CS460: Intro to Database Systems

Class 8: SQL, The Query Language – Part I

Instructor: Manos Athanassoulis

https://midas.bu.edu/classes/CS460/
Today’s course

**intuitive** way to ask **queries**

unlike *procedural languages* (C/C++, java)
[which specify **how** to solve a problem (or answer a question)]

SQL is a **declarative query** language
[we ask **what we want** and the DBMS is going to deliver]
Introduction to SQL

SQL is a relational **query language**

supports **simple** yet **powerful** **querying** of data

It has two parts:

**DDL:** Data Definition Language (define and modify schema)
(we discussed about that in Relational Model)

**DML:** Data Manipulation Language (**intuitively** query data)
Reiterate some terminology

### Relation (or table)

<table>
<thead>
<tr>
<th>sid</th>
<th>name</th>
<th>login</th>
<th>age</th>
<th>gpa</th>
</tr>
</thead>
<tbody>
<tr>
<td>53666</td>
<td>Jones</td>
<td>jones@cs</td>
<td>18</td>
<td>3.4</td>
</tr>
<tr>
<td>53688</td>
<td>Smith</td>
<td>smith@ee</td>
<td>18</td>
<td>3.2</td>
</tr>
</tbody>
</table>

### Row (or tuple)

<table>
<thead>
<tr>
<th>sid</th>
<th>name</th>
<th>login</th>
<th>age</th>
<th>gpa</th>
</tr>
</thead>
<tbody>
<tr>
<td>53666</td>
<td>Jones</td>
<td>jones@cs</td>
<td>18</td>
<td>3.4</td>
</tr>
<tr>
<td>53688</td>
<td>Smith</td>
<td>smith@ee</td>
<td>18</td>
<td>3.2</td>
</tr>
</tbody>
</table>

### Column (or attribute)

<table>
<thead>
<tr>
<th>sid</th>
<th>name</th>
<th>login</th>
<th>age</th>
<th>gpa</th>
</tr>
</thead>
<tbody>
<tr>
<td>53666</td>
<td>Jones</td>
<td>jones@cs</td>
<td>18</td>
<td>3.4</td>
</tr>
<tr>
<td>53688</td>
<td>Smith</td>
<td>smith@ee</td>
<td>18</td>
<td>3.2</td>
</tr>
</tbody>
</table>
Reiterate some terminology

Primary Key (PK)

<table>
<thead>
<tr>
<th>sid</th>
<th>name</th>
<th>login</th>
<th>age</th>
<th>gpa</th>
</tr>
</thead>
<tbody>
<tr>
<td>53666</td>
<td>Jones</td>
<td>jones@cs</td>
<td>18</td>
<td>3.4</td>
</tr>
<tr>
<td>53688</td>
<td>Smith</td>
<td>smith@ee</td>
<td>18</td>
<td>3.2</td>
</tr>
</tbody>
</table>

The PK of a relation is the column (or the group of columns) that can uniquely define a row.

In other words:

Two rows **cannot** have the same PK.
The simplest SQL query

“find all contents of a table”
in this example: “Find all info for all students”

```
SELECT * 
FROM Students S
```

<table>
<thead>
<tr>
<th>sid</th>
<th>name</th>
<th>login</th>
<th>age</th>
<th>gpa</th>
</tr>
</thead>
<tbody>
<tr>
<td>53666</td>
<td>Jones</td>
<td>jones@cs</td>
<td>18</td>
<td>3.4</td>
</tr>
<tr>
<td>53688</td>
<td>Smith</td>
<td>smith@ee</td>
<td>18</td>
<td>3.2</td>
</tr>
<tr>
<td>53777</td>
<td>White</td>
<td>white@cs</td>
<td>19</td>
<td>4.0</td>
</tr>
</tbody>
</table>

to find just names and logins, replace the first line:

```
SELECT S.name, S.login
```
Show specific columns

“find name and login for all students”

```
SELECT S.name, S.login
FROM Students S
```

<table>
<thead>
<tr>
<th>name</th>
<th>login</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jones</td>
<td>jones@cs</td>
</tr>
<tr>
<td>Smith</td>
<td>smith@ee</td>
</tr>
<tr>
<td>White</td>
<td>white@cs</td>
</tr>
</tbody>
</table>

this is called: “project name and login from table Students”
Show specific rows

“find all 18 year old students”

```
SELECT * 
FROM Students S 
WHERE S.age=18
```

<table>
<thead>
<tr>
<th>sid</th>
<th>name</th>
<th>login</th>
<th>age</th>
<th>gpa</th>
</tr>
</thead>
<tbody>
<tr>
<td>53666</td>
<td>Jones</td>
<td>jones@cs</td>
<td>18</td>
<td>3.4</td>
</tr>
<tr>
<td>53688</td>
<td>Smith</td>
<td>smith@ee</td>
<td>18</td>
<td>3.2</td>
</tr>
</tbody>
</table>

this is called: “select students with age 18.”
Querying Multiple Relations

can specify a join over two tables as follows:

```sql
SELECT S.name, E.cid
FROM Students S, Enrolled E
WHERE S.sid=E.sid AND E.grade='B'
```

<table>
<thead>
<tr>
<th>sid</th>
<th>cid</th>
<th>grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>53831</td>
<td>Carnatic101</td>
<td>C</td>
</tr>
<tr>
<td>53831</td>
<td>Reggae203</td>
<td>B</td>
</tr>
<tr>
<td>53650</td>
<td>Topology112</td>
<td>A</td>
</tr>
<tr>
<td>53666</td>
<td>History105</td>
<td>B</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>sid</th>
<th>name</th>
<th>login</th>
<th>age</th>
<th>gpa</th>
</tr>
</thead>
<tbody>
<tr>
<td>53666</td>
<td>Jones</td>
<td>jones@cs</td>
<td>18</td>
<td>3.4</td>
</tr>
<tr>
<td>53688</td>
<td>Smith</td>
<td>smith@ee</td>
<td>18</td>
<td>3.2</td>
</tr>
</tbody>
</table>

result =

<table>
<thead>
<tr>
<th>S.name</th>
<th>E.cid</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jones</td>
<td>History105</td>
</tr>
</tbody>
</table>
Basic SQL Query

relation-list : a list of relations

target-list : a list of attributes of tables in relation-list

qualification : comparisons using AND, OR and NOT
comparisons are: <attr> <op> <const> or <attr1> <op> <attr2>, where op is:

<, >, =, ≤, ≥, ≠

DISTINCT: optional, removes duplicates

By default SQL SELECT does not eliminate duplicates! ("multiset")
Conceptually, a SQL query can be computed:

1. **FROM**: compute \textit{cross-product} of tables (e.g., Students and Enrolled)
2. **WHERE**: Check conditions, discard tuples that fail (applying “selection” condition)
3. **SELECT**: Delete unwanted fields (applying “projection”)
4. **DISTINCT** specified, eliminate duplicate rows

probably the least efficient way to compute a query!

Query Optimization finds the \textit{same answer} more efficiently
Remember the query and the data

```
SELECT S.name, E.cid
FROM Students S, Enrolled E
WHERE S.sid=E.sid AND E.grade='B'
```
Step 1 – Cross Product

Combine with cross-product all tables of the FROM clause.

<table>
<thead>
<tr>
<th>S.sid</th>
<th>S.name</th>
<th>S.login</th>
<th>S.age</th>
<th>S.gpa</th>
<th>E.sid</th>
<th>E.cid</th>
<th>E.grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>53666</td>
<td>Jones</td>
<td>jones@cs</td>
<td>18</td>
<td>3.4</td>
<td>53831</td>
<td>Carnatic101</td>
<td>C</td>
</tr>
<tr>
<td>53666</td>
<td>Jones</td>
<td>jones@cs</td>
<td>18</td>
<td>3.4</td>
<td>53832</td>
<td>Reggae203</td>
<td>B</td>
</tr>
<tr>
<td>53666</td>
<td>Jones</td>
<td>jones@cs</td>
<td>18</td>
<td>3.4</td>
<td>53650</td>
<td>Topology112</td>
<td>A</td>
</tr>
<tr>
<td>53666</td>
<td>Jones</td>
<td>jones@cs</td>
<td>18</td>
<td>3.4</td>
<td>53666</td>
<td>History105</td>
<td>B</td>
</tr>
<tr>
<td>53688</td>
<td>Smith</td>
<td>smith@ee</td>
<td>18</td>
<td>3.2</td>
<td>53831</td>
<td>Carnatic101</td>
<td>C</td>
</tr>
<tr>
<td>53688</td>
<td>Smith</td>
<td>smith@ee</td>
<td>18</td>
<td>3.2</td>
<td>53831</td>
<td>Reggae203</td>
<td>B</td>
</tr>
<tr>
<td>53688</td>
<td>Smith</td>
<td>smith@ee</td>
<td>18</td>
<td>3.2</td>
<td>53650</td>
<td>Topology112</td>
<td>A</td>
</tr>
<tr>
<td>53688</td>
<td>Smith</td>
<td>smith@ee</td>
<td>18</td>
<td>3.2</td>
<td>53666</td>
<td>History105</td>
<td>B</td>
</tr>
</tbody>
</table>

SELECT S.name, E.cid
FROM Students S, Enrolled E
WHERE S.sid=E.sid AND E.grade=‘B'
### Step 2 - Discard tuples that fail predicate

Make sure the **WHERE** clause is true!

<table>
<thead>
<tr>
<th>S.sid</th>
<th>S.name</th>
<th>S.login</th>
<th>S.age</th>
<th>S.gpa</th>
<th>E.sid</th>
<th>E.cid</th>
<th>E.grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>53666</td>
<td>Jones</td>
<td>jones@cs</td>
<td>18</td>
<td>3.4</td>
<td>53831</td>
<td>Carnatic101</td>
<td>C</td>
</tr>
<tr>
<td>53666</td>
<td>Jones</td>
<td>jones@cs</td>
<td>18</td>
<td>3.4</td>
<td>53832</td>
<td>Reggae203</td>
<td>B</td>
</tr>
<tr>
<td>53666</td>
<td>Jones</td>
<td>jones@cs</td>
<td>18</td>
<td>3.4</td>
<td>53650</td>
<td>Topology112</td>
<td>A</td>
</tr>
<tr>
<td>53666</td>
<td>Jones</td>
<td>jones@cs</td>
<td>18</td>
<td>3.4</td>
<td>53666</td>
<td>History105</td>
<td>B</td>
</tr>
<tr>
<td>53688</td>
<td>Smith</td>
<td>smith@ee</td>
<td>18</td>
<td>3.2</td>
<td>53831</td>
<td>Carnatic101</td>
<td>C</td>
</tr>
<tr>
<td>53688</td>
<td>Smith</td>
<td>smith@ee</td>
<td>18</td>
<td>3.2</td>
<td>53831</td>
<td>Reggae203</td>
<td>B</td>
</tr>
<tr>
<td>53688</td>
<td>Smith</td>
<td>smith@ee</td>
<td>18</td>
<td>3.2</td>
<td>53650</td>
<td>Topology112</td>
<td>A</td>
</tr>
<tr>
<td>53688</td>
<td>Smith</td>
<td>smith@ee</td>
<td>18</td>
<td>3.2</td>
<td>53666</td>
<td>History105</td>
<td>B</td>
</tr>
</tbody>
</table>

**SELECT** S.name, E.cid  
**FROM** Students S, Enrolled E  
**WHERE** S.sid=E.sid AND E.grade='B'
### Step 3 - Discard Unwanted Columns

Show only what is on the **SELECT** clause.

<table>
<thead>
<tr>
<th>S.sid</th>
<th>S.name</th>
<th>S.login</th>
<th>S.age</th>
<th>S.gpa</th>
<th>E.sid</th>
<th>E.cid</th>
<th>E.grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>53666</td>
<td>Jones</td>
<td>jones@cs</td>
<td>18</td>
<td>3.4</td>
<td>53831</td>
<td>Carnatic101</td>
<td>C</td>
</tr>
<tr>
<td>53666</td>
<td>Jones</td>
<td>jones@cs</td>
<td>18</td>
<td>3.4</td>
<td>53832</td>
<td>Reggae203</td>
<td>B</td>
</tr>
<tr>
<td>53666</td>
<td>Jones</td>
<td>jones@cs</td>
<td>18</td>
<td>3.4</td>
<td>53650</td>
<td>Topology112</td>
<td>A</td>
</tr>
<tr>
<td>53666</td>
<td>Jones</td>
<td>jones@cs</td>
<td>18</td>
<td>3.4</td>
<td>53666</td>
<td>History105</td>
<td>B</td>
</tr>
<tr>
<td>53688</td>
<td>Smith</td>
<td>smith@ee</td>
<td>18</td>
<td>3.2</td>
<td>53831</td>
<td>Carnatic101</td>
<td>C</td>
</tr>
<tr>
<td>53688</td>
<td>Smith</td>
<td>smith@ee</td>
<td>18</td>
<td>3.2</td>
<td>53831</td>
<td>Reggae203</td>
<td>B</td>
</tr>
<tr>
<td>53688</td>
<td>Smith</td>
<td>smith@ee</td>
<td>18</td>
<td>3.2</td>
<td>53650</td>
<td>Topology112</td>
<td>A</td>
</tr>
<tr>
<td>53688</td>
<td>Smith</td>
<td>smith@ee</td>
<td>18</td>
<td>3.2</td>
<td>53666</td>
<td>History105</td>
<td>B</td>
</tr>
</tbody>
</table>

**SELECT** S.name, E.cid  
**FROM** Students S, Enrolled E  
**WHERE** S.sid=E.sid AND E.grade='B'
Now the Details...

We will use these instances of relations in our examples.

<table>
<thead>
<tr>
<th>sid</th>
<th>bid</th>
<th>day</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>101</td>
<td>10/10/16</td>
</tr>
<tr>
<td>95</td>
<td>103</td>
<td>11/12/16</td>
</tr>
</tbody>
</table>

<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>95</td>
<td>Bob</td>
<td>3</td>
<td>63.5</td>
</tr>
</tbody>
</table>

<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>
## Another Join Query

```sql
SELECT sname
FROM Sailors, Reserves
WHERE Sailors.sid = Reserves.sid AND 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>10/10/16</td>
</tr>
<tr>
<td>22</td>
<td>dustin</td>
<td>7</td>
<td>45.0</td>
<td>95</td>
<td>103</td>
<td>11/12/16</td>
</tr>
<tr>
<td>31</td>
<td>lubber</td>
<td>8</td>
<td>55.5</td>
<td>22</td>
<td>101</td>
<td>10/10/16</td>
</tr>
<tr>
<td>31</td>
<td>lubber</td>
<td>8</td>
<td>55.5</td>
<td>95</td>
<td>103</td>
<td>11/12/16</td>
</tr>
<tr>
<td>95</td>
<td>Bob</td>
<td>3</td>
<td>63.5</td>
<td>22</td>
<td>101</td>
<td>10/10/16</td>
</tr>
<tr>
<td>95</td>
<td>Bob</td>
<td>3</td>
<td>63.5</td>
<td>95</td>
<td>103</td>
<td>11/12/16</td>
</tr>
</tbody>
</table>
Range Variables

can associate “range variables” with the tables in the FROM clause
a shorthand, like the rename operator from relational algebra
saves writing, makes queries easier to understand
“FROM Sailors, Reserves”
“FROM Sailors S, Reserves R”

needed when ambiguity could arise
for example, if same table used multiple times in same FROM (called a “self-join”)
“FROM Sailors s1, Sailors s2”
Range Variables

```
SELECT sname
FROM Sailors, Reserves
WHERE Sailors.sid = Reserves.sid AND bid = 103
```
can be rewritten using range variables as:

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

an example requiring range variables (self-join)

```sql
SELECT s1.sname, s1.age, s2.sname, s2.age
FROM Sailors s1, Sailors s2
WHERE s1.age > s2.age
```

another one: "*" if you don’t want a projection:

```sql
SELECT *
FROM Sailors s
WHERE s.age > 20
```
Find sailors who have reserved at least one boat

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

Does DISTINCT 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

Can use arithmetic expressions in SELECT clause
(plus other operations we’ll discuss later)

Use AS to provide column names

```
SELECT S.age, S.age-5 AS age1, 2*S.age AS age2
FROM   Sailors S
WHERE  S.sname = 'dustin'
```

Can also have expressions in WHERE clause:

```
SELECT   S1.sname AS name1, S2.sname AS name2
FROM     Sailors S1, Sailors S2
WHERE    2*S1.rating = S2.rating - 1
```
String operations

SQL also supports some string operations

“LIKE” is used for string matching.

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

’_’ stands for any one character

’%’ stands for 0 or more arbitrary characters
More Operations

SQL queries produce new tables

If the results of two queries are **union-compatible** (same number and types of columns) then we can apply logical operations

- **UNION**
- **INTERSECTION**
- **SET DIFFERENCE** (called **EXCEPT** or **MINUS**)
Find sids of sailors who have 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)

```
SELECT R.sid
FROM Boats B, Reserves R
WHERE R.bid=B.bid AND
  (B.color='red' OR B.color='green')
```

**VS.**

```
SELECT R.sid
FROM Boats B, Reserves R
WHERE R.bid=B.bid AND B.color='red'
UNION
SELECT R.sid
FROM Boats B, Reserves R
WHERE R.bid=B.bid AND B.color='green'
```
Find sids of sailors who have reserved a red and a green boat

If we simply replace OR by AND in the previous query, we get the wrong answer. (Why?)

Instead, could use a self-join:

```sql
SELECT R1.sid
FROM Boats B1, Reserves R1,
     Boats B2, Reserves R2
WHERE R1.sid = R2.sid
    AND R1.bid = B1.bid
    AND R2.bid = B2.bid
    AND (B1.color = 'red' AND B2.color = 'green')
```
INTERSECT: discussed in the book. Can be used to compute the intersection of any two union-compatible sets of tuples.

Also in text: EXCEPT (sometimes called MINUS)

Included in the SQL/92 standard, but some systems do not support them.

Key field!
Your turn …

1. Find (the names of) all sailors who are over 50 years old
2. Find (the names of) all boats that have been reserved at least once
3. Find all sailors who have not reserved a red boat (hint: use “EXCEPT”)
4. Find all pairs of same-color boats
5. Find all pairs of sailors in which the older sailor has a lower rating
Answers ...

1. Find (the names of) all sailors who are over 50 years old

```sql
SELECT S.sname
FROM   Sailors S
WHERE  S.age > 50
```
Answers ...

2. Find (the names of) all boats that have been reserved at least once

```sql
SELECT DISTINCT B.bname
FROM Boats B, Reserves R
WHERE R.bid=B.bid
```
Answers ...

3. Find all sailors who have not reserved a red boat

```
SELECT S.sid  
FROM   Sailors S  
EXCEPT  
SELECT R.sid  
FROM   Boats B,Reserves R  
WHERE    R.bid=B.bid  
AND      B.color='red'
```
4. Find all pairs of same-color boats

\[
\text{SELECT B1.bname, B2.bname}
\text{FROM Boats B1, Boats B2}
\text{WHERE B1.color = B2.color}
\text{AND B1.bid < B2.bid}
\]
5. Find all pairs of sailors in which the **older** sailor has a **lower** rating

```
SELECT s1.sname, s2.sname
FROM Sailors s1, Sailors s2
WHERE s1.age > s2.age
AND s1.rating < s2.rating
```
Nested Queries

powerful feature of SQL:

WHERE clause can itself contain an SQL query!

Actually, so can FROM and HAVING clauses.

Names of sailors who have reserved boat #103

```sql
SELECT S.sname
FROM Sailors S
WHERE S.sid IN (SELECT R.sid
                FROM Reserves R
                WHERE R.bid=103)
```
Nested Queries

to find sailors who have 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 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)
```

**EXISTS** is another set operator, like **IN** (also **NOT EXISTS**)

If **EXISTS UNIQUE** is used, and * is replaced by `R.bid`, finds sailors with at most one reservation for boat #103.

**UNIQUE** checks for duplicate tuples in a subquery;

Subquery must be recomputed for each Sailors tuple.

Think of subquery as a function call that runs a query!
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: \textit{op ANY}, \textit{op ALL}

Find sailors whose rating is greater than that of some sailor called Horatio:

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

Find sids of sailors who have reserved both a red and a green boat

```sql
SELECT R.sid
FROM Boats B, Reserves R
WHERE R.bid=B.bid
  AND B.color='red'
  AND R.sid IN (SELECT R2.sid
                 FROM Boats B2, Reserves R2
                 WHERE R2.bid=B2.bid
                 AND B2.color='green')
```

Similarly, EXCEPT queries can be re-written using NOT IN.

How would you change this to find names (not sids) of Sailors who’ve reserved both red and green boats?
Query #3 revisited ...

3. Find all sailors who have **not** reserved a red boat
   (this time, without using “EXCEPT”)
3. Find all sailors who have **not** reserved a red boat

```sql
SELECT S.sid
FROM Sailors S
WHERE S.sid NOT IN
  (SELECT R.sid
   FROM Reserves R, Boats B
   WHERE R.bid = B.bid
     AND B.color = 'red')
```
Another Correct Answer ...

3. Find all sailors who have not reserved a red boat

```sql
SELECT S.sid
FROM Sailors S
WHERE NOT EXISTS
  (SELECT *
   FROM Reserves R, Boats B
   WHERE R.sid = S.sid
     AND R.bid = B.bid
     AND B.color = 'red')
```
Division in SQL

Find sailors who have reserved all boats.

Sailors $S$ for which ...

```
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))
```

there is no boat $B$ without ...

Sailors $S$ for which ...

```
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))
```

a Reserves tuple AND $R.sid=S.sid)

showing $S$ reserved $B$
Aggregate Operators

Significant extension of relational algebra.

```
SELECT COUNT (*)
FROM Sailors S

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

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

```
SELECT  S.sname
FROM    Sailors S
WHERE   S.rating = (SELECT  MAX(S2.rating)
                      FROM    Sailors S2)

SELECT  AVG (DISTINCT S.age)
FROM    Sailors S
WHERE   S.rating=10
```
Find name and age of the oldest sailor(s)

The first query is incorrect!

Third query equivalent to second query
allowed in SQL/92 standard, but not supported in some systems.

```
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
```