This lecture continues queries in relational algebra.

Example 2

Find the names of the sailors who have reserved a red boat

\[ \pi_{\text{sname}}((\sigma_{\text{colour}=\text{\textquoteleft red\textquoteright}} \text{\text{boat}}) \bowtie \text{reserves} \bowtie \text{sailor}) \]

This query proceeds as follows:

- First select the rows corresponding to red boats from \text{boat}.
- Next form the natural join of that table with \text{reserves} to find all the information about reservations involving red boats.
- Then form the natural join of that relation with \text{sailor} to join the personal information about the sailors.
- Finally project out the sailor’s name.

Step 1

We can execute this step-by-step in MySQL to see what happens:

\[ \sigma_{\text{colour}=\text{\textquoteleft red\textquoteright}} \text{\text{boat}} \]

<table>
<thead>
<tr>
<th>bid</th>
<th>name</th>
<th>colour</th>
</tr>
</thead>
<tbody>
<tr>
<td>102</td>
<td>Interlake</td>
<td>red</td>
</tr>
<tr>
<td>104</td>
<td>Marine</td>
<td>red</td>
</tr>
</tbody>
</table>
Step 2

\[(\sigma_{\text{colour}=\text{'red'}\ \text{boat}}) \bowtie \text{reserves}\]

<table>
<thead>
<tr>
<th>bid</th>
<th>name</th>
<th>colour</th>
<th>sid</th>
<th>date</th>
</tr>
</thead>
<tbody>
<tr>
<td>102</td>
<td>Interlake</td>
<td>red</td>
<td>22</td>
<td>2006-08-10</td>
</tr>
<tr>
<td>104</td>
<td>Marine</td>
<td>red</td>
<td>22</td>
<td>2006-08-12</td>
</tr>
<tr>
<td>102</td>
<td>Interlake</td>
<td>red</td>
<td>31</td>
<td>2006-08-02</td>
</tr>
<tr>
<td>104</td>
<td>Marine</td>
<td>red</td>
<td>31</td>
<td>2006-08-17</td>
</tr>
<tr>
<td>102</td>
<td>Interlake</td>
<td>red</td>
<td>64</td>
<td>2006-08-18</td>
</tr>
<tr>
<td>102</td>
<td>Interlake</td>
<td>red</td>
<td>64</td>
<td>2006-08-05</td>
</tr>
</tbody>
</table>

Notice that the natural join (\(\bowtie\)) has removed the second occurrence of the join column bid.


Step 3

\[(\sigma_{\text{colour}=\text{'red'}\ \text{boat}}) \bowtie \text{reserves} \bowtie \text{sailor}\]

<table>
<thead>
<tr>
<th>bid</th>
<th>name</th>
<th>colour</th>
<th>sid</th>
<th>date</th>
<th>sname</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>102</td>
<td>Interlake</td>
<td>red</td>
<td>22</td>
<td>2006-08-10</td>
<td>Dustin</td>
<td>45</td>
</tr>
<tr>
<td>104</td>
<td>Marine</td>
<td>red</td>
<td>22</td>
<td>2006-08-12</td>
<td>Dustin</td>
<td>45</td>
</tr>
<tr>
<td>102</td>
<td>Interlake</td>
<td>red</td>
<td>31</td>
<td>2006-08-02</td>
<td>Lubber</td>
<td>55.5</td>
</tr>
<tr>
<td>104</td>
<td>Marine</td>
<td>red</td>
<td>31</td>
<td>2006-08-17</td>
<td>Lubber</td>
<td>55.5</td>
</tr>
<tr>
<td>102</td>
<td>Interlake</td>
<td>red</td>
<td>64</td>
<td>2006-08-18</td>
<td>Horatio</td>
<td>35</td>
</tr>
<tr>
<td>102</td>
<td>Interlake</td>
<td>red</td>
<td>64</td>
<td>2006-08-05</td>
<td>Horatio</td>
<td>35</td>
</tr>
</tbody>
</table>

This has now joined the sailor relation using the sid field and removed the second occurrence of this field.


Finally

\[\pi_{\text{sname}}((\sigma_{\text{colour}=\text{'red'}\ \text{boat}}) \bowtie \text{reserves} \bowtie \text{sailor})\]

<table>
<thead>
<tr>
<th>sname</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dustin</td>
</tr>
<tr>
<td>Lubber</td>
</tr>
<tr>
<td>Horatio</td>
</tr>
</tbody>
</table>

This final step projects the column sname and removes the duplicates.

(GF Royle 2006-8, N Spadaccini 2008) Databases - Relational Algebra II 7 / 27

In MySQL

In MySQL we can make this query in our usual way by performing the joins “manually”.

```
SELECT DISTINCT S.sname
FROM boat B, reserves R, sailor S
WHERE
    B.colour = 'red' AND
    B.bid = R.bid AND
    R.sid = S.sid;
```

In this query, the two joins are performed by the two conditions B.bid = R.bid and R.sid = S.sid while the selection of the red boat is the first condition.

(GF Royle 2006-8, N Spadaccini 2008) Databases - Relational Algebra II 8 / 27
**Queries**

**Joins in MySQL**

MySQL allows you to specify various joins, including natural joins, directly.

\[
\text{SELECT DISTINCT S.sname} \\
\text{FROM boat B NATURAL JOIN reserves NATURAL JOIN sailor S} \\
\text{WHERE B.colour = 'red';}
\]

Notice that seeing `reserves` is just used to join `boat` and `sailor`, there was no need to give it an alias.

---

**Renaming**

It is often convenient to assign a name to one of the “intermediate” tables in a relational algebra expression.

If \( E(R, S, T) \) is any relational algebra expression involving relations \( R, S, T \) for example, then

\[
\rho(A, E(R, S, T))
\]

means “create a new relation called \( A \) by evaluating the expression \( E(R, S, T) \).

In R&G, they introduce some (ugly) notation for renaming the columns of an intermediate relation, but we will not use that.

\[
\rho(\text{redboat}, \sigma\text{colour} = 'red' \text{boat})
\]

---

**Example 3**

Find the names of the sailors who have hired a red or a green boat

\[
\rho(\text{tempboat}, \sigma\text{colour} = 'red' \lor \text{colour} = 'green' \text{boat}) \\
\pi\text{sname}(\text{tempboat} \bowtie \text{reserves} \bowtie \text{sailor})
\]

---

**In MySQL**

We can perform this process exactly like this in MySQL if desired, but at the expense of creating two new tables.

\[
\text{CREATE TEMPORARY TABLE tempboat LIKE boat;}
\]

\[
\text{INSERT INTO tempboat} \\
\text{(SELECT * FROM boat WHERE colour = 'red' \lor colour = 'green');}
\]

\[
\text{SELECT DISTINCT S.sname} \\
\text{FROM tempboat NATURAL JOIN reserves NATURAL JOIN sailor S;}
\]

A temporary table will be removed when the client disconnects from the server.
Queries

Alternatively

An alternative in this case is to find the sailors who have used red boats and green boats in two separate queries.

\[
\begin{align*}
\text{SELECT } & \text{S.sname} \\
\text{FROM } & \text{boat B NATURAL JOIN reserves NATURAL JOIN sailor S} \\
\text{WHERE } & \text{B.colour = 'red'} \\
\text{UNION} & \\
\text{SELECT } & \text{S.sname} \\
\text{FROM } & \text{boat B NATURAL JOIN reserves NATURAL JOIN sailor S} \\
\text{WHERE } & \text{B.colour = 'green'};
\end{align*}
\]

\[
\pi_{\text{sname}}((\sigma_{\text{colour='red'}}\bowtie\text{reserves} \bowtie \text{sailor}) \\
\cup \\
\pi_{\text{sname}}((\sigma_{\text{colour='green'}}\bowtie\text{reserves} \bowtie \text{sailor}))
\]


Queries

A red boat AND a green boat

Things get more interesting when we try to answer

*Which sailors have hired both a red boat and a green boat*

We cannot just replace \( \text{OR} \) with \( \text{AND} \) to get

\[
\rho(\text{tempboat}, \sigma_{\text{colour='red'}} \land \text{colour='green'}}\bowtie\text{boat})
\]

\[
\pi_{\text{sname}}(\text{tempboat} \bowtie\text{reserves} \bowtie \text{sailor})
\]

because this query returns *no results* — there are no boats that are both red and green!

(GF Royle 2006-8, N Spadaccini 2008) Databases - Relational Algebra II 14 / 27

Queries

Intersection

In relational algebra we can frame this query quite naturally by using *intersection* instead of *union*.

\[
\pi_{\text{sname}}((\sigma_{\text{colour='red'}}\bowtie\text{reserves} \bowtie \text{sailor}) \\
\cap \\
\pi_{\text{sname}}((\sigma_{\text{colour='green'}}\bowtie\text{reserves} \bowtie \text{sailor}))
\]

Unfortunately, MySQL 5.0.x does not support an *intersection* operator so this cannot be translated directly into MySQL.

(GF Royle 2006-8, N Spadaccini 2008) Databases - Relational Algebra II 15 / 27

Queries

Two boats

A relational algebra query that *can* be translated directly into MySQL uses the concept of *two boats* reserved by the same sailor.

\[
\rho(R1, \sigma_{\text{sid,bid}}(\sigma_{\text{colour='red'}}\bowtie\text{boat} \bowtie \text{reserves}))
\]

\[
\rho(R2, \sigma_{\text{sid,bid}}(\sigma_{\text{colour='green'}}\bowtie\text{boat} \bowtie \text{reserves}))
\]

\[
\pi_{\text{sname}}(\text{sailor} \bowtie (R1 \bowtie R1.\text{sid}=R2.\text{sid} R2))
\]

Here \( R1 \) is a list of “red-boat reservations” and \( R2 \) is a list of “green-boat reservations”. Notice that we cannot perform a *natural join* on \( R1 \) and \( R2 \) in this situation because both relations have a field called `bid` that we do not want to join on.

(GF Royle 2006-8, N Spadaccini 2008) Databases - Relational Algebra II 16 / 27
In MySQL

This translates into MySQL as

```sql
SELECT DISTINCT S.sname
FROM sailor S, reserves R1, reserves R2, boat B1, boat B2
WHERE R1.bid = B1.bid AND B1.colour = 'red'
AND R2.bid = B2.bid AND B2.colour = 'green'
AND R1.sid = S.sid AND R2.sid = S.sid;
```

We can view this query as finding two boat-reservations \((B_1, R_1)\) and \((B_2, R_2)\) that prove that a given sailor has reserved a red boat and also a green boat.

The division operator

The division operator is an interesting operator that is useful in answering queries that involve “for all” statements.

Consider two relations \(A\) and \(B\) where \(A\) has two columns \(D_x\) and \(D_y\) and \(B\) has a single column \(D_y\).

If we “divide” \(A\) by \(B\) then the resulting relation \(Q = A/B\) has the single column \(D_x\) and is defined as follows:

\[ Q \text{ has a row with value } x \text{ if and only if } (x,y) \in A \text{ for all } y \in B \]

Why division?

What is the motivation for the name “division”?

If we consider integer division, then we could define the quotient \(q = a/b\) by

“the quotient \(q\) of two positive integers \(a\) and \(b\) is the largest integer such that \(qb \leq a\).”

In relational algebra, the quotient relation \(Q = A/B\) of two relations \(A\) and \(B\) is the maximal relation such that

\[ Q \times B \subseteq A. \]
**Queries with division**

*Find the names of the sailors who have reserved all the boats*

\[ \rho(\text{temp}, (\pi_{\text{sid, bid, reserves}}/\pi_{\text{bid, boat}})) \]

\[ \pi_{\text{sname}}(\text{temp} \Join \text{sailor}) \]

The presence of the phrase *all the* or *every* is usually a give-away that the division operator should be used.

---

**Re-expressing division**

Suppose that \( A \) and \( B \) are defined as above. Then

\[ \pi_{Dx}(A) \]

contains all the \( x \)-values that appear in \( A \). Which of these should be in the quotient relation \( Q = A/B? \)

An \( x \)-value does *not* appear in \( Q \) if there is some \( y \) value in \( B \) such that \((x, y)\) is not in \( A \). This latter relation is given by

\[ (\pi_{Dx}(A) \times B) - A \]

---

**More Joins**

The *joins* that we have seen so far are all of the form

\[ A \Join_k B \]

which are examples of *inner joins*.

The only rows in the join are those where a row of \( A \) matches a row of \( B \) according to the join condition. However for some applications it is useful for the join to have a row for *every row of \( A \)* even if there is no matching row in \( B \).

A *left outer join* of \( A \) and \( B \) with join condition \( c \) has two types of row:

- Rows consisting of a row of \( A \) combined with a row of \( B \) where the join condition is satisfied
- Rows consisting of a row of \( A \) combined with a number of NULL fields if that row of \( A \) would not occur otherwise.
Which sailors have no reservations?

An example where this is useful is where we want to find out which sailors have no corresponding reservations.

SELECT * FROM sailor LEFT JOIN reserves
  ON sailor.sid = reserves.sid;

Here the command specifies a LEFT JOIN and explicitly gives the join condition.

Result

<table>
<thead>
<tr>
<th>sid</th>
<th>sname</th>
<th>age</th>
<th>sid</th>
<th>bid</th>
<th>date</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>Dustin</td>
<td>45</td>
<td>22</td>
<td>101</td>
<td>2006-08-10</td>
</tr>
<tr>
<td>22</td>
<td>Dustin</td>
<td>45</td>
<td>22</td>
<td>102</td>
<td>2006-08-10</td>
</tr>
<tr>
<td>22</td>
<td>Dustin</td>
<td>45</td>
<td>22</td>
<td>103</td>
<td>2006-08-11</td>
</tr>
<tr>
<td>22</td>
<td>Dustin</td>
<td>45</td>
<td>22</td>
<td>104</td>
<td>2006-08-12</td>
</tr>
<tr>
<td>29</td>
<td>Brutus</td>
<td>33</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>31</td>
<td>Lubber</td>
<td>55.5</td>
<td>31</td>
<td>102</td>
<td>2006-08-02</td>
</tr>
<tr>
<td>31</td>
<td>Lubber</td>
<td>55.5</td>
<td>31</td>
<td>103</td>
<td>2006-08-03</td>
</tr>
<tr>
<td>31</td>
<td>Lubber</td>
<td>55.5</td>
<td>31</td>
<td>104</td>
<td>2006-08-17</td>
</tr>
<tr>
<td>32</td>
<td>Andy</td>
<td>25.5</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>58</td>
<td>Rusty</td>
<td>35</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>64</td>
<td>Horatio</td>
<td>35</td>
<td>64</td>
<td>102</td>
<td>2006-08-18</td>
</tr>
<tr>
<td>64</td>
<td>Horatio</td>
<td>35</td>
<td>64</td>
<td>102</td>
<td>2006-08-05</td>
</tr>
<tr>
<td>71</td>
<td>Zorba</td>
<td>16</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>74</td>
<td>Horatio</td>
<td>35</td>
<td>74</td>
<td>103</td>
<td>2006-08-05</td>
</tr>
<tr>
<td>85</td>
<td>Art</td>
<td>25.5</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>95</td>
<td>Bob</td>
<td>63.5</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
</tr>
</tbody>
</table>

To check those with no reservations, we check that the second occurrence of the field sid is NULL in the joined table.

SELECT sname FROM sailor LEFT JOIN reserves
  ON sailor.sid = reserves.sid
  WHERE reserves.sid IS NULL;

+--------+
<table>
<thead>
<tr>
<th>sname</th>
</tr>
</thead>
<tbody>
<tr>
<td>Brutus</td>
</tr>
<tr>
<td>Andy</td>
</tr>
<tr>
<td>Rusty</td>
</tr>
<tr>
<td>Zorba</td>
</tr>
<tr>
<td>Art</td>
</tr>
<tr>
<td>Bob</td>
</tr>
</tbody>
</table>
+--------+