SQL

**Basic SQL Query**

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

- Default is that duplicates are not eliminated!
  - Need to explicitly say “DISTINCT”

```
SELECT S.Name
FROM Sailors S
WHERE S.Age > 25
```

```
SELECT DISTINCT S.Name
FROM Sailors S
WHERE S.Age > 25
```

**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 condition.
  - Delete attributes that are not in 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>
</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>22</td>
<td>103</td>
<td>10/10/96</td>
</tr>
<tr>
<td>58</td>
<td>103</td>
<td>11/12/96</td>
</tr>
<tr>
<td>31</td>
<td>101</td>
<td>10/10/96</td>
</tr>
<tr>
<td>31</td>
<td>103</td>
<td>11/12/96</td>
</tr>
<tr>
<td>58</td>
<td>103</td>
<td>11/12/96</td>
</tr>
</tbody>
</table>

**A Slightly Modified Query**

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

- Would adding **DISTINCT** to this query make a difference?
Find sid's of sailors who've reserved a red or a green boat

\[
\text{SELECT } S\text{.sid} \\
\text{FROM Sailors S, Boats B, Reserves R} \\
\text{WHERE S\text{.sid}=R\text{.sid} AND R\text{.bid}=B\text{.bid} AND (B\text{.color}='\text{red}' OR B\text{.color}='\text{green}')}
\]

What does this query compute?

\[
\text{SELECT } S\text{.sid} \\
\text{FROM Sailors S, Boats B1, Reserves R1, Boats B2, Reserves R2} \\
\text{WHERE S\text{.sid}=R1\text{.sid} AND R1\text{.bid}=B1\text{.bid} AND S\text{.sid}=R2\text{.sid} AND R2\text{.bid}=B2\text{.bid} AND B1\text{.color}='\text{red}' AND B2\text{.color}='\text{green}'}
\]

Find sid's of sailors who've reserved a red and a green boat

\[
\text{SELECT S\text{.sid} FROM Sailors S, Boats B, Reserves R} \\
\text{WHERE S\text{.sid}=R\text{.sid} AND R\text{.bid}=B\text{.bid} AND B\text{.color}='\text{red}'} \\
\text{INTERSECT} \\
\text{SELECT S\text{.sid} FROM Sailors S, Boats B, Reserves R} \\
\text{WHERE S\text{.sid}=R\text{.sid} AND R\text{.bid}=B\text{.bid} AND B\text{.color}='\text{green}'}
\]

Expressions and Strings

\[
\text{SELECT S\text{.age}, S\text{.age}-5 AS age2, 2*S\text{.age} AS age2} \\
\text{FROM Sailors S} \\
\text{WHERE S\text{.sname} LIKE 'B\_\_B'}
\]

Nested Queries (with Correlation)

Find names of sailors who have reserved boat #103:

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

Find names of sailors who have not reserved boat #103:

\[
\text{SELECT S\text{.sname} FROM Sailors S} \\
\text{WHERE NOT EXISTS (SELECT \_ FROM Reserves R WHERE R\text{.bid}=103 AND S\text{.sid}=R\text{.sid})}
\]
**Division in SQL**

Find sailors who've reserved all boats

```sql
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))
```

**Division in SQL (without Except!)**

Find sailors who've reserved all boats.

```sql
SELECT S.sname
FROM Sailors S
WHERE NOT EXISTS ((
    SELECT B.bid
    FROM Boats B)
    EXCEPT
    (SELECT R.bid
     FROM Reserves R
     WHERE R.bid=B.bid
     AND R.sid=S.sid))
```

**More on Set-Comparison Operators**

- `op ANY, op ALL`
  - `op` can be `>, <, =, ≥, ≤, ≠`
  - Find sailors whose rating is greater than that of all sailors called Horatio:

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

**Aggregate Operators**

- Significant extension of relational algebra.

```sql
SELECT COUNT (*)
FROM Sailors S
WHERE S.rating=10
```

```sql
SELECT COUNT (*)
FROM Sailors S
WHERE S.sname='Bob'
```

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

**Find name and age of the oldest sailor(s) with rating > 7**

```sql
SELECT S.sname, MAX (S.age)
FROM Sailors S
WHERE S.rating > 7
```

- This query is illegal!
- MAX (and other aggregate operators) can only return one tuple!

**Find name and age of the oldest sailor(s) with rating > 7**

```sql
SELECT S.sname, S.age
FROM Sailors S
WHERE S.rating > 7 AND S.age = (SELECT MAX (S2.age)
FROM Sailors S2
WHERE S2.Rating > 7)
```
Aggregate Operators

- 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.
  
  - If rating values go from 1 to 10; we can write 10 queries that look like this:
    
    For $i = 1, 2, \ldots, 10$:
    
    $\text{SELECT MIN}(S.\text{age})$
    $\text{FROM Sailors } S$
    $\text{WHERE } S.\text{rating} = i$

GROUP BY

$\text{Find the age of the youngest sailor for each rating level}$

GROUP BY grouping-list

$\text{SELECT } [\text{DISTINCT}]$ target-list
$\text{FROM relation-list}$
$\text{[WHERE condition]}$

Conceptual Evaluation Strategy

- Semantics of an SQL query defined as follows:
  
  - Compute the cross-product of relation-list.
  - Discard resulting tuples if they fail condition.
  - Delete attributes that are not in target-list.
  - Remaining tuples are partitioned into groups by the value of the attributes in grouping-list.
  - One answer tuple is generated per group.
- Note: Does not imply query will actually be evaluated this way!

Find the age of the youngest sailor with age $\geq 18$, for each rating with at least one such sailor

$\text{SELECT S.rating, MIN(S.age)}$
$\text{FROM Sailors S}$
$\text{WHERE S.age} \geq 18$
$\text{GROUP BY S.rating}$

What does this query compute?

$\text{SELECT B.bid, COUNT(*) AS scount}$
$\text{FROM Reserves R, Boats B}$
$\text{WHERE R.bid=B.bid AND B.color='red'}$
$\text{GROUP BY B.bid}$

Find those ratings for which the average age is the minimum over all ratings

$\text{SELECT Temp.rating, Temp.avgage}$
$\text{FROM (SELECT S.rating, AVG(S.age) AS avgage}$
$\text{FROM Sailors S}$
$\text{GROUP BY S.rating) AS Temp}$
$\text{WHERE Temp.avgage} = (\text{SELECT MIN} (\text{Temp2.avgage})$
$\text{FROM (SELECT AVG(S.age) as avgage}$
$\text{FROM Sailors S}$
$\text{GROUP BY S.rating)})$
Find those ratings for which the average age is the minimum over all ratings

SELECT S.rating
FROM Sailors S
WHERE S.age = (SELECT MIN(AVG(S2.age)) FROM Sailors S2)

• This is WRONG!
  – Cannot “nest” aggregates

What does this query compute?

SELECT Temp.rating, Temp.minage
FROM (SELECT S.rating, MIN(S.age) AS minage, COUNT(*) AS cnt
FROM Sailors S
WHERE S.age >= 18
GROUP BY S.rating) AS Temp
WHERE Temp.cnt >= 2

Queries With GROUP BY and HAVING

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

Find the age of the youngest sailor with age >= 18,
for each rating with at least 2 such sailors

SELECT S.rating, MIN(S.Age)
FROM Sailors S
WHERE S.age >= 18
GROUP BY S.rating
HAVING COUNT(*) >= 2

Conceptual Evaluation Strategy

• Semantics of an SQL query defined as follows:
  - Compute the cross-product of relation-list
  - Discard resulting tuples if they fail condition.
  - Delete attributes that are not in target-list.
  - Remaining tuples are partitioned into groups by the value
    of the attributes in grouping-list.
  - The group-qualification is applied to eliminate some groups.
  - One answer tuple is generated per qualifying group.
• Note: Does not imply query will actually be evaluated this way!

Find the age of the youngest sailor with age >= 18,
for each rating with at least 2 sailors (of any age)

SELECT S.rating, MIN(S.Age)
FROM Sailors S
WHERE S.age >= 18
GROUP BY S.rating
HAVING 1 < (SELECT COUNT(*)
FROM Sailors S2
WHERE S.rating=S2.rating)
### Null Values

- Field values in a tuple are sometimes *unknown*  
  - *e.g.* a rating has not been assigned
- Field values are sometimes *inapplicable*  
  - *e.g.* no spouse’s name
- SQL provides a special value `null` for such situations.

### Queries and Null Values

- What if `S.Age` is `NULL`?  
  - `S.Age > 25` returns `NULL`!
- Implies a predicate can return `3` values  
  - `True`, `false`, `NULL`
- Three valued logic!  
- Where clause eliminates rows that do not return `true` (i.e., which are `false` or `NULL`)

### Three-valued Logic

<table>
<thead>
<tr>
<th><code>A</code></th>
<th><code>NOT(A)</code></th>
<th><code>OR</code></th>
</tr>
</thead>
<tbody>
<tr>
<td><code>True</code></td>
<td><code>False</code></td>
<td><code>True</code></td>
</tr>
<tr>
<td><code>False</code></td>
<td><code>True</code></td>
<td><code>False</code></td>
</tr>
<tr>
<td><code>NULL</code></td>
<td><code>NULL</code></td>
<td><code>NULL</code></td>
</tr>
</tbody>
</table>

### General Constraints

- Useful when more general ICs than keys are involved
- Can use queries to express constraint
- Constraints can be named

- **CREATE TABLE** `Reserves`  
  ```sql
  CREATE TABLE Reserves  
  ( sname CHAR(10),  
    bid INTEGER,  
    day DATE,  
    PRIMARY KEY (bid, day),  
    CONSTRAINT noInterlakeRes  
    CHECK ( Interlake <>  
      ( SELECT B.bname  
        FROM Boats B  
        WHERE B.bid=bid ) )
  )
  ```

### Constraints Over Multiple Relations

**Number of boats plus number of sailors is < 100**

- **CREATE ASSERTION** `smallClub`  
  ```sql
  CREATE ASSERTION smallClub  
  CHECK  
  ( (SELECT COUNT(S.sid) FROM Sailors S)  
    + (SELECT COUNT(B.bid) FROM Boats B) < 100 )
  ```