Surprise! (🤔)

- These are the slides for our unplanned special episode of “Friday Nights With Databases”.
- The good news is that you are now ideally positioned to tackle HW #4, on the Relational Algebra, from start to finish!
- Note: There will be NO change in HW #4’s due date, as today’s “oops” didn’t affect the available working time between its release and its deadline.
Joins

- **Condition Join**: \( R \bowtie_c S = \sigma_c (R \times S) \)

<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>58</td>
<td>58</td>
<td>101</td>
</tr>
<tr>
<td>31</td>
<td>rubber</td>
<td>8</td>
<td>55.5</td>
<td>58</td>
<td>103</td>
<td>11/12/96</td>
</tr>
</tbody>
</table>

\( S \bowtie_{sid<sid} R \)

- **Result schema** same as that of cross-product.
- Fewer tuples than cross-product, so might be able to compute more efficiently
- Sometimes (often!) called a *theta-join*.

More Joins

- **Equi-Join**: A special case of condition join where the condition \( c \) contains only *equalities*.

<table>
<thead>
<tr>
<th>sid</th>
<th>sname</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>101</td>
<td>10/10/96</td>
</tr>
<tr>
<td>58</td>
<td>rusty</td>
<td>10</td>
<td>35.0</td>
<td>103</td>
<td>11/12/96</td>
</tr>
</tbody>
</table>

\( S \bowtie_{sid} R \)

- **Result schema** similar to cross-product, but only one copy of fields for which equality is specified.
- **Natural Join**: An equijoin on *all* commonly named fields.
Division

- Not a primitive operator, but extremely useful for expressing queries like:
  
  \textit{Find sailors who have reserved all boats.}
  
- Let A have 2 fields, x and y, while B has one field y, so we have relations A(x,y) and B(y):
  
  - \( A/B \) contains the x tuples (e.g., sailors) such that for every y tuple (e.g., boat) in B, there is an xy tuple in A.
  
  - Or: If the set of y values (boats) associated with an x value (sailor) in A contains all y values in B, the x value is in \( A/B \).
  
- In general, x and y can be any lists of fields; y is the list of fields in B, and \( x \cup y \) is the list of fields of A.

Examples of Division A/B

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

\[ A \]

\[ A/B1 \]

\[ A/B2 \]

\[ A/B3 \]
Expressing A/B Using Basic Operators
(Advanced Topic 🎉)

- Division not an essential op; just a useful shorthand. (Also true of joins, but joins are so common and important that relational database systems implement joins 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 a y value from B, we obtain an xy tuple that does not appear in A.
  
  Disqualified x values (D): \[ \pi_x (\pi_x (A \times B) - A) \]

  \[ A/B: \pi_x (A) - D \]

Ex: Wisconsin Sailing Club Database

<table>
<thead>
<tr>
<th>Sailors</th>
<th>Reserves</th>
<th>Boats</th>
</tr>
</thead>
<tbody>
<tr>
<td>sid</td>
<td>sname</td>
<td>rating</td>
</tr>
<tr>
<td>22</td>
<td>Dustin</td>
<td>7</td>
</tr>
<tr>
<td>29</td>
<td>Brutus</td>
<td>1</td>
</tr>
<tr>
<td>31</td>
<td>Lubber</td>
<td>8</td>
</tr>
<tr>
<td>32</td>
<td>Andy</td>
<td>8</td>
</tr>
<tr>
<td>58</td>
<td>Rusty</td>
<td>10</td>
</tr>
<tr>
<td>64</td>
<td>Horatio</td>
<td>7</td>
</tr>
<tr>
<td>71</td>
<td>Zorba</td>
<td>10</td>
</tr>
<tr>
<td>74</td>
<td>Horatio</td>
<td>9</td>
</tr>
<tr>
<td>85</td>
<td>Art</td>
<td>4</td>
</tr>
<tr>
<td>95</td>
<td>Bob</td>
<td>3</td>
</tr>
</tbody>
</table>
Find names of sailors who’ve reserved boat #103

Sailors(sid, sname, rating, age) Reserves(sid, bid, day)
Boats(bid, bname, color)

Solution 1: \[ \pi_{sname}((\sigma_{bid=103} \text{Reserves}) \bowtie \text{Sailors}) \]

Solution 2:
\[ \rho (\text{Temp}_1, \sigma_{bid=103} \text{Reserves}) \]
\[ \rho (\text{Temp}_2, \text{Temp}_1 \bowtie \text{Sailors}) \]
\[ \pi_{sname} (\text{Temp}_2) \]

Solution 3: \[ \pi_{sname}(\sigma_{bid=103}(\text{Reserves} \bowtie \text{Sailors})) \]
Find names of sailors who’ve reserved a red boat

- Information about boat color only available in Boats; so need to do another join:

\[ \pi_{\text{name}}((\sigma_{\text{color} = \text{red}} \text{Boats}) \bowtie \text{Reserves} \bowtie \text{Sailors}) \]

- A more “efficient” solution:

\[ \pi_{\text{name}}(\pi_{\text{sid}}((\sigma_{\text{bid}} \text{Boats}) \bowtie \text{Reserves} \bowtie \text{Sailors}) \]

A query optimizer will find the latter, given the 1st query!
Find sailors who’ve reserved a red or a green boat

\[
\text{Sailors}(\text{sid}, \text{sname}, \text{rating}, \text{age}) \quad \text{Reserves}(\text{sid}, \text{bid}, \text{day}) \\
\text{Boats}(\text{bid}, \text{bname}, \text{color})
\]

- Can identify all red or green boats, then find sailors who’ve reserved one of these boats:

\[
\rho (\text{Tempboats}, (\sigma_{\text{color} = \text{red}} \lor \text{color} = \text{green}) \text{Boats}))
\]

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

- Can also define Tempboats using union! (Q: How?)

- What happens if \( \lor \) is replaced by \( \land \) in this query?

Find sailors who’ve reserved a red and a green boat

\[
\text{Sailors}(\text{sid}, \text{sname}, \text{rating}, \text{age}) \quad \text{Reserves}(\text{sid}, \text{bid}, \text{day}) \\
\text{Boats}(\text{bid}, \text{bname}, \text{color})
\]

- Previous approach won’t work! Must identify sailors who’ve reserved red boats and sailors who’ve reserved green boats, then find their intersection (notice that \text{sid} is a key for Sailors!):

\[
\rho (\text{Tempred}, \pi_{\text{sid}}((\sigma_{\text{color} = \text{red}} \text{Boats}) \bowtie \text{Reserves}))
\]

\[
\rho (\text{Tempgreen}, \pi_{\text{sid}}((\sigma_{\text{color} = \text{green}} \text{Boats}) \bowtie \text{Reserves}))
\]

\[
\pi_{\text{sname}}((\text{Tempred} \cap \text{Tempgreen}) \bowtie \text{Sailors})
\]
Find sailors who’ve reserved a red and a green boat:

- Previous approach won’t work! Must identify sailors who’ve reserved red boats and sailors who’ve reserved green boats, then find their intersection (notice that sid is a key for Sailors!):

  ```plaintext
  rp(s, TempRed \cap TempGrn) \times Sailors
  ```

Find the names of sailors who’ve reserved all boats:

- Uses division; schemas of the input relations feeding the / operator must be carefully chosen:

  ```plaintext
  \rho (\sigma_{\text{color}='red'}(\text{Boats}) \bowtie \text{Reserves}) / \rho (\sigma_{\text{color}='green'}(\text{Boats}) \bowtie \text{Reserves})
  \pi_{\text{sname}}(\text{Tempsids} \bowtie \text{Sailors})
  ```

- To find sailors who’ve reserved all ‘Interlake’ boats:

  ```plaintext
  \sigma_{\text{bname}='\text{Interlake'}}(\text{Boats}) / \pi_{\text{bid}}(\text{Tempsids} \bowtie \text{Sailors})
  ```
Find the names of sailors who’ve reserved all boats:

\[ \Pi_{\text{sname}} (\text{Sailors}) \]

Uses division; schemas of the input relations feeding the / operator must be carefully chosen:

\[ \pi_{\text{sid}, \text{bid}} (\text{Reserves}) / (\pi_{\text{bid}} (\text{Boats})) \]

To find sailors who’ve reserved all Interlake boats:

\[ \pi_{\text{bid}} (\text{Reserves}) \times (\pi_{\text{bname}} = \text{Interlake} (\text{Boats})) \]

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke
Relational Algebra Summary

- The relational model has (several) rigorously defined query languages that are both simple and powerful in nature.
- Relational algebra is more operational; very useful as an internal representation for query evaluation plans.
- Several ways of expressing a given query; a query optimizer should choose the most efficient version. (Take CS122C...! 😊)
- We’ll add a few more operators later on…
- Next up for now: Relational Calculus