### 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>3</td>
<td>25.5</td>
</tr>
<tr>
<td>95</td>
<td>Bob</td>
<td>3</td>
<td>63.5</td>
</tr>
</tbody>
</table>

### Reserves

<table>
<thead>
<tr>
<th>sid</th>
<th>bid</th>
<th>day</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>101</td>
<td>10/10/04</td>
</tr>
<tr>
<td>22</td>
<td>102</td>
<td>10/10/04</td>
</tr>
<tr>
<td>22</td>
<td>103</td>
<td>10/08/04</td>
</tr>
<tr>
<td>22</td>
<td>104</td>
<td>10/07/04</td>
</tr>
<tr>
<td>31</td>
<td>102</td>
<td>11/10/04</td>
</tr>
<tr>
<td>31</td>
<td>103</td>
<td>11/06/04</td>
</tr>
<tr>
<td>31</td>
<td>104</td>
<td>11/12/04</td>
</tr>
<tr>
<td>64</td>
<td>101</td>
<td>09/05/04</td>
</tr>
<tr>
<td>64</td>
<td>102</td>
<td>09/08/04</td>
</tr>
<tr>
<td>74</td>
<td>103</td>
<td>09/08/04</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>
Illustration of INTERSECT… 1

Find names of sailors who’ve reserved a red and a green boat

Intuitively, we would write the SQL query as:

```sql
SELECT  S.sname
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');
```
Illustration of INTERSECT…2

We can also do this using a INTERSECT keyword:

```
SELECT  S.sname
FROM   Sailors S, Boats B, Reserves R
WHERE   S.sid=R.sid AND R.bid=B.bid AND B.color= 'red'
INTERSECT
SELECT  S2.sname
FROM   Sailors S2, Boats B2, Reserves R2
WHERE   S2.sid=R2.sid AND R2.bid=B2.bid AND B2.color= 'green'
```

(Is this correct??)
(Semi-)Correct SQL Query for the Previous Example

SELECT  S.sid
FROM  Sailors S, Boats B, Reserves R
       AND B.color='red'
INTERSECT
SELECT  S2.sid
FROM  Sailors S2, Boats B2, Reserves R2
WHERE  S2.sid=R2.sid AND R2.bid=B2.bid
       AND B2.color='green' ;

(This time we have actually extracted the sids of sailors, and not their names. In fact, we don’t even need Sailors here.)
(But the query asks for the names of the sailors.)
Illustration of EXCEPT

*Find the sids of all sailors who have reserved red boats but not green boats:*

```
SELECT S.sid
FROM Sailors S, Boats B, Reserves R
WHERE S.sid=R.sid AND R.bid=B.bid AND B.color= 'red'
EXCEPT
SELECT S2.sid
FROM Sailors S2, Boats B2, Reserves R2
WHERE S2.sid=R2.sid AND R2.bid=B2.bid AND B2.color= 'green';
```

*Use MINUS instead of EXCEPT in Oracle*
Nested Queries

• A **nested** query is a query that has another query embedded within it; this embedded query is called the **subquery**.

• Subqueries generally occur within the WHERE clause (but can also appear within the FROM and HAVING clauses)

• Nested queries are a very powerful feature of SQL. They help us write short and efficient queries.

(Think of nested **for** loops in C++. Nested queries in SQL are similar)
Nested Query 1

*Find names of sailors who have reserved boat 103*

```sql
SELECT S.sname
FROM Sailors S
WHERE S.sid IN (SELECT R.sid
                 FROM Reserves R
                 WHERE R.bid=103);
```
Nested Query 2

Find names of sailors who have not reserved boat 103

SELECT S.sname
FROM Sailors S
WHERE S.sid NOT IN (SELECT R.sid
FROM Reserves R
WHERE R.bid=103);
Nested Query 3

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

```
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'));
```

What about *Find the names of sailors who have NOT reserved a red boat*?
Revisit a previous query

Find names of sailors who’ve reserved a red and a green boat

SELECT  S.sid
FROM    Sailors S, Boats B, Reserves R
        AND B.color='red'
INTERSECT
SELECT  S2.sid
FROM    Sailors S2, Boats B2, Reserves R2
WHERE   S2.sid=R2.sid AND R2.bid=B2.bid
        AND B2.color='green';
Revisit a previous query

Find names of sailors who’ve reserved a red and a green boat

SELECT  S.sname  
FROM  Sailor S  
WHERE  S.sid IN (SELECT  R.sid  
FROM  Boats B, Reserves R  
WHERE  R.bid=B.bid AND B.color='red'  
INTERSECT  
SELECT  R2.sid  
FROM  Boats B2, Reserves R2  
WHERE  R2.bid=B2.bid AND B2.color='green' );
Correlated Nested Queries… 1

• Thus far, we have seen nested queries where the inner subquery is independent of the outer query.

• We can make the inner subquery depend on the outer query. This is called correlation.
Correlated Nested Queries...2

Find names of sailors who have reserved boat 103

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

Tests whether the set is nonempty

(For finding sailors who have not reserved boat 103, we would use NOT EXISTS)
Correlated Nested Query 2

Find the names of sailors who have reserved ALL boats (DIVISION)

\[ A = \pi_{sid} \left( (\pi_{sid} \text{Sailors}) \times (\pi_{bid} \text{Boats}) \right) - \pi_{sid,bid} \text{Reserves} \]

\[ \pi_{sid} \text{Sailors} - A \]

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));
Correlated Nested Query 2

Alternatively,

*Find the names of sailors who have reserved ALL boats*

```
SELECT S.sname
FROM Sailors S
WHERE NOT EXISTS (SELECT B.bid
FROM Boats B
WHERE NOT EXISTS (SELECT R.bid
FROM Reserves R
WHERE R.bid = B.bid AND R.sid = S.sid ));
```
ANY and ALL operators

*Find sailors whose rating is better than some sailor named Horatio*

SELECT S.sid
FROM Sailors S
WHERE S.rating > ANY (SELECT S2.rating
FROM Sailors S2
WHERE S2.sname='Horatio');
Using ALL operator

Find sailors whose rating is better than every sailor named Horatio

```
SELECT S.sid
FROM Sailors S
WHERE S.rating > ALL(SELECT S2.ratio
                     FROM Sailors S2
                     WHERE S2.sname='Horatio');
```

What if there were no sailor called Horatio?

Note that IN is equivalent to = ANY
NOT IN is equivalent to <> ALL
BETWEEN and AND operators

• The BETWEEN and AND operator selects a range of data between two values.

• These values can be numbers, text, or dates.
**BETWEEN and AND Example**

*Find the names of sailors whose age is between 25 and 35*

```sql
SELECT S.sname
FROM Sailors S
WHERE S.age BETWEEN 25 AND 35;
```
Post Processing

• Processing on the result of an SQL query:
  – Sorting: can sort the tuples in the output by any column (even the ones not appearing in the SELECT clause)
  – Duplicate removal
  – Example:

    ```sql
    SELECT DISTINCT S.sname
    FROM  Sailors S, Reserves R
    WHERE  S.sid=R.sid AND R.bid=103
    ORDER BY S.sid ASC, S.sname DESC;
    ```

• Aggregation operators
Aggregate operators

- What is aggregation?
  - Computing arithmetic expressions, such as **Minimum** or **Maximum**

- The aggregate operators supported by SQL are: COUNT, SUM, AVG, MIN, MAX
Aggregate Operators

- **COUNT(A)**: The number of values in the column A
- **SUM(A)**: The sum of all values in column A
- **AVG(A)**: The average of all values in column A
- **MAX(A)**: The maximum value in column A
- **MIN(A)**: The minimum value in column A

(We can use DISTINCT with COUNT, SUM and AVG to compute only over non-duplicated columns)
Using the COUNT operator

Count the number of sailors

SELECT COUNT (*)
FROM  Sailors S;
Another Aggregate Query

Count the number of different sailor names

SELECT COUNT (DISTINCT S.sname) FROM Sailors S;
Example of SUM operator

*Find the sum of ages of all sailors with a rating of 10*

```sql
SELECT  SUM (S.age)
FROM  Sailors S
WHERE  S.rating=10;
```
Example of AVG operator

Find the average age of all sailors with rating 10

SELECT AVG (S.age) FROM Sailors S WHERE S.rating=10;
Example of MAX operator

Find the name and age of the oldest sailor

SELECT S.sname, MAX(S.age)  
FROM Sailors S;

But this is illegal in SQL!!
Correct SQL Query for MAX

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

```
SELECT S.sname, S.age
FROM Sailors S
WHERE ROWNUM <= 1
ORDER BY S.age DESC;
```