SQL: Queries, Constraints, Triggers
Part 1
Chapter 5.1-5.4
Structured Query Language (SQL)

- Introduced in 1974 by IBM
- “De facto” standard db query language
- Caveats
  - Standard has evolved (major revisions in 1992 and 1999)
  - Semantics and Syntax may vary slightly among DBMS implementation
Baby Example Instances

- We will start with these instances of the Sailors and Reserves relations in our examples.
- If the key for the Reserves relation contained only the attributes `sid` and `bid`, how would the semantics differ?

<table>
<thead>
<tr>
<th></th>
<th>sid</th>
<th>sname</th>
<th>rating</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>22</td>
<td>dustin</td>
<td>7</td>
<td>45.0</td>
</tr>
<tr>
<td>2</td>
<td>31</td>
<td>lubber</td>
<td>8</td>
<td>55.5</td>
</tr>
<tr>
<td>3</td>
<td>58</td>
<td>rusty</td>
<td>10</td>
<td>35.0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>sid</th>
<th>bid</th>
<th>day</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>22</td>
<td>101</td>
<td>10/10/96</td>
</tr>
<tr>
<td>5</td>
<td>58</td>
<td>103</td>
<td>11/12/96</td>
</tr>
</tbody>
</table>
Basic SQL Query

```
SELECT [DISTINCT] target-list
FROM relation-list
WHERE qualification
```

- **target-list** A list of attributes of relations in relation-list
- **relation-list** A list of relation names (possibly with a range-variable after each name).

- **qualification** Comparisons (Attr op const or Attr1 op Attr2, where op is one of <, >, =, ≤, ≥, ≠) combined using AND, OR and NOT.

- **DISTINCT** is an optional keyword indicating that the answer should not contain duplicates. Default is that duplicates are **not** eliminated!
Semantics of an SQL query defined in terms of the following **conceptual evaluation strategy**:

- Compute the cross-product of the *relation-list*.
- Select \( \sigma \) tuples if they satisfy *qualifications*.
- Project \( \pi \) attributes that in the *target-list*.
- If **DISTINCT** is specified, eliminate duplicate rows.

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

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

<table>
<thead>
<tr>
<th>(sid)</th>
<th>sname</th>
<th>rating</th>
<th>age</th>
<th>(sid)</th>
<th>bid</th>
<th>day</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>dustin</td>
<td>7</td>
<td>45.0</td>
<td>22</td>
<td>101</td>
<td>10/10/96</td>
</tr>
<tr>
<td>22</td>
<td>dustin</td>
<td>7</td>
<td>45.0</td>
<td>58</td>
<td>103</td>
<td>11/12/96</td>
</tr>
<tr>
<td>31</td>
<td>lubber</td>
<td>8</td>
<td>55.5</td>
<td>22</td>
<td>101</td>
<td>10/10/96</td>
</tr>
<tr>
<td>31</td>
<td>lubber</td>
<td>8</td>
<td>55.5</td>
<td>58</td>
<td>103</td>
<td>11/12/96</td>
</tr>
<tr>
<td>58</td>
<td>rusty</td>
<td>10</td>
<td>35.0</td>
<td>22</td>
<td>101</td>
<td>10/10/96</td>
</tr>
<tr>
<td>58</td>
<td>rusty</td>
<td>10</td>
<td>35.0</td>
<td>58</td>
<td>103</td>
<td>11/12/96</td>
</tr>
</tbody>
</table>
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 bid=103
```

OR

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

It is good style, however, to use range variables always!
Find sailors who’ve reserved at least one boat

```
SELECT S.sid
FROM Sailors S, Reserves R
WHERE S.sid=R.sid
```

- Would adding DISTINCT to this query make a difference?
- What is the effect of replacing S.sid by S.sname in the SELECT clause? Would adding DISTINCT to this variant of the query make a difference?
Expressions and Strings

SELECT  S.age, S.age-5 AS age1, 2*S.age AS age2  
FROM  Sailors S  
WHERE  S.sname LIKE ‘_us%’

- Illustrates use of arithmetic expressions and string pattern matching: Find triples (of ages of sailors and two fields defined by expressions) for sailors whose names begin and end with B and contain at least three characters.
- AS renames fields (ρ) in result. (Some SQL implementations allow the use of ‘newalias=expr’ as well)
- LIKE is used for string matching. “_” stands for any one character and “%” stands for 0 or more arbitrary characters.
### “Infant” Sailors/Reserves/Boats instance

**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>10/10/98</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:**

<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>
Find sid’s of sailors who’ve reserved a red or a green boat

- Two approaches
- **UNION**: Can be used to compute the union of any two union-compatible sets of tuples (which are themselves the result of SQL queries).
- If we replace **OR** by **AND** in the first version, what do we get?
- Also available: **EXCEPT** (What do we get if we replace **UNION** by **EXCEPT**?)

```sql
SELECT S.sid
FROM Sailors S, Boats B, Reserves R
AND (B.color='red' OR B.color='green')

SELECT S.sid
FROM Sailors S, Boats B, Reserves R
AND B.color='red'

UNION

SELECT S.sid
FROM Sailors S, Boats B, Reserves R
AND B.color='green'
```
Find sid’s of sailors who’ve reserved a red and a green boat

- Solution 1: Multiple instancing of the same relation in the relation-list using another variable
  
- Solution 2: **INTERSECT**: Can be used to compute the intersection of any two union-compatible sets of tuples.

- Included in the SQL/92 standard, but some systems don’t support it.

- Contrast symmetry of the UNION and INTERSECT queries with the first version.

```sql
SELECT S.sid
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')
```

```sql
SELECT S.sid
FROM Sailors S, Boats B, Reserves R
    AND B.color='red'
INTERSECT
SELECT S.sid
FROM Sailors S, Boats B, Reserves R
    AND 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 FROM and HAVING clauses.)

- To find sailors who’ve not reserved #103, use NOT IN.
- To understand semantics of nested queries, think of a nested loops evaluation: For each Sailors tuple, check the qualification by computing the 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.
- If UNIQUE is used in the inner SELECT, and * is replaced by R.bid, finds sailors with exactly one reservation for boat #103. (UNIQUE checks for duplicate tuples; * denotes all attributes. Why do we have to replace * by R.bid?)
- Illustrates why, in general, a subquery must be re-evaluated for each Sailors tuple.
More on Set-Comparison Operators

- We’ve already seen IN, EXISTS and UNIQUE. Can also use NOT IN, NOT EXISTS and NOT UNIQUE.
- Also available: \( op \text{ ANY, op } \text{ ALL, op } \text{ IN } \), \( >, <, =, \geq, \leq, \neq \)
- Find sailors whose rating is greater than that of some sailor called Horatio:

```sql
SELECT *
FROM Sailors S
WHERE S.rating > ANY (SELECT S2.rating
FROM Sailors S2
WHERE S2.sname='Horatio')
```
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 re-written using NOT IN.
- To find names (not sid’s) of Sailors who’ve reserved both red and green boats, just replace S.sid by S.sname in SELECT clause. (What about INTERSECT query?)
Division in SQL

Find sailors who’ve reserved all boats.

- The hard way, without EXCEPT:

  (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)

  Sailors S such that ...

  there is no boat B without ...

  a Reserves tuple showing S reserved B

  (2) 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)))
Next Time

- We’ve covered the portion of SQL that has the same power as relation algebra
- Next time we will consider some important extensions, that cannot be expressed in relational algebra, but are nonetheless useful tools for and a natural additions to query specification