Relational Algebra 2

Week 5
Relational Algebra (So far)

- Basic operations:
  - **Selection** ($\sigma$) Selects a subset of rows from relation.
  - **Projection** ($\pi$) Deletes unwanted columns from relation.
  - **Cross-product** ($\times$) Allows us to combine two relations.
  - **Set-difference** (-) Tuples in reln. 1, but not in reln. 2.
  - **Union** ($\cup$) Tuples in reln. 1 and tuples in reln. 2.

Also,
- **Rename** ($\rho$) Changes names of the attributes
- **Intersection** ($\cap$) Tuples in both reln. 1 and in reln. 2.

- Since each operation returns a relation, operations can be composed! (Algebra is “closed”.)
- Use of temporary relations recommended.
Additional Operations

We define additional operations that do not add any power to the relational algebra, but that simplify common queries.

– Natural join
– Conditional Join
– Equi-Join
– Division

Also, we’ve already seen “Set intersection”:

\[ r \cap s = r - (r - s) \]
Quick note on notation

If we have two or more relations which feature the same attribute names, we could confuse them. To prevent this we can use dot notation. For example

\[ \text{good\_customers.loan-number} \]
Natural-Join Operation: Motivation

Very often, we have a query and the answer is not contained in a single relation. For example, I might wish to know where Apu banks. The classic relational algebra way to do such queries is a cross product, followed by a selection which tests for equality on some pair of fields.

\[ \sigma_{\text{borrower}.l\text{-}number = \text{loan}.l\text{-}number}((\text{borrower} \times \text{loan})) \]

While this works…
- it is unintuitive
- it requires a lot of memory
- the notation is cumbersome

So, we have a more intuitive way of achieving the same effect, the natural join, denoted by the \( \Join \) symbol.
Natural-Join Operation: Intuition

Natural join combines a cross product and a selection into one operation. It performs a selection forcing equality on those attributes that appear in both relation schemes. Duplicates are removed as in all relation operations.

So, if the relations have one attribute in common, as in the last slide ("l-number"), for example, we have...

\[
\text{borrower} \bowtie \text{loan} = \sigma_{\text{borrower}.l\text{-number} = \text{loan}.l\text{-number}}(\text{borrower} \times \text{loan}))
\]

There are two special cases:

- If the two relations have no attributes in common, then their natural join is simply their cross product.
- If the two relations have more than one attribute in common, then the natural join selects only the rows where all pairs of matching attributes match. (let’s see an example on the next slide).
Both the *l-name* and the *f-name* match, so select.

Only the *f-names* match, so don’t select.

Only the *l-names* match, so don’t select.

We remove duplicate attributes…

The natural join of $A$ and $B$

Note that this is just a way to visualize the natural join, we don’t really have to do the cross product as in this example.
Natural-Join Operation

- Notation: \( r \Join s \)
- Let \( r \) and \( s \) be relation instances on schemas \( R \) and \( S \) respectively. The result is a relation on schema \( R \cup S \) which is obtained by considering each pair of tuples \( t_r \) from \( r \) and \( t_s \) from \( s \).
- If \( t_r \) and \( t_s \) have the same value on each of the attributes in \( R \cap S \), a tuple \( t \) is added to the result, where
  - \( t \) has the same value as \( t_r \) on \( r \)
  - \( t \) has the same value as \( t_s \) on \( s \)
- Example:
  \[
  R = (A, B, C, D) \\
  S = (E, B, D)
  \]
  Result schema = \((A, B, C, D, E)\)
- \( r \Join s \) is defined as:
  \[
  \pi_{r.A, r.B, r.C, r.D, s.E} (\sigma_{r.B = s.B \land r.D = s.D} (r \times s))
  \]
Natural Join Operation – Example

- Relation instances $r, s$:

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$A$</td>
<td>$B$</td>
<td>$C$</td>
<td>$D$</td>
</tr>
<tr>
<td>$\alpha$</td>
<td>1</td>
<td>$\alpha$</td>
<td>a</td>
</tr>
<tr>
<td>$\beta$</td>
<td>2</td>
<td>$\gamma$</td>
<td>a</td>
</tr>
<tr>
<td>$\gamma$</td>
<td>4</td>
<td>$\beta$</td>
<td>b</td>
</tr>
<tr>
<td>$\alpha$</td>
<td>1</td>
<td>$\gamma$</td>
<td>a</td>
</tr>
<tr>
<td>$\delta$</td>
<td>2</td>
<td>$\beta$</td>
<td>b</td>
</tr>
</tbody>
</table>

$r 
\times s$

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$B$</td>
<td>$D$</td>
<td>$E$</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>a</td>
<td>$\alpha$</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>a</td>
<td>$\beta$</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>a</td>
<td>$\gamma$</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>b</td>
<td>$\delta$</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>b</td>
<td>$\in$</td>
<td></td>
</tr>
</tbody>
</table>

How did we get here?

Let's do a trace over the next few slides…
First we note which attributes the two relations have in common…
There are two rows in \( s \) that match our first row in \( r \), (in the relevant attributes) so both are joined to our first row…
…there are no rows in $s$ that match our second row in $r$, so do nothing…
...there are no rows in $s$ that match our third row in $r$, so do nothing...
There are two rows in $S$ that match our fourth row in $R$, so both are joined to our fourth row…
There is one row that matches our fifth row in $r$, so it is joined to our fifth row and we are done!
Natural Join on Sailors Example

<table>
<thead>
<tr>
<th>sid</th>
<th>surname</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>58</td>
<td>rusty</td>
<td>10</td>
<td>35.0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>sid</th>
<th>bid</th>
<th>day</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>101</td>
<td>10/10/96</td>
</tr>
<tr>
<td>58</td>
<td>103</td>
<td>11/12/96</td>
</tr>
</tbody>
</table>

\[ S_1 \bowtie R_1 = \]

<table>
<thead>
<tr>
<th>sid</th>
<th>surname</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>
Earlier We Saw…

Query: Find the name of the sailor who reserved boat 101.

\[
\text{Temp} = \rho (sid \rightarrow sid1, S1) \times \rho (sid \rightarrow sid2, R1)
\]

\[
\text{Result} = \pi Sname (\sigma sid1 = sid2 \land bid = 101 (\text{Temp}))
\]

* Note my use of “temporary” relation Temp.
Query revisited using natural join

Query: Find the name of the sailor who reserved boat 101.

Result = $\pi_{\text{Sname}} (\sigma_{\text{bid}=101} (S1 \bowtie R1))$

Or

Result = $\pi_{\text{Sname}} (S1 \bowtie \sigma_{\text{bid}=101} (R1))$

What’s the difference between these two approaches?
Conditional-Join Operation:

The conditional join is actually the most general type of join. I introduced the natural join first only because it is more intuitive and... natural!

Just like natural join, conditional join combines a cross product and a selection into one operation. However instead of only selecting rows that have equality on those attributes that appear in both relation schemes, we allow selection based on any predicate.

\[ r \Join_c s = \sigma_c(r \times s) \]

Where \( c \) is any predicate the attributes of \( r \) and/or \( s \)

Duplicate rows are removed as always, but duplicate columns are not removed!
Conditional-Join Example:

We want to find all women that are younger than their husbands…

We want to find all women that are younger than their husbands…

We want to find all women that are younger than their husbands...

Note we have removed ambiguity of attribute names by using “dot” notation.
Also note the redundant information in the marr-lic attributes.
Equi-Join

• Equi-Join: Special case of conditional join where the conditions consist only of equalities.

• Natural Join: Special case of equi-join in which equalities are specified on ALL fields having the same names in both relations.
Equi-Join

\[ r \bowtie s \quad r.\text{Marr-Lic} = s.\text{Marr-Lic} \]

<table>
<thead>
<tr>
<th>l-name</th>
<th>f-name</th>
<th>marr-Lic</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simpson</td>
<td>Marge</td>
<td>777</td>
<td>35</td>
</tr>
<tr>
<td>Lovejoy</td>
<td>Helen</td>
<td>234</td>
<td>38</td>
</tr>
<tr>
<td>Flanders</td>
<td>Maude</td>
<td>555</td>
<td>24</td>
</tr>
<tr>
<td>Krabappel</td>
<td>Edna</td>
<td>978</td>
<td>40</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>l-name</th>
<th>f-name</th>
<th>marr-Lic</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simpson</td>
<td>Homer</td>
<td>777</td>
<td>36</td>
</tr>
<tr>
<td>Lovejoy</td>
<td>Timothy</td>
<td>234</td>
<td>36</td>
</tr>
<tr>
<td>Simpson</td>
<td>Bart</td>
<td>null</td>
<td>9</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>r.l-name</th>
<th>r.f-name</th>
<th>Marr-Lic</th>
<th>r.age</th>
<th>s.l-name</th>
<th>s.f-name</th>
<th>s.age</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simpson</td>
<td>Marge</td>
<td>777</td>
<td>35</td>
<td>Simpson</td>
<td>Homer</td>
<td>36</td>
</tr>
<tr>
<td>Lovejoy</td>
<td>Helen</td>
<td>234</td>
<td>38</td>
<td>Lovejoy</td>
<td>Timothy</td>
<td>36</td>
</tr>
</tbody>
</table>
Review on Joins

• All joins combine a cross product and a selection into one operation.

• Conditional Join
  – the selection condition can be of any predicate (e.g. rating1 > rating2)

• Equi-Join:
  – Special case of conditional join where the conditions consist only of equalities.

• Natural Join
  – Special case of equi-join in which equalities are specified on ALL fields having the same names in both relations.
Banking Examples

branch (branch-id, branch-city, assets)

customer (customer-id, customer-name, customer-city)

account (account-number, branch-id, balance)

loan (loan-number, branch-id, amount)

depositor (customer-id, account-number)

borrower (customer-id, loan-number)
Example Queries 1

• Find all loans over $1200

“select from the relation loan, only the rows which have a amount greater than 1200”

<table>
<thead>
<tr>
<th>loan-number</th>
<th>branch-id</th>
<th>amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>1234</td>
<td>001</td>
<td>1,923.03</td>
</tr>
<tr>
<td>3421</td>
<td>002</td>
<td>123.00</td>
</tr>
<tr>
<td>2342</td>
<td>004</td>
<td>56.25</td>
</tr>
<tr>
<td>4531</td>
<td>005</td>
<td>120.03</td>
</tr>
</tbody>
</table>

$\sigma_{amount > 1200} (\text{loan})$

<table>
<thead>
<tr>
<th>loan-number</th>
<th>branch-id</th>
<th>amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>1234</td>
<td>001</td>
<td>1,923.03</td>
</tr>
</tbody>
</table>
Example Queries 2

• Find the loan number for each loan of an amount greater than $1200

“select from the relation loan, only the rows which have a amount greater than 1200, then project out just the loan_number”

<table>
<thead>
<tr>
<th>loan-number</th>
<th>branch-id</th>
<th>amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>1234</td>
<td>001</td>
<td>1,923.03</td>
</tr>
<tr>
<td>3421</td>
<td>002</td>
<td>123.00</td>
</tr>
<tr>
<td>2342</td>
<td>004</td>
<td>56.25</td>
</tr>
<tr>
<td>4531</td>
<td>005</td>
<td>120.03</td>
</tr>
</tbody>
</table>

\[ \sigma_{\text{amount} > 1200} \text{(loan)} \]

\[ \pi_{\text{loan-number}} (\sigma_{\text{amount} > 1200} \text{(loan)}) \]
Example Queries 3

• Find all loans greater than $1200 or less than $75

“select from the relation loan, only the rows which have a amount greater than 1200 or an amount less than 75

<table>
<thead>
<tr>
<th>loan-number</th>
<th>branch-id</th>
<th>amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>1234</td>
<td>001</td>
<td>1,923.03</td>
</tr>
<tr>
<td>3421</td>
<td>002</td>
<td>123.00</td>
</tr>
<tr>
<td>2342</td>
<td>004</td>
<td>56.25</td>
</tr>
<tr>
<td>4531</td>
<td>005</td>
<td>120.03</td>
</tr>
</tbody>
</table>

\[ \sigma_{\text{amount} > 1200 \lor \text{amount} < 75}(\text{loan}) \]
Example Queries 4

- Find the IDs of all customers who have a loan, an account, or both, from the bank

<table>
<thead>
<tr>
<th>customer-id</th>
<th>loan-number</th>
</tr>
</thead>
<tbody>
<tr>
<td>201</td>
<td>1234</td>
</tr>
<tr>
<td>304</td>
<td>3421</td>
</tr>
<tr>
<td>425</td>
<td>2342</td>
</tr>
<tr>
<td>109</td>
<td>4531</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>customer-id</th>
<th>account-number</th>
</tr>
</thead>
<tbody>
<tr>
<td>333</td>
<td>3467</td>
</tr>
<tr>
<td>304</td>
<td>2312</td>
</tr>
<tr>
<td>201</td>
<td>9999</td>
</tr>
<tr>
<td>492</td>
<td>3423</td>
</tr>
</tbody>
</table>

\[ \pi_{\text{customer-id}}(\text{borrower}) \cup \pi_{\text{customer-id}}(\text{depositor}) \]
Find the IDs of all customers who have a loan at branch 001.

We retrieve *borrower* and *loan*…

…we calculate their cross product…

<table>
<thead>
<tr>
<th>customer-id</th>
<th>loan-number</th>
<th>branch-id</th>
<th>amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>1234</td>
<td>001</td>
<td>1,923.03</td>
<td></td>
</tr>
<tr>
<td>3421</td>
<td>002</td>
<td>123.00</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>customer-id</th>
<th>borrower.loan-number</th>
<th>loan.loan-number</th>
<th>branch-id</th>
<th>amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>201</td>
<td>1234</td>
<td>1234</td>
<td>001</td>
<td>1,923.03</td>
</tr>
<tr>
<td>201</td>
<td>1234</td>
<td>3421</td>
<td>002</td>
<td>123.00</td>
</tr>
<tr>
<td>304</td>
<td>3421</td>
<td>1234</td>
<td>001</td>
<td>1,923.03</td>
</tr>
<tr>
<td>304</td>
<td>3421</td>
<td>3421</td>
<td>002</td>
<td>123.00</td>
</tr>
</tbody>
</table>
…we calculate their cross product…

<table>
<thead>
<tr>
<th>customer-id</th>
<th>borrower.loan-number</th>
<th>loan.loan-number</th>
<th>branch-id</th>
<th>amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>201</td>
<td>1234</td>
<td>1234</td>
<td>001</td>
<td>1,923.03</td>
</tr>
<tr>
<td>201</td>
<td>1234</td>
<td>3421</td>
<td>002</td>
<td>123.00</td>
</tr>
<tr>
<td>304</td>
<td>3421</td>
<td>1234</td>
<td>001</td>
<td>1,923.03</td>
</tr>
<tr>
<td>304</td>
<td>3421</td>
<td>3421</td>
<td>002</td>
<td>123.00</td>
</tr>
</tbody>
</table>

…we select the rows where borrower.loan-number is equal to loan.loan-number…

<table>
<thead>
<tr>
<th>customer-id</th>
<th>borrower.loan-number</th>
<th>loan.loan-number</th>
<th>branch-id</th>
<th>amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>201</td>
<td>1234</td>
<td>1234</td>
<td>001</td>
<td>1,923.03</td>
</tr>
<tr>
<td>304</td>
<td>3421</td>
<td>3421</td>
<td>002</td>
<td>123.00</td>
</tr>
</tbody>
</table>

…we select the rows where branch-id is equal to “001”

<table>
<thead>
<tr>
<th>customer-id</th>
<th>borrower.loan-number</th>
<th>loan.loan-number</th>
<th>branch-id</th>
<th>amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>201</td>
<td>1234</td>
<td>1234</td>
<td>001</td>
<td>1,923.03</td>
</tr>
</tbody>
</table>

…we project out the customer-id.

$$\pi_{\text{customer-id}}(\sigma_{\text{branch-id}=‘001’}(\sigma_{\text{borrower.loan-number} = \text{loan.loan-number}}(\text{borrower} \times \text{loan})))$$
Now Using Natural Join

Find the IDs of all customers who have a loan at branch 001.

We retrieve borrower and loan…

1234 in borrower is matched with 1234 in loan…

3421 in borrower is matched with 3421 in loan…

The rest is the same.

\[
\pi_{\text{customer-id}} \left( \sigma_{\text{branch-id}='001'} \left( \sigma_{\text{borrower.loan-number} = \text{loan.loan-number}} \left( \text{borrower} \times \text{loan} \right) \right) \right)
\]

\[
= \pi_{\text{customer-id}} \left( \sigma_{\text{branch-id}='001'} \left( \text{borrower} \bowtie \text{loan} \right) \right)
\]
Example Queries 7

- Find the names of all customers who have a loan, an account, or both, from the bank

\[ \pi_{\text{customer-id}}(\text{borrower}) \cup \pi_{\text{customer-id}}(\text{depositor}) \]

<table>
<thead>
<tr>
<th>customer-id</th>
<th>customer-name</th>
<th>customer-city</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>Carol</td>
<td>Fairfax</td>
</tr>
<tr>
<td>109</td>
<td>David</td>
<td>Fairfax</td>
</tr>
<tr>
<td>201</td>
<td>John</td>
<td>Vienna</td>
</tr>
<tr>
<td>304</td>
<td>Mary</td>
<td>McLean</td>
</tr>
<tr>
<td>333</td>
<td>Ben</td>
<td>Chantilly</td>
</tr>
<tr>
<td>425</td>
<td>David</td>
<td>Manassas</td>
</tr>
<tr>
<td>492</td>
<td>Jason</td>
<td>Fairfax</td>
</tr>
<tr>
<td>501</td>
<td>Adam</td>
<td>Burke</td>
</tr>
</tbody>
</table>

Note this example is split over two slides!
Example Queries

- Find the names of all customers who have a loan, an account, or both, from the bank

<table>
<thead>
<tr>
<th>customer-id</th>
<th>customer-name</th>
<th>customer-city</th>
</tr>
</thead>
<tbody>
<tr>
<td>109</td>
<td>David</td>
<td>Fairfax</td>
</tr>
<tr>
<td>201</td>
<td>John</td>
<td>Vienna</td>
</tr>
<tr>
<td>304</td>
<td>Mary</td>
<td>McLean</td>
</tr>
<tr>
<td>333</td>
<td>Ben</td>
<td>Chantilly</td>
</tr>
<tr>
<td>425</td>
<td>David</td>
<td>Manassas</td>
</tr>
<tr>
<td>492</td>
<td>Jason</td>
<td>Fairfax</td>
</tr>
</tbody>
</table>

$$\pi_{\text{customer-name}} (( \pi_{\text{customer-id}} (\text{borrower}) \cup \pi_{\text{customer-id}} (\text{depositor}) ) \bowtie \text{customer})$$
Example Queries 8

Find the largest account balance

We do a rename to get a “copy” of the balance column from account. We call this copy $d$…

… next we will do a cross product…
… do a cross product…

…select out all rows where `account.balance` is less than `d.balance`…

<table>
<thead>
<tr>
<th>account-number</th>
<th>branch-id</th>
<th>account.balance</th>
<th>d.balance</th>
</tr>
</thead>
<tbody>
<tr>
<td>7777</td>
<td>001</td>
<td>100.30</td>
<td>100.30</td>
</tr>
<tr>
<td>7777</td>
<td>001</td>
<td>100.30</td>
<td>12.34</td>
</tr>
<tr>
<td>7777</td>
<td>001</td>
<td>100.30</td>
<td>45.34</td>
</tr>
<tr>
<td>8888</td>
<td>003</td>
<td>12.34</td>
<td>100.30</td>
</tr>
<tr>
<td>8888</td>
<td>003</td>
<td>12.34</td>
<td>12.34</td>
</tr>
<tr>
<td>8888</td>
<td>003</td>
<td>12.34</td>
<td>45.34</td>
</tr>
<tr>
<td>6666</td>
<td>004</td>
<td>45.34</td>
<td>100.30</td>
</tr>
<tr>
<td>6666</td>
<td>004</td>
<td>45.34</td>
<td>12.34</td>
</tr>
<tr>
<td>6666</td>
<td>004</td>
<td>45.34</td>
<td>45.34</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>account-number</th>
<th>branch-id</th>
<th>account.balance</th>
<th>d.balance</th>
</tr>
</thead>
<tbody>
<tr>
<td>8888</td>
<td>003</td>
<td>12.34</td>
<td>100.30</td>
</tr>
<tr>
<td>8888</td>
<td>003</td>
<td>12.34</td>
<td>45.34</td>
</tr>
<tr>
<td>6666</td>
<td>004</td>
<td>45.34</td>
<td>100.30</td>
</tr>
</tbody>
</table>
.. next we project out 
\textit{account.balance}…

…then we do a set difference between it and the original 
\textit{account.balance} from the account relation…

… the set difference leaves us with one number, the largest value!

\[
\pi_{\text{balance}}(\text{account}) - \pi_{\text{account.balance}} (\sigma_{\text{account.balance} < \text{d.balance}} (\text{account} \times \rho(d, \pi_{\text{balance}}(\text{account}))))
\]
Now Using Conditional Join

Find the largest account balance

\[
\pi_{\text{balance}}(\text{account}) - \pi_{\text{account.balance}}(\sigma_{\text{account.balance} < d.\text{balance}} (\text{account} \times \rho(d, \pi_{\text{balance}}(\text{account}))))
\]

\[
\rho(d, \pi_{\text{account.balance}}(\text{account}))
\]

\[
\pi_{\text{balance}}(\text{account}) - \pi_{\text{account.balance}}(\text{account} \bowtie_{\text{account.balance} < d.\text{balance}} d)
\]
More Examples on Sailors Relations

Sailors(sid, sname, rating, age)
Boats(bid, bname, color)
Reserves(sid, bid, day)
Find names of sailors who’ve reserved boat #103

- Solution 1: Find those who reserved boat 103, join with Sailors to find the names, and project out the names

- Solution 2: Join Reserves and Sailors to get all information, and find those who reserved boat 103. Project out the names.

Which one is more efficient?
Find names of sailors who’ve reserved boat #103

- Solution 1: Find those who reserved boat 103, join with Sailors to find the names, and project out the names
  \[ \pi_{\text{name}} (\sigma_{\text{bid} = 103} (\text{Reserves}) \bowtie \text{Sailors}) \]

- Solution 2: Join Reserves and Sailors to get all information, and find those who reserved boat 103. Project out the names.
  \[ \pi_{\text{name}} (\sigma_{\text{bid} = 103} (\text{Reserves} \bowtie \text{Sailors})) \]

Which one is more efficient?
Find names of sailors who’ve reserved a red boat

• Information about boat color only available in Boats; so need an extra join:

• A more efficient solution: Find the bids of red boats first before doing the join.

☞ A query optimizer can find this given the first solution!
Find names of sailors who’ve reserved a red boat

• Information about boat color only available in Boats; so need an extra join:

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

• A more efficient solution: Find the bids of red boats first before doing the join.

$$\pi_{\text{name}}(\pi_{\text{sid}}((\pi_{\text{bid}} \sigma_{\text{color} = \text{'red'}} \text{Boats}) \bowtie \text{Res}) \bowtie \text{Sailors})$$

⚠️ A query optimizer can find this given the first solution!
Find sailors who’ve reserved a red or a green boat

• Can identify all red or green boats, then find sailors who’ve reserved one of these boats:
Find sailors who’ve reserved a red or a green boat

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

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

\[
\text{Result} = \pi_{\text{sname}}(\text{Tempboats} \bowtie \text{Reserves} \bowtie \text{Sailors})
\]
Find sailors who’ve reserved a **red** or a **green** boat

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

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

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

- Can also define Tempboats using union:

\[
\text{Tempboats} = \sigma_{\text{color} = \text{'red'}}(\text{Boats}) \cup \sigma_{\text{color} = \text{'green'}}(\text{Boats})
\]

- What happens if “or” is replaced by “and”? 
Find sailors who’ve reserved a red and a green boat

- Previous first approach won’t work! (Why not?) Must use intersection.

\[
\begin{align*}
\text{Tempred} &= \pi_{\text{sid}}((\sigma_{\text{color} = \text{\textquoteleft}red\textquoteright})(\text{Boats}))\bowtie\text{Reserves} \\
\text{Tempgreen} &= \pi_{\text{sid}}((\sigma_{\text{color} = \text{\textquoteleft}green\textquoteright})(\text{Boats}))\bowtie\text{Reserves} \\
\text{Result} &= \pi_{\text{fname}}((\text{Tempred} \cap \text{Tempgreen})\bowtie\text{Sailors})
\end{align*}
\]
Consider yet another query

- Find the sailor(s) who reserved all the red boats.

R1 | B
---|---
| sid | bid | day | bid | color |
| 22  | 101 | 10/10/96 | 101 | Red   |
| 22  | 103 | 10/11/96 | 102 | Green |
| 56  | 102 | 11/12/96 | 103 | Red   |
Start an attempt

• Who reserved what boat:

\[ S_1 = \pi_{\text{sid}, \text{bid}} (R_1) = \]

\[
\begin{array}{|c|c|}
\hline
\text{sid} & \text{bid} \\
\hline
22 & 101 \\
22 & 103 \\
56 & 102 \\
\hline
\end{array}
\]

• All the red boats:

\[ S_2 = \pi_{\text{bid}} (\sigma_{\text{color} = \text{red}} (B)) = \]

\[
\begin{array}{|c|}
\hline
\text{bid} \\
\hline
101 \\
103 \\
\hline
\end{array}
\]

Now what?
Find the sailor(s) who reserved all the red boats.

- We will solve the problem the “hard” way, and then will introduce an operator specifically for this kind of problem.

**Idea:** Compute the sids of sailors who didn’t reserve all red boats.

1. Find all possible reservations that could be made on red boats.
2. Find *actual* reservations on red boats
3. Find the possible reservations on red boats that were not actually made (#1 – #2) <- that is a minus sign.
4. Project out the sids from 3 – these are the sailors who didn’t have reservation on some red boat(s).
Find the sailor(s) who reserved all the red boats.

- **Idea:** Compute the sids of sailors who *didn’t* reserve all red boats (then find the difference between this set and set of all sailors).

1. Find all possible reservations that could be made on red boats.
   \[ \text{AllPossibleRes} = \pi_{\text{sid}} (R1) \times \pi_{\text{bid}} \sigma_{\text{color} = \text{"red"}} (B) \]

2. Find *actual* reservations on red boats
   \[ \text{AllRedRes} = \pi_{\text{sid, bid}} (R1) \Join \pi_{\text{bid}} \sigma_{\text{color} = \text{"red"}} (B) \]

3. Find the possible reservations on red boats that were not actually made, and project out the sids.
   \[ \pi_{\text{sid}} (\text{AllPossibleRes} - \text{AllRedRes}) \]

4. Find sids that are not in the result from above (sailors such that there is no red boat that’s not reserved by him/her)
   \[ \pi_{\text{sid}} (R1) - \pi_{\text{sid}} (\text{AllPossibleRes} - \text{AllRedRes}) \]
Division Operation

\[ r / s \]

- Suited to queries that include the phrase “for all”, e.g. Find sailors who have reserved \textit{all} red boats.
- Produce the tuples in one relation, \( r \), that match \textit{all} tuples in another relation, \( s \).
- Let \( S1 \) have 2 fields, \( x \) and \( y \); \( S2 \) have only field \( y \):
  - \( S1/S2 = \{ (x) | \forall \langle y \rangle \ in \ S2 \ (\exists \langle x, y \rangle \ in \ S1) \} \)
  - i.e., \( S1/S2 \) contains all \( x \) tuples (sailors) such that for \textit{every} \( y \) tuple (redboat) in \( S2 \), there is an \( xy \) tuple in \( S1 \) (i.e., \( x \) reserved \( y \)).

- In general, \( x \) and \( y \) can be any lists of fields; \( y \) is the list of fields in \( S2 \), and \( x \cup y \) is the list of fields of \( S1 \).
- Let \( r \) and \( s \) be relations on schemas \( R \) and \( S \) respectively where
  - \( R = (A_1, \ldots, A_m, B_1, \ldots, B_n) \),
  - \( S = (B_1, \ldots, B_n) \),
- The result of \( r / s \) is a relation on schema \( R - S = (A_1, \ldots, A_m) \)
Division (cont’d)

Relation \( r \)

Relation \( s \)

B

\[
\begin{array}{c}
\hline
a \\
\hline
b \\
\hline
c \\
\hline
\end{array}
\]

\[
\begin{array}{c}
\hline
\text{In } r/s \\
\hline
\text{Not in } r/s \\
\hline
\end{array}
\]

1 \quad a \\
1 \quad b \\
2 \quad a \\
2 \quad b \\
2 \quad c \\
3 \quad a \\
3 \quad b \\
3 \quad c \\
4 \quad b \\
4 \quad c
Division Operation – Example

Relations \( r, s \):

<table>
<thead>
<tr>
<th>( A )</th>
<th>( B )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \alpha )</td>
<td>1</td>
</tr>
<tr>
<td>( \alpha )</td>
<td>2</td>
</tr>
<tr>
<td>( \alpha )</td>
<td>3</td>
</tr>
<tr>
<td>( \beta )</td>
<td>1</td>
</tr>
<tr>
<td>( \gamma )</td>
<td>1</td>
</tr>
<tr>
<td>( \delta )</td>
<td>1</td>
</tr>
<tr>
<td>( \delta )</td>
<td>3</td>
</tr>
<tr>
<td>( \delta )</td>
<td>4</td>
</tr>
<tr>
<td>( \in )</td>
<td>6</td>
</tr>
<tr>
<td>( \in )</td>
<td>1</td>
</tr>
<tr>
<td>( \beta )</td>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>( B )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
</tbody>
</table>

\( r \) / \( s \):

\( \alpha \) occurs in the presence of both 1 and 2, so it is returned.
\( \beta \) occurs in the presence of both 1 and 2, so it is returned.
\( \gamma \) does not occur in the presence of both 1 and 2, so is ignored.

...
Another Division Example

Relations \( r, s \):

\[
\begin{array}{cccccc}
A & B & C & D & E \\
\alpha & a & \alpha & a & 1 \\
\alpha & a & \gamma & a & 1 \\
\alpha & a & \gamma & b & 1 \\
\beta & a & \gamma & a & 1 \\
\beta & a & \gamma & b & 3 \\
\gamma & a & \gamma & a & 1 \\
\gamma & a & \gamma & b & 1 \\
\gamma & a & \beta & b & 1 \\
\end{array}
\]

\[
\begin{array}{cc}
D & E \\
a & 1 \\
b & 1 \\
\end{array}
\]

\[
\begin{array}{ccc}
A & B & C \\
\alpha & a & \gamma \\
\gamma & a & \gamma \\
\end{array}
\]

\(<\alpha, a, \gamma>\) occurs in the presence of both \(<a,1>\) and \(<b,1>\), so it is returned.

\(<\gamma, a, \gamma>\) occurs in the presence of both \(<a,1>\) and \(<b,1>\), so it is returned.

\(<\beta, a, \gamma>\) does not occur in the presence of both \(<a,1>\) and \(<b,1>\), so it is ignored.
More Division Examples: 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>p2</td>
</tr>
<tr>
<td>s1</td>
<td>p2</td>
<td>p2</td>
<td>p4</td>
</tr>
<tr>
<td>s1</td>
<td>p3</td>
<td>p2</td>
<td>p4</td>
</tr>
<tr>
<td>s1</td>
<td>p4</td>
<td>p2</td>
<td>p4</td>
</tr>
<tr>
<td>s2</td>
<td>p1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>s2</td>
<td>p2</td>
<td></td>
<td></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

B1

B2

B3

A/B2

A/B3

A/B1

A/B2

A/B3
Find the sailor(s) who reserved ALL red boats

- who reserved what boat:

\[ S_1 = \pi_{\text{sid}, \text{bid}}(R_1) = \]

\[
\begin{array}{|c|c|}
\hline
\text{sid} & \text{bid} \\
\hline
22 & 101 \\
22 & 103 \\
58 & 102 \\
\hline
\end{array}
\]

- All the red boats:

\[ S_2 = \pi_{\text{bid}}(\sigma_{\text{color}=\text{red}}(B)) = \]

\[
\begin{array}{|c|}
\hline
\text{bid} \\
\hline
101 \\
103 \\
\hline
\end{array}
\]

=> S1/S2
Find the names of sailors who’ve reserved all boats

- Uses division; schemas of the input relations to “divide” must be carefully chosen:

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

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

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

\[
\ldots / \pi_{\text{bid}}(\sigma_{\text{bname} = 'Interlake'}(\text{Boats}))
\]
• SALES(supId, prodId);
• PRODUCTS(prodId);
• SALES/PRODUCTS: the ids of the suppliers supplying ALL products.
Expressing A/B Using Basic Operators

• Division is not essential op; just a useful shorthand.
  - (Also true of joins, but joins are so common that systems implement joins specially. Division is NOT implemented in SQL).

• **Idea:** For *SALES/PRODUCTS*, compute the IDs of suppliers that don’t supply all products.

\[
A = \pi_{\text{sid}}\left(\pi_{\text{sid}}(\text{Sales} \times \text{Products}) - \text{Sales}\right)
\]

The answer is \( \pi_{\text{sid}}(\text{Sales}) - A \)
Additional Operator: Outer Join

• An extension of the join operation that avoids loss of information.
• Computes the join and then adds tuples from one relation that does not match tuples in the other relation to the result of the join.
• Uses null values:
  – null signifies that the value is unknown or does not exist
  – All comparisons involving null are (roughly speaking) false by definition.
  • Will study precise meaning of comparisons with nulls later
Outer Join – Example

- Relation *loan*

- Relation *borrower*

<table>
<thead>
<tr>
<th>loan-number</th>
<th>branch-name</th>
<th>amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>L-170</td>
<td>Springfield</td>
<td>3000</td>
</tr>
<tr>
<td>L-230</td>
<td>Shelbyville</td>
<td>4000</td>
</tr>
<tr>
<td>L-260</td>
<td>Dublin</td>
<td>1700</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>customer-name</th>
<th>loan-number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simpson</td>
<td>L-170</td>
</tr>
<tr>
<td>Wiggum</td>
<td>L-230</td>
</tr>
<tr>
<td>Flanders</td>
<td>L-155</td>
</tr>
</tbody>
</table>
### Outer Join – Example

**Inner Join**

\[ \text{loan} \Join \text{Borrower} \]

<table>
<thead>
<tr>
<th>loan-number</th>
<th>branch-name</th>
<th>amount</th>
<th>customer-name</th>
</tr>
</thead>
<tbody>
<tr>
<td>L-170</td>
<td>Springfield</td>
<td>3000</td>
<td>Simpson</td>
</tr>
<tr>
<td>L-230</td>
<td>Shelbyville</td>
<td>4000</td>
<td>Wiggum</td>
</tr>
</tbody>
</table>

**Left Outer Join**

\[ \text{loan} \LeftJoin \text{borrower} \]

<table>
<thead>
<tr>
<th>loan-number</th>
<th>branch-name</th>
<th>amount</th>
<th>customer-name</th>
</tr>
</thead>
<tbody>
<tr>
<td>L-170</td>
<td>Springfield</td>
<td>3000</td>
<td>Simpson</td>
</tr>
<tr>
<td>L-230</td>
<td>Shelbyville</td>
<td>4000</td>
<td>Wiggum</td>
</tr>
<tr>
<td>L-260</td>
<td>Dublin</td>
<td>1700</td>
<td></td>
</tr>
</tbody>
</table>
Outer Join – Example

### Right Outer Join

<table>
<thead>
<tr>
<th>loan-number</th>
<th>branch-name</th>
<th>amount</th>
<th>customer-name</th>
</tr>
</thead>
<tbody>
<tr>
<td>L-170</td>
<td>Springfield</td>
<td>3000</td>
<td>Simpson</td>
</tr>
<tr>
<td>L-230</td>
<td>Shelbyville</td>
<td>4000</td>
<td>Wiggum</td>
</tr>
<tr>
<td>L-155</td>
<td>null</td>
<td>null</td>
<td>Flanders</td>
</tr>
</tbody>
</table>

### Full Outer Join

<table>
<thead>
<tr>
<th>loan-number</th>
<th>branch-name</th>
<th>amount</th>
<th>customer-name</th>
</tr>
</thead>
<tbody>
<tr>
<td>L-170</td>
<td>Springfield</td>
<td>3000</td>
<td>Simpson</td>
</tr>
<tr>
<td>L-230</td>
<td>Shelbyville</td>
<td>4000</td>
<td>Wiggum</td>
</tr>
<tr>
<td>L-155</td>
<td>null</td>
<td>null</td>
<td>Flanders</td>
</tr>
<tr>
<td>L-260</td>
<td>Dublin</td>
<td>1700</td>
<td></td>
</tr>
</tbody>
</table>

---

**loan** $\bowtie$ **borrower**

---

**loan** $\bowtie\bowtie$ **borrower**
Null Values

- It is possible for tuples to have a null value, denoted by null, for some of their attributes
- null signifies an unknown value or that a value does not exist.
- The result of any arithmetic expression involving null is null.
- Aggregate functions simply ignore null values
  - Is an arbitrary decision. Could have returned null as result instead.
  - We follow the semantics of SQL in its handling of null values
- For duplicate elimination and grouping, null is treated like any other value, and two nulls are assumed to be the same
  - Alternative: assume each null is different from each other
  - Both are arbitrary decisions, so we simply follow SQL
Null Values

- Comparisons with null values return the special truth value *unknown*
- Three-valued logic using the truth value *unknown*:
  - **OR:** \((\text{unknown or true}) = \text{true},\) 
    \((\text{unknown or false}) = \text{unknown}\) 
    \((\text{unknown or unknown}) = \text{unknown}\)
  - **AND:** \((\text{true and unknown}) = \text{unknown},\) 
    \((\text{false and unknown}) = \text{false},\) 
    \((\text{unknown and unknown}) = \text{unknown}\)
  - **NOT:** \((\text{not unknown}) = \text{unknown}\)
  - In SQL “*P is unknown*” evaluates to true if predicate *P* evaluates to *unknown*
- Result of select predicate is treated as *false* if it evaluates to *unknown*
Summary

- The relational model has rigorously defined query languages that are simple and powerful.
- Relational algebra is more operational; useful as internal representation for query evaluation plans.
- Several ways of expressing a given query; a query optimizer should choose the most efficient version.
- Operations covered: 5 basic operations (selection, projection, union, set difference, cross product), rename, joins (natural join, equi-join, conditional join, outer joins), division