S2.name = ‘Dustin’)

Q: What if there were no sailors called Dustin?

A: The names of all sailors will be returned! (Be Careful)
Nested Queries with Set-Comparison Operators

- Find sailors with the highest sid

<table>
<thead>
<tr>
<th>Sailors</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sid</td>
</tr>
<tr>
<td>------</td>
</tr>
<tr>
<td>22</td>
</tr>
<tr>
<td>29</td>
</tr>
</tbody>
</table>

SELECT *  
FROM  Sailors S  
WHERE  S.sid  

*is greater than every other sid*
Nested Queries with Set-Comparison Operators

- Find sailors with the highest sid

<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>
</tbody>
</table>

```
SELECT *
FROM Sailors S
WHERE S.sid is greater than every
(SELECT S2.sid
FROM Sailors S2)
```
Nested Queries with Set-Comparison Operators

- Find sailors with the highest sid

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Sid</td>
<td>Sname</td>
<td>Rating</td>
<td>age</td>
</tr>
<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>
</tbody>
</table>

SELECT *  
FROM Sailors S  
WHERE S.sid > ALL  
(SELECT S2.sid  
FROM Sailors S2)
Nested Queries with Set-Comparison Operators

- Find sailors with the highest sid

<table>
<thead>
<tr>
<th>Sailors</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sid</td>
</tr>
<tr>
<td>22</td>
</tr>
<tr>
<td>29</td>
</tr>
</tbody>
</table>

```sql
SELECT * FROM Sailors S
WHERE S.sid >= ALL (SELECT S2.sid FROM Sailors S2)
```

Now Correct!
Nested Queries with Set-Comparison Operators

- Find sailors with the highest sid- *without nested subquery*

<table>
<thead>
<tr>
<th>Sailors</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sid</td>
</tr>
<tr>
<td>-------</td>
</tr>
<tr>
<td>22</td>
</tr>
<tr>
<td>29</td>
</tr>
</tbody>
</table>

```
SELECT *
FROM  Sailors S1, Sailors S2
WHERE  S1.sid > S2.sid
```

Q: What does this give?
Nested Queries with Set-Comparison Operators

- Find sailors with the highest sid- **without nested subquery**

<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>
</tbody>
</table>

$S1 \times S2$

<table>
<thead>
<tr>
<th>S1.Sid</th>
<th>S2.sid</th>
<th>....</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>22</td>
<td>....</td>
</tr>
<tr>
<td>22</td>
<td>29</td>
<td>....</td>
</tr>
<tr>
<td>29</td>
<td>22</td>
<td></td>
</tr>
<tr>
<td>29</td>
<td>29</td>
<td></td>
</tr>
</tbody>
</table>

$S1.sid > S2.sid$
Nested Queries with Set-Comparison Operators

- Find sailors with the highest sid - *without nested subquery*

<table>
<thead>
<tr>
<th>Sailors</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Sid</td>
<td>Sname</td>
<td>Rating</td>
<td>age</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>Dustin</td>
<td>7</td>
<td>45.0</td>
<td></td>
</tr>
<tr>
<td>29</td>
<td>Brutus</td>
<td>1</td>
<td>33.0</td>
<td></td>
</tr>
</tbody>
</table>

```
SELECT *
FROM  Sailors S1, Sailors S2
WHERE  S1.sid > S2.sid
```

Q: What does this give?

A: All but the smallest sid!
Nested Queries with Set-Comparison Operators

- Find sailors with the highest sid- *without nested subquery*

<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>
</tbody>
</table>

```sql
SELECT * 
FROM  Sailors S1, Sailors S2 
WHERE  S1.sid < S2.sid
```

Q: What does this give?

A: All but the highest sid!
Nested Queries with Set-Comparison Operators

- Find sailors with the highest sid- *without nested subquery*

<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>
</tbody>
</table>

\[
\text{(SELECT * FROM Sailors) EXCEPT (SELECT S1.sid, S1.sname, S1.rating, S1.age FROM Sailors S1, Sailors S2 WHERE S1.sid < S2.sid)}
\]

Therefore...

\[\text{I.e., ALL} - (\text{ALL} - \text{Highest}) = \text{Highest}\]
Alternative Ways

- Find sailors with the highest sid

<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>
</tbody>
</table>

```sql
(SELECT *
FROM Sailors)
EXCEPT
(SELECT S1.sid, S1.sname, S1.rating, S1.age
FROM Sailors S1, Sailors S2
WHERE S1.sid < S2.sid)
```

**VS.**

```sql
SELECT *
FROM Sailors S
WHERE S.sid >= ALL
(SELECT S2.sid
FROM Sailors S2)
```
Revisit: Another Example

- Find the names of sailors who have reserved both a red and a green boat

\[
(\text{select } S.\text{sname } \text{from Sailors } S, \text{ Reserves } R, \text{ Boats } B \\
\text{where } S.\text{sid} = R.\text{sid} \text{ and } R.\text{bid} = B.\text{bid} \text{ and } B.\text{color} = 'green')
\]

\[
\text{intersect}
\]

\[
(\text{select } S2.\text{sname } \text{from Sailors } S2, \text{ Reserves } R2, \text{ Boats } B2 \\
\text{where } S2.\text{sid} = R2.\text{sid} \text{ and } R2.\text{bid} = B2.\text{bid} \text{ and } B2.\text{color} = 'red')
\]

The query contains a “subtle bug” which arises because we are using \textit{sname} to identify Sailors, and “sname” is not a key for Sailors!

If we want to compute the names of such Sailors, we would need a NESTED QUERY.
A Correct Way

- Find the names of sailors who have reserved both a red and a green boat

```
(select S.sname from Sailors S, Reserves R, Boats B
where S.sid = R.sid and R.bid = B.bid and B.color = 'green')
AND S.sid IN
(select S2.sid from Sailors S2, Reserves R2, Boats B2
where S2.sid = R2.sid and R2.bid = B2.bid and B2.color = 'red')
```

Similarly, queries using EXCEPT can be re-written using NOT IN
Revisit: Another Example

- Find the name and age of the oldest sailor

```
select S.sname, max(S.age) from Sailors S
```

This query is illegal in SQL- If the “select” clause uses an aggregate function, it must use ONLY aggregate function unless the query contains a “group by” clause!
A Correct Way

- Find the name and age of the oldest sailor

<table>
<thead>
<tr>
<th>Sailors</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sid</td>
</tr>
<tr>
<td>--------</td>
</tr>
<tr>
<td>22</td>
</tr>
<tr>
<td>29</td>
</tr>
</tbody>
</table>

```
SELECT S.sname, S.age
FROM Sailors S
WHERE S.age = (SELECT MAX(S2.age)
FROM Sailors S2)
```
Alternative Ways

- Find the name and age of the oldest sailor

<table>
<thead>
<tr>
<th>Sailors</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sid</td>
</tr>
<tr>
<td>22</td>
</tr>
<tr>
<td>29</td>
</tr>
</tbody>
</table>

**SELECT** S.sname, S.age
**FROM** Sailors S
**WHERE** S.age = (SELECT MAX(S2.age)
**FROM** Sailors S2)  

VS.

**SELECT** S.sname, **MAX**(S.age)
**FROM** Sailors S
**GROUP** **BY** S.sname
Revisit: Another Example

- Find age of the youngest sailor with age $\geq 18$, for each rating level with at least 2 such sailors

<table>
<thead>
<tr>
<th>Sailors</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sid</td>
</tr>
<tr>
<td>22</td>
</tr>
<tr>
<td>29</td>
</tr>
</tbody>
</table>

SELECT S.rating, MIN (S.age) AS minage
FROM Sailors S
WHERE S.age >= 18
GROUP BY S.rating
HAVING COUNT (*) > 1
An Alternative Way

- Find age of the youngest sailor with age ≥ 18, for each rating level with at least 2 such 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>
</tbody>
</table>
```

```sql
SELECT S.rating, MIN(S.age) AS minage
FROM Sailors S
WHERE S.age >= 18
GROUP BY S.rating
HAVING 1 < (SELECT COUNT(*)
            FROM Sailors S2
            WHERE S.rating = S2.rating)
```

The HAVING clause can include subqueries!
Yet Another Way

- Find age of the youngest sailor with age ≥ 18, for each rating level with at least 2 such 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>
</tbody>
</table>

```sql
SELECT Temp.rating, Temp.minage
FROM (SELECT S.rating, MIN(S.age) AS minage,
       COUNT(*) AS ratingcount
       FROM Sailors S
       WHERE S.age >= 18
       GROUP BY S.rating) AS Temp
WHERE Temp.ratingcount > 1
```

OR...

The FROM clause can include subqueries!

Necessary!
Expressing the Division Operator in SQL

- Find the names of sailors who have reserved all boats

<table>
<thead>
<tr>
<th>Sailors</th>
<th>Reserves</th>
<th>Boats</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sid</td>
<td>Sname</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>
</tbody>
</table>

```sql
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))
```
Outline

- Nested Queries
- Insertions, Deletions and Updates
- NULL values and Join Variants
Reminder: Our Mini-U DB

<table>
<thead>
<tr>
<th>STUDENT</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Ssn</td>
<td>Name</td>
<td>Address</td>
</tr>
<tr>
<td>123</td>
<td>smith</td>
<td>main str</td>
</tr>
<tr>
<td>234</td>
<td>jones</td>
<td>QF ave</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CLASS</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>c-id</td>
<td>c-name</td>
<td>units</td>
<td></td>
</tr>
<tr>
<td>15-413</td>
<td>s.e.</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>15-412</td>
<td>o.s.</td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TAKES</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>SSN</td>
<td>c-id</td>
<td>grade</td>
</tr>
<tr>
<td>123</td>
<td>15-413</td>
<td>A</td>
</tr>
<tr>
<td>234</td>
<td>15-413</td>
<td>B</td>
</tr>
</tbody>
</table>
Revisit: Insertions

```sql
insert into student(ssn, name, address)
values (123, 'smith', 'main')
```

OR...

```sql
insert into student
values (123, 'smith', 'main')
```
Bulk Insertions

- How to insert, say, a table of “foreign-student”, in *bulk*?

```
insert into student
    select ssn, name, address
from foreign-student
```
Revisit: Deletions

- Delete the record of ‘smith’

```
delete from student
where name=‘smith’
```

Be careful - it deletes ALL the ‘smith’s!
Revisit: Updates

- Update the grade to ‘A’ for ssn=123 and course 15-415

```
update takes
set grade='A'
where ssn = 123 and c-id= ‘15-415’
```
Updating Views

- Consider the following view:

  ```sql
  create view db-takes as
  (select * from takes where c-id="15-415")
  ```

- What if c-id is modified to ‘15-440’?

- What if c-id is deleted?

A Rule of thumb: A command that affects a row in the view affects all corresponding rows in underlying tables!

View updates are tricky - typically, we can only update views that have no joins, nor aggregates!
Outline

Nested Queries

Insertions, Deletions and Updates

NULL values and Join Variants
NULL Values

- Column values can be *unknown* (e.g., a sailor may not yet have a rating assigned)

- Column values may be *inapplicable* (e.g., a maiden-name column for men!)

- The **NULL** value can be used in such situations

- However, the NULL value complicates many issues!
  - Using NULL with aggregate operations
    - `COUNT(*)` handles NULL values like any other values
    - `SUM`, `AVG`, `MIN`, and `MAX` discard NULL values
  - Comparing NULL values to valid values
  - Comparing NULL values to NULL values
Comparing Values In the Presence of NULL

- Considering a row with rating = NULL and age = 20; what will be the result of comparing it with the following rows?
  - Rating = 8 OR age < 40 → TRUE
  - Rating = 8 AND age < 40 → unknown

- In general:
  - NOT unknown → unknown
  - True OR unknown → True
  - False OR unknown → unknown
  - False AND unknown → False
  - True AND unknown → unknown
  - Unknown [AND|OR] unknown → unknown

In the context of *duplicates*, the comparison of two NULL values is implicitly treated as TRUE (Anomaly!)
Comparing Values In the Presence of NULL

- Considering a row with rating = NULL and age = 20; what will be the result of comparing it with the following rows?
  - Rating = 8 OR age < 40 \(\rightarrow\) TRUE
  - Rating = 8 AND age < 40 \(\rightarrow\) unknown

- In general:
  - NOT unknown \(\rightarrow\) unknown
  - True OR unknown \(\rightarrow\) True
  - False OR unknown \(\rightarrow\) unknown
  - False AND unknown \(\rightarrow\) False
  - True AND unknown \(\rightarrow\) unknown
  - Unknown [AND|OR|=] unknown \(\rightarrow\) unknown

*Three-Valued Logic!*
Inner Join

- Tuples of a relation that do not match some rows in another relation (according to a join condition $c$) do not appear in the result
  - Such a join is referred to as “Inner Join” (so far, all inner joins)

```
select ssn, c-name
from takes, class
where takes.c-id = class.c-id
```

Equivalently:

```
select ssn, c-name
from takes join class on takes.c-id = class.c-id
```
### Inner Join

- Find all SSN(s) taking course s.e.

<table>
<thead>
<tr>
<th>SSN</th>
<th>c-id</th>
<th>grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>15-413</td>
<td>A</td>
</tr>
<tr>
<td>234</td>
<td>15-413</td>
<td>B</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SSN</th>
<th>c-name</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>s.e</td>
</tr>
<tr>
<td>234</td>
<td>s.e</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>c-id</th>
<th>c-name</th>
<th>units</th>
</tr>
</thead>
<tbody>
<tr>
<td>15-413</td>
<td>s.e.</td>
<td>2</td>
</tr>
<tr>
<td>15-412</td>
<td>o.s.</td>
<td>2</td>
</tr>
</tbody>
</table>

**o.s.: gone!**
Outer Join

- But, tuples of a relation that do not match some rows in another relation (according to a join condition \( c \)) can still appear exactly once in the result
  - Such a join is referred to as "Outer Join"
  - Result columns will be assigned NULL values

```sql
select ssn, c-name
from takes outer join class
on takes.c-id=class.c-id
```
Outer Join

- Find all SSN(s) taking course s.e.

<table>
<thead>
<tr>
<th>TAKES</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>SSN</td>
<td>c-id</td>
</tr>
<tr>
<td>123</td>
<td>15-413</td>
</tr>
<tr>
<td>234</td>
<td>15-413</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CLASS</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>c-id</td>
<td>c-name</td>
</tr>
<tr>
<td>15-413</td>
<td>s.e.</td>
</tr>
<tr>
<td>15-412</td>
<td>o.s.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SSN</th>
<th>c-name</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>s.e.</td>
</tr>
<tr>
<td>234</td>
<td>s.e.</td>
</tr>
<tr>
<td>null</td>
<td>o.s.</td>
</tr>
</tbody>
</table>
Joins

- In general:

```
select [column list]
from  table_name
  [inner | {left | right | full} outer] join
  table_name
on qualification_list
Where ...
```
Summary

- Nested Queries
  - IN, NOT IN, EXISTS, NOT EXISTS, \( op \) ANY and \( op \) ALL where \( op \in \{<, \leq, =, >, angle, \geq, >\} \)
  - Re-writing INTERSECT using IN
  - Re-writing EXCEPT using NOT IN
  - Expressing the division operation using NOT EXISTS and EXCEPT (there are other ways to achieve that!)

- Other DML commands: INSERT (including bulk insertions), DELETE and UPDATE (for tables and views)

- Null values and inner vs. outer Joins
Next Class

SQL- Part III & Storing Data: Disks and Files (if time allows)