CS460: Intro to Database Systems

Class 6: SQL, The Query Language – Part I

Instructor: Manos Athanassoulis

https://bu-disc.github.io/CS460/
Relational Algebra

Relational Query Languages
Selection & Projection
Union, Set Difference & Intersection
Cross product & Joins
Examples
Division

From Previous Class
Last Compound Operator: Division

useful for expressing “for all” queries like:
“find sids of sailors who have reserved all boats”

for A/B attributes of B are subset of attributes of A
may need to “project” to make this happen.
e.g., let A have 2 fields, x and y ; B have only field y :

\[ A/B = \{ \langle x \rangle | \forall \langle y \rangle \in B(\exists \langle x, y \rangle \in A) \} \]

A/B contains all x tuples such that for every y tuple in B, there is an xy tuple in A
Examples of Division A/B

A

<table>
<thead>
<tr>
<th>sno</th>
<th>pno</th>
</tr>
</thead>
<tbody>
<tr>
<td>s1</td>
<td>p1</td>
</tr>
<tr>
<td>s1</td>
<td>p2</td>
</tr>
<tr>
<td>s1</td>
<td>p3</td>
</tr>
<tr>
<td>s1</td>
<td>p4</td>
</tr>
<tr>
<td>s2</td>
<td>p1</td>
</tr>
<tr>
<td>s2</td>
<td>p2</td>
</tr>
<tr>
<td>s3</td>
<td>p2</td>
</tr>
<tr>
<td>s4</td>
<td>p2</td>
</tr>
<tr>
<td>s4</td>
<td>p4</td>
</tr>
</tbody>
</table>

A/B1

<table>
<thead>
<tr>
<th>pno</th>
</tr>
</thead>
<tbody>
<tr>
<td>p2</td>
</tr>
</tbody>
</table>

B1

<table>
<thead>
<tr>
<th>pno</th>
</tr>
</thead>
<tbody>
<tr>
<td>p2</td>
</tr>
<tr>
<td>p4</td>
</tr>
</tbody>
</table>

B2

<table>
<thead>
<tr>
<th>sno</th>
</tr>
</thead>
<tbody>
<tr>
<td>s1</td>
</tr>
<tr>
<td>s2</td>
</tr>
<tr>
<td>s3</td>
</tr>
<tr>
<td>s4</td>
</tr>
</tbody>
</table>

B3

<table>
<thead>
<tr>
<th>pno</th>
</tr>
</thead>
<tbody>
<tr>
<td>p1</td>
</tr>
<tr>
<td>p2</td>
</tr>
<tr>
<td>p4</td>
</tr>
</tbody>
</table>
Examples of Division A/B

A

<table>
<thead>
<tr>
<th>sno</th>
<th>pno</th>
</tr>
</thead>
<tbody>
<tr>
<td>s1</td>
<td>p1</td>
</tr>
<tr>
<td>s1</td>
<td>p2</td>
</tr>
<tr>
<td>s1</td>
<td>p3</td>
</tr>
<tr>
<td>s1</td>
<td>p4</td>
</tr>
<tr>
<td>s2</td>
<td>p1</td>
</tr>
<tr>
<td>s2</td>
<td>p2</td>
</tr>
<tr>
<td>s3</td>
<td>p2</td>
</tr>
<tr>
<td>s4</td>
<td>p2</td>
</tr>
<tr>
<td>s4</td>
<td>p4</td>
</tr>
</tbody>
</table>

A/B1

<table>
<thead>
<tr>
<th>pno</th>
</tr>
</thead>
<tbody>
<tr>
<td>p1</td>
</tr>
<tr>
<td>p2</td>
</tr>
</tbody>
</table>

B1

<table>
<thead>
<tr>
<th>pno</th>
</tr>
</thead>
<tbody>
<tr>
<td>p2</td>
</tr>
<tr>
<td>p4</td>
</tr>
</tbody>
</table>

B2

A/B2

<table>
<thead>
<tr>
<th>pno</th>
</tr>
</thead>
<tbody>
<tr>
<td>p1</td>
</tr>
<tr>
<td>p2</td>
</tr>
<tr>
<td>p4</td>
</tr>
</tbody>
</table>

B3
Examples of Division A/B

A

<table>
<thead>
<tr>
<th>sno</th>
<th>pno</th>
</tr>
</thead>
<tbody>
<tr>
<td>s1</td>
<td>p1</td>
</tr>
<tr>
<td>s1</td>
<td>p2</td>
</tr>
<tr>
<td>s1</td>
<td>p3</td>
</tr>
<tr>
<td>s1</td>
<td>p4</td>
</tr>
<tr>
<td>s2</td>
<td>p1</td>
</tr>
<tr>
<td>s2</td>
<td>p2</td>
</tr>
<tr>
<td>s3</td>
<td>p2</td>
</tr>
<tr>
<td>s4</td>
<td>p2</td>
</tr>
<tr>
<td>s4</td>
<td>p4</td>
</tr>
</tbody>
</table>

A/B1

<table>
<thead>
<tr>
<th>pno</th>
</tr>
</thead>
<tbody>
<tr>
<td>p1</td>
</tr>
<tr>
<td>p2</td>
</tr>
</tbody>
</table>

B1

<table>
<thead>
<tr>
<th>pno</th>
</tr>
</thead>
<tbody>
<tr>
<td>p2</td>
</tr>
<tr>
<td>p4</td>
</tr>
</tbody>
</table>

B2

<table>
<thead>
<tr>
<th>sno</th>
</tr>
</thead>
<tbody>
<tr>
<td>s1</td>
</tr>
<tr>
<td>s2</td>
</tr>
<tr>
<td>s3</td>
</tr>
<tr>
<td>s4</td>
</tr>
</tbody>
</table>

B3

<table>
<thead>
<tr>
<th>sno</th>
</tr>
</thead>
<tbody>
<tr>
<td>s1</td>
</tr>
</tbody>
</table>

A/B3
Expressing A/B Using Basic Operators

division is not essential op; just a shorthand
(true for joins, but so common that are implemented specially)

Idea: For A/B, compute all x values that are not “disqualified” by some y value in B

x value is disqualified if by attaching y value from B, we obtain an xy tuple that is not in A

Disqualified x values: \( \pi_x ((\pi_x(A) \times B) - A) \)

\( A/B: \pi_x(A) - \text{Disqualified x values} \)
Expressing \( A/B: \quad \pi_{sno}(A) - \pi_{sno}(\pi_{sno}(A) \times B) - A \)

\[
\begin{array}{c|c}
\text{sno} & \text{pno} \\
\hline
s1 & p1 \\
s1 & p2 \\
s1 & p3 \\
s1 & p4 \\
s2 & p1 \\
s2 & p2 \\
s3 & p2 \\
s4 & p2 \\
s4 & p4 \\
\end{array}
\]

\[
\begin{array}{c|c}
\text{sno} & \text{pno} \\
\hline
s1 & p1 \\
s1 & p2 \\
s1 & p4 \\
s2 & p1 \\
s2 & p2 \\
s2 & p4 \\
s3 & p1 \\
s3 & p2 \\
s3 & p4 \\
s4 & p1 \\
s4 & p2 \\
s4 & p4 \\
\end{array}
\]

\[
A
\]

\[
T1 = \pi_{sno}(A) \times B
\]

\[
\begin{array}{c}
\text{sno} \\
\hline
s1 \\
s2 \\
s3 \\
s4 \\
\end{array}
\]

\[
\begin{array}{c}
pno \\
\hline
p1 \\
p2 \\
p3 \\
p4 \\
\end{array}
\]

\[
\leftarrow \times B
\]
Expressing A/B: $\pi_{sno}(A) - \pi_{sno}((\pi_{sno}(A) \times B) - A)$

\[ T1 = \pi_{sno}(A) \times B \]

\[ T2 = \pi_{sno}(T1 - A) \]
Expressing $A/B$: $\pi_{sno}(A) - \pi_{sno}((\pi_{sno}(A) \times B) - A)$

\[
\begin{align*}
T_1 &= \pi_{sno}(A) \\
T_2 &= \pi_{sno}(T_1 - A)
\end{align*}
\]
Find the names of sailors who have reserved all boats

use division; schemas of the input relations to / must be carefully chosen (why?)

\[
\rho \left( \text{Tempsids}, \left( \pi_{\text{sid}, \text{bid}} \text{Reserves} \right) / \left( \pi_{\text{bid}} \text{Boats} \right) \right)
\]

\[
\pi_{\text{sname}} \left( \text{Tempsids} \bowtie \text{Sailors} \right)
\]

To find sailors who have reserved all "Interlake" boats:

\[
\ldots / \pi_{\text{bid}} \left( \sigma_{\text{bname}=\text{'Interlake'}} \text{Boats} \right)
\]
Find the names of sailors who have reserved all boats

use division; schemas of the input relations to / must be carefully chosen (why?)

\[ \rho \left( \text{Tempsids}, \left( \pi_{\text{sid,bid}} \text{Reserves} \right) / \left( \pi_{\text{bid}} \text{Boats} \right) \right) \]

\[ \pi_{\text{sname}} (\text{Tempsids} \bowtie \text{Sailors}) \]

\[ \text{what if we divided } \text{Reserves} / \pi_{\text{bid}} (\text{Boats}) ? \]

this would return the pairs of (sid, date) that have a value for every boat, i.e., the sids that rented every boat, every day they made any reservation!!!! Not so useful!
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)

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

<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>

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”

```sql
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:

```sql
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>

displayed as: “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:

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

result:

<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”)

```
SELECT  [DISTINCT]  target-list
FROM     relation-list
WHERE    qualification
```
Query Semantics

Conceptually, a SQL query can be computed:

1. **FROM**: compute **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. (4) if **DISTINCT** specified, eliminate duplicate rows

probably the least efficient way to compute a query!

Query Optimization finds the **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'
```

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

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

can be rewritten using range variables as:

```sql
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)

```
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:

```
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

```sql
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:

```sql
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)

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

VS.

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

```
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

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

```
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

```
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'
```
Answers ...

4. Find all pairs of same-color boats

SELECT B1.bname, B2.bname
FROM   Boats B1, Boats B2
WHERE  B1.color = B2.color
Answers ...

5. Find all pairs of sailors in which the **older** sailor has a **lower** rating

\[
\begin{align*}
\text{SELECT} & \quad S1.sname, S2.sname \\
\text{FROM} & \quad \text{Sailors } S1, \text{ Sailors } S2 \\
\text{WHERE} & \quad S1.age > S2.age \\
& \quad \text{AND } S1.rating < S2.rating
\end{align*}
\]
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

```
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*

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

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

If *EXISTS UNIQUE* is used, and *is replaced by* \(R\text{.bid}\), 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, 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”)*
Answer ...

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

```
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

```
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))
```

*a Reserves tuple 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'

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

single column
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

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

single column
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, 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
```