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 $<$, $>$, $\leq$, $\geq$, $=$, $\neq$) 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 \textit{relation-list}.
  - Discard resulting tuples if they fail \textit{qualifications}.
  - Delete attributes that are not in \textit{target-list}.

- This strategy is probably the least efficient way to compute a query! An optimizer will find more efficient strategies to compute \textit{the same answers}. 
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>

Duplicate Results
Preventing Duplicate Tuples in the Result

• Use the **DISTINCT** keyword in the SELECT clause:

```sql
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:**

\[ \pi_{\text{sname}} ((\sigma_{\text{bid}=103} \text{Reserves}) \bowtie \text{Sailors}) \]

**SQL:**
Example SQL Query…

*Find the names of sailors who have reserved boat 103*

**Relational Algebra:**

\[ \pi_{\text{sname}} (\sigma_{\text{bid}=103} (\text{Reserves} \bowtie \text{Sailors})) \]

**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>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:

- 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:

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

OR

```
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*

```
SELECT R.sid
FROM Boats B, Reserves R
WHERE B.bid=R.bid AND B.color='red';
```
Example SQL Query…3

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

```sql
SELECT S.sname
FROM Sailors S, Boats B, Reserves R
WHERE S.sid=R.sid AND B.bid=R.bid AND
    B.color=‘red’;
```
Example SQL Query…4

Find the **colors** of boats reserved by ‘Lubber’

```sql
SELECT B.color
FROM Sailors S, Reserves R, Boats B
WHERE S.sid=R.sid AND R.bid=B.bid AND
  S.sname=‘Lubber’;
```
Example SQL Query...5

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

```
SELECT S.sname
FROM Sailors S, Reserves R
WHERE S.sid=R.sid;
```
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>
</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>

<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:

```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.)
(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

```
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*

```sql
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)
Correlated Nested Query - Division

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

```
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));
```

(For each sailor S, we check to see that the set of boats reserved by S includes every boat)
Correlated Nested Query 2

Alternatively,

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

```sql
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 ));
```
### NOT EXISTS vs. NOT IN

**Employee2**

<table>
<thead>
<tr>
<th>employee_id</th>
<th>employee_name</th>
<th>manager_id</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>John</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>David</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>Joe</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>Brandon</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>Chris</td>
<td>NULL</td>
</tr>
<tr>
<td>6</td>
<td>Jen</td>
<td>5</td>
</tr>
<tr>
<td>7</td>
<td>Kim</td>
<td>5</td>
</tr>
<tr>
<td>8</td>
<td>Mary</td>
<td>5</td>
</tr>
<tr>
<td>9</td>
<td>Dennis</td>
<td>5</td>
</tr>
<tr>
<td>10</td>
<td>Jim</td>
<td>5</td>
</tr>
</tbody>
</table>
NOT EXISTS vs. NOT IN

• Find employees who are not managers
Try:

SELECT COUNT(*)
FROM Employee2 E
WHERE E.employee_id NOT IN
  (SELECT E2.manager_id
   FROM Employee2 E2);
NOT EXISTS vs. NOT IN

- Find employees who are not managers

SELECT COUNT(*)
FROM Employee2 E
WHERE E.employee_id NOT IN
  (SELECT E2.manager_id
   FROM Employee2 E2);

COUNT = 0 (!)
NOT EXISTS vs. NOT IN

• Find employees who are not managers

Try again:

```
SELECT COUNT(*)
FROM Employee2 E
WHERE NOT EXISTS
  (SELECT *
   FROM Employee2 E2
   WHERE E2.manager_id = E.employee_id);
```
NOT EXISTS vs. NOT IN

- Find employees who are not managers

Try again:

```
SELECT COUNT(*)
FROM Employee2 E
WHERE NOT EXISTS
  (SELECT *
   FROM Employee2 E2
   WHERE E2.manager_id = E.employee_id);
COUNT = 9!
```
NOT EXISTS vs. NOT IN

• Find employees who are not managers

Another option:

SELECT COUNT(*)
FROM Employee2 E LEFT OUTER JOIN Employee2 E2
    ON E.employee_id = E2.manager_id
WHERE E2.manager_id IS NULL;
NOT EXISTS vs. NOT IN

• Performance
  – NOT IN: Query performs nested full table scans
  – NOT EXISTS: Query can use an index within the sub-query.