SQL Queries

Chapter 5
The Structured Query Language

- Developed by IBM (system R) in the 1970s
- *The* most widely used language for creating, manipulating, and querying relational DBMS.
- Need for a standard since it is used by many vendors
- Standards:
  - SQL-86
  - SQL-89 (minor revision)
  - SQL-92 (major revision)
  - SQL-99 (major extensions, current standard)
**Example Instances**

- We will use these instances of the Sailors and Reserves relations in our examples.

<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>

<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>

**Relations**

- **Sailor** (sid, sname, rating, age)
- **Reserve** (sid, bid, day)
- **Boat** (bid, color)

```
R1
sid  bid  day      
22   101  10/10/96|
58   103  11/12/96|
```

```
S2
sid  sname  rating | age  |
28  uppy   9       | 35.0 |
31   lubber 8       | 55.5 |
44   guppy  5       | 35.0 |
58   rusty  10      | 35.0 |
```
Basic SQL Query

- **relation-list**  A list of relation names
- **target-list**  A list of attributes of relations in relation-list
- **qualification**  Comparisons (Attr \( \text{op} \) const or Attr1 \( \text{op} \) Attr2, where \( \text{op} \) is one of \( , , , \leq, \geq, \neq \) ) 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!
Conceptual Evaluation Strategy

- Semantics of an SQL query defined in terms of the following conceptual evaluation strategy:
  1. Compute the cross-product of relation-list.
  2. Discard resulting tuples if they fail qualifications.
  3. Delete attributes that are not in target-list.
  4. 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
```
Expressions and Strings

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

- 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 and = are two ways to name fields in result.
- LIKE is used for string matching. ‘_’ stands for any one character and ‘%’ stands for 0 or more arbitrary characters.
Find sailors who’ve reserved at least one boat

Would adding DISTINCT to this query make a difference?

Find the names of sailors who have reserved a red boat
Find sid’s and names of sailors who’ve reserved a red or a green boat

- **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, S.sname
FROM Sailors S, Boats B, Reserves R
AND (B.color='red' OR B.color='green')

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

- **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 how much the other versions differ.

Sailor (sid, sname, rating, age)
Reserve (sid, bid, day)
Boat (bid, color)

```
SELECT S.sid, 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'
```

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

SELECT S.sid, 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'
Find sid’s and names of sailors who’ve reserved red boats but not green boats

- Sailor (sid, sname, rating, age)
- Reserve (sid, bid, day)
- Boat (bid, color)

```
SELECT 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'
EXCEPT
SELECT S.sid, S.sname
FROM Sailors S, Boats B, Reserves R
WHERE S.sid=R.sid AND R.bid=B.bid AND B.color='green'
```

- **EXCEPT**: Can be used to compute the difference of any two union-compatible sets of tuples

- Many systems recognize the keyword **MINUS** instead of **EXCEPT**
Nested Queries

Find names of sailors who’ve reserved boat #103

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 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

Find names of sailors who’ve 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)
```

- Sailor (sid, sname, rating, age)
- Reserve (sid, bid, day)
- Boat (bid, color)
**Nested Queries**

- Sailor (sid, sname, rating, age)
- Reserve (sid, bid, day)
- Boat (bid, color)

*Find names of sailors who’ve not reserved a red boat*

```
SELECT S.sname
FROM Sailors S
WHERE S.sid NOT IN (SELECT R.sid
                     FROM Reserves R
                     WHERE R.bid IN (SELECT B.bid
                                      FROM Boats B
                                      WHERE B.color = 'red'))
```
Nested Queries with Correlation

Find names of sailors who’ve reserved boat #103:

\[
\text{SELECT S.sname} \\
\text{FROM Sailors S} \\
\text{WHERE EXISTS (SELECT *} \\
\text{FROM Reserves R} \\
\text{WHERE R.bid=103 AND S.sid=R.sid)}
\]

- EXISTS is another set comparison operator, like IN.
- If UNIQUE is used, and * is replaced by \(R.bid\), finds sailors with at most one reservation for boat #103.
- Illustrates why, in general, subquery must be recomputed 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`, `op >`, `op >=`, `op <=`, `op !=`
- Find sailors whose rating is greater than that of some sailor called Horatio:

```
SELECT * 
FROM  Sailors S
WHERE  S.rating > ANY (SELECT S2.rating 
FROM  Sailors S2 
WHERE S2.sname='Horatio')
```
Rewriting \textbf{INTERSECT} Queries Using \textbf{IN}

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

\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'
     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')
\end{verbatim}

- Similarly, \textbf{EXCEPT} queries re-written using \textbf{NOT IN}.
- Useful if your system does not support \textbf{INTERSECT}
or \textbf{EXCEPT}
Division in SQL

Find sailors who’ve reserved all boats.

- Let’s do it the hard way, without EXCEPT:

```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))
```

```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))
```
**Aggregate Operators**

- Significant extension of relational algebra.

<table>
<thead>
<tr>
<th>sid</th>
<th>sname</th>
<th>rating</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>28</td>
<td>Bob</td>
<td>10</td>
<td>35</td>
</tr>
<tr>
<td>31</td>
<td>Bob</td>
<td>10</td>
<td>20</td>
</tr>
<tr>
<td>44</td>
<td>guppy</td>
<td>5</td>
<td>50</td>
</tr>
<tr>
<td>58</td>
<td>rusty</td>
<td>10</td>
<td>35</td>
</tr>
</tbody>
</table>

- COUNT (*)
- COUNT ([DISTINCT] A)
- SUM ([DISTINCT] A)
- AVG ([DISTINCT] A)
- MAX (A)
- MIN (A)

- SELECT COUNT (*)
  FROM Sailors S

- SELECT AVG (S.age)
  FROM Sailors S
  WHERE S.rating=10

- SELECT AVG (DISTINCT S.age)
  FROM Sailors S
  WHERE S.rating=10

- SELECT COUNT (DISTINCT S.rating)
  FROM Sailors S
  WHERE S.sname='Bob'

- SELECT MAX (S.age)
  FROM Sailors S
Find name and age of the oldest sailor(s)

- The first query is illegal! (We’ll look into the reason a bit later, when we discuss \texttt{GROUP BY}.)
- The third query is equivalent to the second query, and is allowed in the SQL/92 standard, but is not supported in some systems.

\begin{verbatim}
SELECT  S.sname, MAX (S.age) 
FROM   Sailors S 

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

SELECT  S.sname, S.age 
FROM   Sailors S 
WHERE  (SELECT  MAX (S2.age) 
        FROM   Sailors S2) 
       = S.age
\end{verbatim}
Motivation for Grouping

- So far, we’ve applied aggregate operators to all (qualifying) tuples. Sometimes, we want to apply them to each of several groups of tuples.

- Consider: Find the age of the youngest sailor for each rating level.
  - Suppose we know that rating values go from 1 to 10; we can write 10 queries that look like this (!):
    ```sql
    SELECT MIN (S.age)
    FROM Sailors S
    WHERE S.rating = i
    For i = 1, 2, ..., 10:
    ```
  - In general, we don’t know how many rating levels exist, and what the rating values for these levels are!
Queries With GROUP BY and HAVING

SELECT [DISTINCT] target-list
FROM relation-list
WHERE qualification
GROUP BY grouping-list
HAVING group-qualification

- The target-list contains (i) attribute names (ii) terms with aggregate operations (e.g., MIN (S.age)).
  - The attribute list (i) must be a subset of grouping-list. Intuitively, each answer tuple corresponds to a group, and these attributes must have a single value per group. (A group is a set of tuples that have the same value for all attributes in grouping-list.)