SQL: Basic Queries

Problem Set #1
is now online
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 implementations
“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?

### 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>31</td>
<td>lubber</td>
<td>8</td>
<td>55.5</td>
</tr>
<tr>
<td>58</td>
<td>rusty</td>
<td>10</td>
<td>35.0</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>10/10/96</td>
</tr>
<tr>
<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. By default duplicates are **not** eliminated!
Conceptual Evaluation Strategy

- Semantics of an SQL query defined in terms of the following conceptual evaluation strategy:
  - Compute the cross-product of the relation-list.
  - Select tuples (rows) if they satisfy qualifications.
  - Select attributes (columns) 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>

Outputs:

- sname
- rusty
Table Aliases (Variables)

❖ Really needed only if the same relation appears more than once 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
```
Find sailors who’ve reserved at least one boat

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

- Why is the DISTINCT keyword useful here?
- What is the effect of replacing `S.sid` by `S.sname` in the SELECT clause?
- Does DISTINCT work as expected in this case?
- Just because a query appears to gives a correct answer on a specific database instance, does not mean that it is correct!
Expressions and Strings

- 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 have ‘us’ as the second and third letter of their name.

- `AS` renames fields in result. (Some SQL implementations allow the use of `newalias=expr` as well)

- `LIKE` is used for approximate string matching. “_” stands for any one character and “%” stands for 0 or more arbitrary characters.

SELECT S.age, S.age*12.0 AS ageMonths, 10-S.rating AS revRating
FROM Sailors S
WHERE S.sname LIKE '_us%'

<table>
<thead>
<tr>
<th>age</th>
<th>ageMonths</th>
<th>revRating</th>
</tr>
</thead>
<tbody>
<tr>
<td>45.0</td>
<td>540.0</td>
<td>3</td>
</tr>
<tr>
<td>35.0</td>
<td>420.0</td>
<td>0</td>
</tr>
</tbody>
</table>
## A more extensive example

### “Infant” Sailors/Reserves/Boats instance

<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>
<tr>
<td>31</td>
<td>Lubber</td>
<td>8</td>
</tr>
<tr>
<td>32</td>
<td>Andy</td>
<td>8</td>
</tr>
<tr>
<td>58</td>
<td>Rusty</td>
<td>10</td>
</tr>
<tr>
<td>64</td>
<td>Horatio</td>
<td>7</td>
</tr>
<tr>
<td>71</td>
<td>Zorba</td>
<td>10</td>
</tr>
<tr>
<td>74</td>
<td>Horatio</td>
<td>9</td>
</tr>
<tr>
<td>85</td>
<td>Art</td>
<td>3</td>
</tr>
<tr>
<td>95</td>
<td>Bob</td>
<td>3</td>
</tr>
</tbody>
</table>
Find sid’s of sailors who’ve reserved a red or a green boat

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

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

```sql
SELECT S.sname, S.sid
FROM   Sailors S, Boats B, Reserves R
AND    B.color= "red"
UNION
SELECT S.sname, S.sid
FROM   Sailors S, Boats B, Reserves R
AND    B.color= "green"
EXCEPT
SELECT S.sname, S.sid
FROM   Sailors S, Boats B, Reserves R
AND    B.color= "red"
```
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.

- Consider the symmetry of the UNION, EXCEPT, and INTERSECT queries versus the first, multiple instancing version.

```
SELECT DISTINCT S.sname, 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")

SELECT S.sname, S.sid
FROM   Sailors S, Boats B, Reserves R
       AND B.color="red"
INTERSECT
SELECT S.sname, S.sid
FROM   Sailors S, Boats B, Reserves R
       AND B.color="green"

<table>
<thead>
<tr>
<th>sname</th>
<th>sid</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dustin</td>
<td>22</td>
</tr>
<tr>
<td>Lubber</td>
<td>31</td>
</tr>
</tbody>
</table>
```
Nested Queries

Find names of sailors who’ve never reserved boat #103:

```sql
SELECT S.sid, S.sname
FROM   Sailors S
WHERE  S.sid NOT IN (SELECT DISTINCT R.sid
                      FROM   Reserves R
                      WHERE  R.bid=103)
```

- A very powerful feature of SQL: a WHERE clause can itself contain an SQL query!
- To find sailors who’ve reserved #103, use 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 any boat:

\[
\text{SELECT } S.\text{sid}, S.\text{surname} \\
\text{FROM } \text{Sailors } S \\
\text{WHERE } \text{EXISTS (SELECT} \\
\text{FROM Reserves R} \\
\text{WHERE } S.\text{sid}=R.\text{sid)}
\]

- **EXISTS** is another set comparison operator, like **IN**.
- 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* **ANY**, *op** **ALL**, *op** **IN**

❖ 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')
```

Not every SQL dialect supports **ANY** and **ALL**. However, `min()` and `max()` functions can usually be used to achieve the desired effect.

```sql
SELECT * 
FROM Sailors S 
WHERE S.rating > (SELECT min(S2.rating) 
FROM Sailors S2 
WHERE S2.sname='Horatio')
```

---

<table>
<thead>
<tr>
<th>sid</th>
<th>sname</th>
<th>rating</th>
<th>age</th>
</tr>
</thead>
<tbody>
<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>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>
</tbody>
</table>
Rewriting INTERSECT Queries Using IN

Find sid’s of sailors who’ve reserved both a red and a green boat:

```sql
SELECT DISTINCT S.sid, S.sname
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.

<table>
<thead>
<tr>
<th>sid</th>
<th>sname</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>Dustin</td>
</tr>
<tr>
<td>31</td>
<td>Lubber</td>
</tr>
</tbody>
</table>
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)

(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))

Sailors S such that ...
there is no boat B without ...
a Reserves tuple showing S reserved B
Next Time

- We’ve covered the portion of SQL that strictly returns "tuples from tables"

- Next time we will consider some important extensions, that summarize sets of tuples. They are useful and a natural additions to query specification.