Basic SQL Query

- Default is that duplicates are *not* eliminated!
  - Need to explicitly say "DISTINCT"

```sql
SELECT S.sname
FROM Sailors S
WHERE S.age > 25
```

```sql
SELECT DISTINCT S.sname
FROM Sailors S
WHERE S.age > 25
```

SQL Query

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

<table>
<thead>
<tr>
<th>Sailors</th>
<th>Reserves</th>
</tr>
</thead>
<tbody>
<tr>
<td>sid</td>
<td>bid</td>
</tr>
<tr>
<td>22</td>
<td>101</td>
</tr>
<tr>
<td>31</td>
<td>103</td>
</tr>
<tr>
<td>58</td>
<td></td>
</tr>
<tr>
<td>name</td>
<td>rating</td>
</tr>
<tr>
<td>dustin</td>
<td>7</td>
</tr>
<tr>
<td>lubber</td>
<td>8</td>
</tr>
<tr>
<td>rusty</td>
<td>10</td>
</tr>
</tbody>
</table>
Conceptual Evaluation Strategy

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

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>name</th>
<th>rating</th>
<th>age</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>10/10/96</td>
</tr>
<tr>
<td>22</td>
<td>dustin</td>
<td>7</td>
<td>45.0</td>
<td>58</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>10/10/96</td>
</tr>
<tr>
<td>31</td>
<td>lubber</td>
<td>8</td>
<td>55.5</td>
<td>58</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>10/10/96</td>
</tr>
<tr>
<td>58</td>
<td>rusty</td>
<td>10</td>
<td>35.0</td>
<td>58</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?
ICE: Find sid's of sailors who've reserved a red or a green boat

ICE: What Does This Query Compute?

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'

ICE: Find sid's of sailors who've reserved a red and a green boat
Expressions and Strings

- **AS** is used to name fields in result.
- **LIKE** is used for string matching
  - `\_` stands for any one character
  - `\%` stands for 0 or more arbitrary characters.

```sql
SELECT S.age, S.age-5 AS age1, 2*S.age AS age2
FROM Sailors S
WHERE S.sname LIKE 'B_%B'
```

Nested Queries (with Correlation)

**Find names of sailors who have reserved boat #103:**

```sql
SELECT S.sname
FROM Sailors S
WHERE EXISTS (SELECT * FROM Reserves R
              WHERE R.bid=103 AND S.sid=R.sid)
```

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

```sql
SELECT S.sname
FROM Sailors S
WHERE NOT EXISTS (SELECT * FROM Reserves R
                 WHERE R.bid=103 AND S.sid=R.sid)
```
Division in SQL

Find sailors who've reserved all boats

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.

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

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:
Aggregate Operators

Significant extension of relational algebra.

<table>
<thead>
<tr>
<th>Aggregate Functions</th>
<th>SQL Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>COUNT(*)</td>
<td>SELECT COUNT(*) FROM Sailors S</td>
</tr>
<tr>
<td>SUM (DISTINCT A)</td>
<td>SELECT AVG(S.age) FROM Sailors S WHERE S.rating=10</td>
</tr>
<tr>
<td>AVG (DISTINCT A)</td>
<td>SELECT COUNT(DISTINCT S.rating) FROM Sailors S WHERE S.sname='Bob'</td>
</tr>
</tbody>
</table>

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

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 \( i = 1, 2, \ldots, 10 \): 
SELECT MIN(S.age) FROM Sailors S WHERE S.rating = \( i \)
GROUP BY

Find the age of the youngest sailor for each rating level

<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>15.5</td>
</tr>
<tr>
<td>71</td>
<td>zorba</td>
<td>10</td>
<td>16.0</td>
</tr>
<tr>
<td>64</td>
<td>horatio</td>
<td>7</td>
<td>35.0</td>
</tr>
<tr>
<td>29</td>
<td>brutus</td>
<td>1</td>
<td>33.0</td>
</tr>
<tr>
<td>58</td>
<td>rusty</td>
<td>10</td>
<td>35.0</td>
</tr>
</tbody>
</table>

Answer relation

Conceptual Evaluation Strategy

• Semantics of an SQL query defined as follows:

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

<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>15.5</td>
</tr>
<tr>
<td>71</td>
<td>zorba</td>
<td>10</td>
<td>16.0</td>
</tr>
<tr>
<td>64</td>
<td>horatio</td>
<td>7</td>
<td>35.0</td>
</tr>
<tr>
<td>29</td>
<td>brutus</td>
<td>1</td>
<td>33.0</td>
</tr>
<tr>
<td>58</td>
<td>rusty</td>
<td>10</td>
<td>35.0</td>
</tr>
</tbody>
</table>
ICE: What does this query compute?

```
SELECT  B.bid,  COUNT(*) AS scount
FROM    Reserves R, Boats B
WHERE   R.bid=B.bid AND B.color='red'
GROUP BY B.bid
```

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

```
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 level 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 such sailors**

```
SELECT S.rating, MIN(S.Age)
FROM Sailors S
WHERE S.age >= 18
GROUP BY S.rating
HAVING COUNT(*) > 1
```
ICE: Find the age of the youngest sailor with age \( \geq 18 \), for each rating with at least 2 sailors (of any age)

\[
\text{SELECT S.rating, MIN(S.age) AS youngest_age}
\text{FROM Sailors S}
\text{GROUP BY S.rating}
\]

Find the average age for each rating, and order results in ascending order on avg. age

\[
\text{SELECT S.rating, AVG(S.age) AS avgage}
\text{FROM Sailors S}
\text{GROUP BY S.rating}
\text{ORDER BY avgage}
\]

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?

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

Three-valued Logic

- What if one or both of S.age and S.rating are NULL?

```
SELECT S.Name
FROM Sailors S
WHERE NOT(S.Age > 25) OR S.rating > 7
```

A/B

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

A/B

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>A OR B</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
<td>True</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>False</td>
<td>False</td>
<td>False</td>
</tr>
<tr>
<td>NULL</td>
<td>True</td>
<td>NULL</td>
</tr>
<tr>
<td>NULL</td>
<td>False</td>
<td>NULL</td>
</tr>
</tbody>
</table>

NOT Truth Table

General Constraints

```
CREATE TABLE Reserves
(sname CHAR(10) NOT NULL,
bid INTEGER NOT NULL,
day DATE NOT NULL,
PRIMARY KEY (bid, day),
CONSTRAINT noInterlakeRes
CHECK ('Interlake' <>
(SELECT B.bname
FROM Boats B
WHERE B.bid=bid))
```
CREATE ASSERTION smallClub
CHECK
   ( (SELECT COUNT (S.sid) FROM Sailors S) + (SELECT COUNT (B.bid) FROM Boats B) < 100 )