SQL - 1

Week 6
**Basic form of SQL Queries**

<table>
<thead>
<tr>
<th>SELECT</th>
<th>target-list</th>
</tr>
</thead>
<tbody>
<tr>
<td>FROM</td>
<td>relation-list</td>
</tr>
<tr>
<td>WHERE</td>
<td>qualification</td>
</tr>
</tbody>
</table>

- **target-list**  A list of attributes of output relations in relation-list
- **relation-list** A list of relation names (possibly with a range-variable after each name)
  
  e.g. Sailors S, Reserves R
- **qualification**  Comparisons (Attr op const or Attr1 op Attr2, where op is one of <, >, ≤, ≥, =, ≠) combined using AND, OR and NOT.
What’s contained in an SQL Query?

Every SQL Query must have:

- **SELECT** clause: specifies columns to be retained in result
- **FROM** clause: specifies a cross-product of tables

The **WHERE** clause (optional) specifies selection conditions on the tables mentioned in the **FROM** clause.
General SQL Conceptual Evaluation Strategy

• Semantics of an SQL query defined in terms of the following conceptual evaluation strategy:
  – Compute the cross-product of relation-list.
  – Discard resulting tuples if they fail qualifications.
  – Delete attributes that are not in target-list.

• This strategy is probably the least efficient way to compute a query! An optimizer will find more efficient strategies to compute the same answers.
Conceptual Evaluation Strategy

Nested loops evaluation:

Foreach tuple t1 in R1
... 
Foreach tuple tn in Rn
  1. Substitute the attribute names in the qualification part with values from t1, …, tn
  2. If the modified qualification part evaluates True then output target-attribute-values else do nothing
end
... 
end

SELECT target-attribute-list
FROM R1, …, Rn
WHERE qualification
Table Definitions

We will be using the following relations in our examples:

Sailors(sid, sname, rating, age)

Boats(bid, bname, color)

Reserves(sid, bid, day)
## 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>
A Simple SQL Query

Find the names and ages of all 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>
Result of Previous Query

SELECT S.sname, S.age
FROM Sailors S;

<table>
<thead>
<tr>
<th>sname</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dustin</td>
<td>45.0</td>
</tr>
<tr>
<td>Brutus</td>
<td>33.0</td>
</tr>
<tr>
<td>Lubber</td>
<td>55.5</td>
</tr>
<tr>
<td>Andy</td>
<td>25.5</td>
</tr>
<tr>
<td>Rusty</td>
<td>35.0</td>
</tr>
<tr>
<td>Horatio</td>
<td>35.0</td>
</tr>
<tr>
<td>Zorba</td>
<td>16.0</td>
</tr>
<tr>
<td>Horatio</td>
<td>35.0</td>
</tr>
<tr>
<td>Art</td>
<td>25.5</td>
</tr>
<tr>
<td>Bob</td>
<td>63.5</td>
</tr>
</tbody>
</table>
Preventing Duplicate Tuples in the Result

- Use the `DISTINCT` keyword in the `SELECT` clause:

```
SELECT DISTINCT S.sname, S.age
FROM Sailors S;
```
Results of Original Query without Duplicates

<table>
<thead>
<tr>
<th>sname</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dustin</td>
<td>45.0</td>
</tr>
<tr>
<td>Brutus</td>
<td>33.0</td>
</tr>
<tr>
<td>Lubber</td>
<td>55.5</td>
</tr>
<tr>
<td>Andy</td>
<td>25.5</td>
</tr>
<tr>
<td>Rusty</td>
<td>35.0</td>
</tr>
<tr>
<td>Horatio</td>
<td>35.0</td>
</tr>
<tr>
<td>Zorba</td>
<td>16.0</td>
</tr>
<tr>
<td>Art</td>
<td>25.5</td>
</tr>
<tr>
<td>Bob</td>
<td>63.5</td>
</tr>
</tbody>
</table>
Example SQL Query…1

Find the names of sailors who have reserved boat 103

**Relational Algebra:**
\[ \pi_{sname} ((\sigma_{bid=103} Reserves) \bowtie Sailors) \]

**SQL:**
SELECT  S.sname  
FROM     Sailors S, Reserves R  
WHERE  S.sid=R.sid AND R.bid=103;
Result of Previous Query

<table>
<thead>
<tr>
<th>sid</th>
<th>bid</th>
<th>day</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>103</td>
<td>10/08/04</td>
</tr>
<tr>
<td>31</td>
<td>103</td>
<td>11/06/04</td>
</tr>
<tr>
<td>74</td>
<td>103</td>
<td>09/08/04</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>sid</th>
<th>name</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>

Result:

- Dustin
- Lubber
- Horatio
A Note on Range Variables

• Really needed only if the same relation appears twice in the FROM clause. The previous query can also be written as:

```sql
SELECT  S.sname
FROM     Sailors S, Reserves R
WHERE  S.sid=R.sid AND R.bid=103;
```

**OR**

```sql
SELECT  sname
FROM     Sailors, Reserves
WHERE  Sailors.sid=Reserves.sid AND bid=103;
```

However, it is a good style to always use range variables!
Example SQL Query...2

Find the *sids* of sailors who have reserved a red boat
Example SQL Query…3

*Find the names of sailors who have reserved a red boat*
Example SQL Query...4

Find the **colors** of boats reserved by ‘Lubber’
Example SQL Query...5

Find the *names* of sailors who have reserved *at least* one boat
Expressions and Strings

• **AS** and **=** are two ways to name fields in result.

• **LIKE** is used for string matching. ‘_’ stands for exactly one arbitrary character and ‘%’ stands for 0 or more arbitrary characters.
Expressions and Strings Example

Find triples (of ages of sailors and two fields defined by expressions, i.e. current age-1 and twice the current age) for sailors whose names begin and end with B and contain at least three characters.

SELECT S.age, age1=S.age-1, 2*S.age AS age2
FROM Sailors S
WHERE S.sname LIKE 'B_%B';

Result:

<table>
<thead>
<tr>
<th>sid</th>
<th>sname</th>
<th>rating</th>
<th>age</th>
<th>age1</th>
<th>age2</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>Dustin</td>
<td>7</td>
<td>45.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>29</td>
<td>Brutus</td>
<td>1</td>
<td>33.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>Lubber</td>
<td>8</td>
<td>55.5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>Andy</td>
<td>8</td>
<td>25.5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>58</td>
<td>Rusty</td>
<td>10</td>
<td>35.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>64</td>
<td>Horatio</td>
<td>7</td>
<td>35.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>71</td>
<td>Zorba</td>
<td>10</td>
<td>16.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>74</td>
<td>Horatio</td>
<td>9</td>
<td>35.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>85</td>
<td>Art</td>
<td>3</td>
<td>25.5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>95</td>
<td>Bob</td>
<td>3</td>
<td>63.5</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>age</th>
<th>age1</th>
<th>age2</th>
</tr>
</thead>
<tbody>
<tr>
<td>63.5</td>
<td>62.5</td>
<td>127.0</td>
</tr>
</tbody>
</table>
UNION, INTERSECT, EXCEPT

- **UNION**: Can be used to compute the union of any two *union-compatible* sets of tuples (which are themselves the result of SQL queries).

- **EXCEPT**: Can be used to compute the set-difference operation on two *union-compatible* sets of tuples (Note: In ORACLE, the command for set-difference is *MINUS*).

- **INTERSECT**: Can be used to compute the intersection of any two *union-compatible* sets of tuples.
Illustration of UNION…1

Find the names of sailors who have reserved a red or a green boat

Intuitively, we would write:

```
SELECT  S.sname
FROM  Sailors S, Boats B, Reserves R
AND (B.color='red' OR B.color='green');
```
Illustration of UNION…2

We can also do this using a UNION keyword:

```sql
SELECT  S.sname
FROM  Sailors S, Boats B, Reserves R
       AND B.color= ‘red’
UNION
SELECT  S.sname
FROM  Sailors S, Boats B, Reserves R
       AND B.color= ‘green’ ;
```

Unlike other operations, UNION eliminates duplicates! Same as INTERSECT, EXCEPT. To retain duplicates, use “UNION ALL”
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:

```
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

```sql
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.)
(But the query asks for the names of the sailors.)
Illustration of EXCEPT

Find the sids of all sailors who have reserved red boats \textbf{but not} green boats:

\begin{verbatim}
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';
\end{verbatim}

Use \textbf{MINUS} instead of \textbf{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*

```
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…

• 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. If it is, then return TRUE.

(For finding sailors who have not reserved boat 103, we would use NOT EXISTS)
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’ );

(Can you find the probable bug in this SQL query??)
Hint: what if there are several sailors named 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.rating
    FROM Sailors S2
    WHERE S2.sname=‘Horatio’);
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;
Example of SUM operator

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

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 );
Another Aggregate Query

Count the number of different sailors

SELECT COUNT (DISTINCT S.sname)
FROM Sailors S
More to come…

• BETWEEN…AND

Advanced SQL concepts :

• GROUP BY
• ORDER BY
• HAVING