Relational Algebra 1

Week 4
Relational Query Languages

• *Query languages*: Allow manipulation and retrieval of data from a database.

• Relational model supports simple, powerful QLs:
  – Strong formal foundation based on logic.
  – Allows for much optimization.

• Query Languages != programming languages!
  – QLs not expected to be “Turing complete”.
  – QLs not intended to be used for complex calculations.
  – QLs support easy, efficient access to large data sets.
Formal Relational Query Languages

Two mathematical Query Languages form the basis for “real” languages (e.g. SQL), and for implementation:

1. **Relational Algebra**: More operational, very useful for representing execution plans.

2. **Relational Calculus**: Lets users describe what they want, rather than how to compute it. (Non-operational, declarative.)

👉 Understanding Algebra is key to understanding SQL, and query processing!
The Role of Relational Algebra in a DBMS
Algebra Preliminaries

- A query is applied to *relation instances*, and the result of a query is also a relation instance.
  - *Schemas* of input relations for a query are **fixed** (but query will run regardless of instance!)
  - The **schema for the result** of a given query is also **fixed**! Determined by definition of query language constructs.
Relational Algebra

- Procedural language
- Five basic operators
  - selection
  - projection
  - union
  - set difference
  - Cross product

SQL is closely based on relational algebra.

- The are some other operators which are composed of the above operators. These show up so often that we give them special names.
- The operators take one or two relations as inputs and give a new relation as a result.
Select Operation – Example

• Relation \( r \)

\[
\begin{array}{cccc}
A & B & C & D \\
\alpha & \alpha & 1 & 7 \\
\alpha & \beta & 5 & 7 \\
\beta & \beta & 12 & 3 \\
\beta & \beta & 23 & 10 \\
\end{array}
\]

Intuition: The select operation allows us to retrieve some rows of a relation (by “some” I mean anywhere from none of them to all of them)

Here I have retrieved all the rows of the relation \( r \) where the value in field \( A \) equals the value in field \( B \), and the value in field \( D \) is greater than 5.
Select Operation

• Notation: $\sigma_p(r)$

• $p$ is called the selection predicate

• Defined as:

$$\sigma_p(r) = \{ t | t \in r \text{ and } p(t) \}$$

Where $p$ is a formula in propositional calculus consisting of terms connected by: $\wedge$ (and), $\vee$ (or), $\neg$ (not)

Each term is one of:

<attribute> op <attribute> or <constant>

where op is one of: $=, \neq, >, \geq, <, \leq$

• Example of selection:

$$\sigma_{name='Lee'}(professor)$$
Project Operation – Example I

- Relation $r$:

<table>
<thead>
<tr>
<th></th>
<th>$A$</th>
<th>$B$</th>
<th>$C$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\alpha$</td>
<td>10</td>
<td></td>
<td>7</td>
</tr>
<tr>
<td>$\alpha$</td>
<td>20</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>$\beta$</td>
<td>30</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>$\beta$</td>
<td>40</td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>

Intuition: The **project** operation allows us to retrieve some columns of a relation (by “some” I mean anywhere from none of them to all of them)

- $\pi_{A,C}(r)$

<table>
<thead>
<tr>
<th></th>
<th>$A$</th>
<th>$C$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\alpha$</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>$\alpha$</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>$\beta$</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>$\beta$</td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>

Here I have retrieved columns $A$ and $C$. 

Greek lower-case pi
Project Operation – Example II

- **Relation $r$:**

<table>
<thead>
<tr>
<th>$A$</th>
<th>$B$</th>
<th>$C$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\alpha$</td>
<td>10</td>
<td>1</td>
</tr>
<tr>
<td>$\alpha$</td>
<td>20</td>
<td>1</td>
</tr>
<tr>
<td>$\beta$</td>
<td>30</td>
<td>1</td>
</tr>
<tr>
<td>$\beta$</td>
<td>40</td>
<td>2</td>
</tr>
</tbody>
</table>

  **Intuition:** The project operation removes duplicate rows, since relations are sets.

- **$\pi_{A,C}(r)$**

<table>
<thead>
<tr>
<th>$A$</th>
<th>$C$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\alpha$</td>
<td>1</td>
</tr>
<tr>
<td>$\alpha$</td>
<td>1</td>
</tr>
<tr>
<td>$\beta$</td>
<td>1</td>
</tr>
<tr>
<td>$\beta$</td>
<td>2</td>
</tr>
</tbody>
</table>

  Here there are two rows with $A = \alpha$ and $C = 1$. So one was discarded.
Project Operation

• Notation:

\[ \pi_{A_1, A_2, \ldots, A_k}(r) \]

where \( A_1, A_2 \) are attribute names and \( r \) is a relation name.

• The result is defined as the relation of \( k \) columns obtained by erasing the columns that are not listed.

• Duplicate rows removed from result, since relations are sets.
Union 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>$\beta$</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$A$</th>
<th>$B$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\alpha$</td>
<td>2</td>
</tr>
<tr>
<td>$\beta$</td>
<td>3</td>
</tr>
</tbody>
</table>

$r \cup s$:

<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>$\beta$</td>
<td>1</td>
</tr>
<tr>
<td>$\beta$</td>
<td>3</td>
</tr>
</tbody>
</table>

Intuition: The **union** operation concatenates two relations, and removes duplicate rows (since relations are sets).

Here there are two rows with $A = \alpha$ and $B = 2$. So one was discarded.
Union Operation

• Notation: \( r \cup s \)
• Defined as:
  \[
  r \cup s = \{ t \mid t \in r \text{ or } t \in s \}
  \]
  “Union-compatible”

For \( r \cup s \) to be valid.

1. \( r, s \) must have the *same arity* (same number of attributes)
2. The attribute domains must be *compatible* (e.g., 2\(^{nd}\) column of \( r \) deals with the same type of values as does the 2\(^{nd}\) column of \( s \)).

Although the field types must be the same, the names can be different. For example I can union *professor* and *lecturer* where:

```plaintext
professor(PID : string, name : string)
lecturer(LID : string, first_name : string)
```
Related Operation: Intersection

Relations $r, s$:

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\alpha$</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>$\alpha$</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>$\beta$</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

$\text{r}$

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\alpha$</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>$\beta$</td>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>

$s$

$r \cap s$:

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\alpha$</td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>

- Similar to Union operation.
- But Intersection is NOT one of the five basic operations.

- Intuition: The intersection operation computes the common rows between two relations.
Set Difference 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>$\beta$</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\alpha$</td>
<td>2</td>
</tr>
<tr>
<td>$\beta$</td>
<td>3</td>
</tr>
</tbody>
</table>

Intuition: The set difference operation returns all the rows that are in $r$ but not in $s$.
Set Difference Operation

• Notation \( r – s \)
• Defined as:
  \[
  r – s = \{ t \mid t \in r \text{ and } t \notin s \}
  \]
• Set differences must be taken between *compatible* relations.
  “Union-compatible”
  – \( r \) and \( s \) must have the *same arity*
  – attribute domains of \( r \) and \( s \) must be compatible
• Note that in general \( r – s \neq s – r \)
Cross-Product Operation-Example

Relations \( r, s \):

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \alpha )</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>( \beta )</td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \alpha )</td>
<td>10</td>
<td>a</td>
<td></td>
</tr>
<tr>
<td>( \beta )</td>
<td>10</td>
<td>a</td>
<td></td>
</tr>
<tr>
<td>( \beta )</td>
<td>20</td>
<td>b</td>
<td></td>
</tr>
<tr>
<td>( \gamma )</td>
<td>10</td>
<td>b</td>
<td></td>
</tr>
</tbody>
</table>

\( r \times S \):

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \alpha )</td>
<td>1</td>
<td>( \alpha )</td>
<td>10</td>
<td>a</td>
<td></td>
</tr>
<tr>
<td>( \alpha )</td>
<td>1</td>
<td>( \beta )</td>
<td>10</td>
<td>a</td>
<td></td>
</tr>
<tr>
<td>( \alpha )</td>
<td>1</td>
<td>( \beta )</td>
<td>20</td>
<td>b</td>
<td></td>
</tr>
<tr>
<td>( \alpha )</td>
<td>1</td>
<td>( \gamma )</td>
<td>10</td>
<td>b</td>
<td></td>
</tr>
<tr>
<td>( \beta )</td>
<td>2</td>
<td>( \alpha )</td>
<td>10</td>
<td>a</td>
<td></td>
</tr>
<tr>
<td>( \beta )</td>
<td>2</td>
<td>( \beta )</td>
<td>10</td>
<td>a</td>
<td></td>
</tr>
<tr>
<td>( \beta )</td>
<td>2</td>
<td>( \beta )</td>
<td>20</td>
<td>b</td>
<td></td>
</tr>
<tr>
<td>( \beta )</td>
<td>2</td>
<td>( \gamma )</td>
<td>10</td>
<td>b</td>
<td></td>
</tr>
</tbody>
</table>

Intuition: The cross product operation returns all possible combinations of rows in \( r \) with rows in \( S \).

In other words the result is every possible pairing of the rows of \( r \) and \( S \).
Cross-Product Operation

• Notation \( r \times s \)
• Defined as:

\[
    r \times s = \{ t q \mid t \in r \text{ and } q \in s \}
\]
• Assume that attributes of \( r(R) \) and \( s(S) \) are disjoint. (That is, \( R \cap S = \emptyset \)).
• If attributes names of \( r(R) \) and \( s(S) \) are not disjoint, then renaming must be used.
Composition of Operations

- We can build expressions using multiple operations
- Example: $\sigma_{A=C}(r \times s)$

```
\begin{array}{|c|c|}
\hline
A & B \\
\hline
\alpha & 1 \\
\beta & 2 \\
\hline
\end{array}
```

```
\begin{array}{|c|c|c|}
\hline
C & D & E \\
\hline
\alpha & 10 & a \\
\beta & 10 & a \\
\beta & 20 & b \\
\gamma & 10 & b \\
\hline
\end{array}
```

```
\begin{array}{|c|c|c|c|c|c|}
\hline
A & B & C & D & E \\
\hline
\alpha & 1 & \alpha & 10 & a & a \\
\alpha & 1 & \beta & 10 & a & b \\
\alpha & 1 & \beta & 20 & b & b \\
\beta & 1 & \gamma & 10 & a & a \\
\beta & 2 & \beta & 10 & a & b \\
\beta & 2 & \gamma & 20 & b & b \\
\beta & 2 & \gamma & 10 & a & b \\
\hline
\end{array}
```

"take the cross product of $r$ and $s$, then return only the rows where $A$ equals $B"
Rename Operation

- Allows us to name, and therefore to refer to, the results of relational-algebra expressions.

Example:

\[ \rho (myRelation, (r – s)) \]

Renaming columns (rename A to A2):

\[ \rho (myRelation(A->A2), (r – s)) \]

Take the set difference of \( r \) and \( s \), and call the result \( myRelation \)

Renaming in relational algebra is essentially the same as assignment in a programming language.
Rename Operation

If a relational-algebra expression $Y$ has arity $n$, then

$$\rho(X(A\rightarrow A1, B\rightarrow A2, \ldots), Y)$$

returns the result of expression $Y$ under the name $X$, and with the attributes renamed to $A1, A2, \ldots, An$.

For example,

$$\rho (myRelation(A\rightarrow E, B\rightarrow K), (r - s))$$

Take the set difference of $r$ and $s$, and call the result $myRelation$, while renaming the first field to $E$, and the second field to $K$. 
Sailors Example

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

- “Sailors” and “Reserves” relations for 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>10/10/96</td>
</tr>
<tr>
<td>58</td>
<td>103</td>
<td>11/12/96</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>58</td>
<td>rusty</td>
<td>10</td>
<td>35.0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>sid</th>
<th>sname</th>
<th>rating</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>28</td>
<td>yuppy</td>
<td>9</td>
<td>35.0</td>
</tr>
<tr>
<td>31</td>
<td>lubber</td>
<td>8</td>
<td>55.5</td>
</tr>
<tr>
<td>44</td>
<td>guppy</td>
<td>5</td>
<td>35.0</td>
</tr>
<tr>
<td>58</td>
<td>rusty</td>
<td>10</td>
<td>35.0</td>
</tr>
</tbody>
</table>
Algebra Operations

• Look what we want to get from the following table:

<table>
<thead>
<tr>
<th>sid</th>
<th>sname</th>
<th>rating</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>28</td>
<td>yuppy</td>
<td>9</td>
<td>35.0</td>
</tr>
<tr>
<td>31</td>
<td>lubber</td>
<td>8</td>
<td>55.5</td>
</tr>
<tr>
<td>44</td>
<td>guppy</td>
<td>5</td>
<td>35.0</td>
</tr>
<tr>
<td>58</td>
<td>rusty</td>
<td>10</td>
<td>35.0</td>
</tr>
</tbody>
</table>
Selection

- Selects rows that satisfy *selection condition*.
- No duplicates in result! (Why?)
- *Schema* of result identical to schema of (only) input relation.

\[
\sigma_{\text{rating} > 8}(S2) = 
\]

<table>
<thead>
<tr>
<th>sid</th>
<th>sname</th>
<th>rating</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>28</td>
<td>yuppy</td>
<td>9</td>
<td>35.0</td>
</tr>
<tr>
<td>31</td>
<td>lubber</td>
<td>8</td>
<td>55.5</td>
</tr>
<tr>
<td>44</td>
<td>guppy</td>
<td>5</td>
<td>35.0</td>
</tr>
<tr>
<td>58</td>
<td>rusty</td>
<td>10</td>
<td>35.0</td>
</tr>
</tbody>
</table>
Projection

- Deletes attributes that are not in projection list.
- **Schema** of result contains exactly the fields in the projection list, with the same names that they had in the (only) input relation.
- Projection operator has to eliminate duplicates! (Why??)
  - Note: real systems typically don’t do duplicate elimination unless the user explicitly asks for it.
Composition of Operations

- *Result* relation can be the *input* for another relational algebra operation! (*Operator composition*)

\[ \pi_{\text{Name}, \text{Rating}}(\sigma_{\text{Rating} > 8}(S2)) = \]

<table>
<thead>
<tr>
<th>sid</th>
<th>name</th>
<th>rating</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>28</td>
<td>uppy</td>
<td>9</td>
<td>35.0</td>
</tr>
<tr>
<td>31</td>
<td>lubber</td>
<td>8</td>
<td>55.5</td>
</tr>
<tr>
<td>44</td>
<td>guppy</td>
<td>5</td>
<td>35.0</td>
</tr>
<tr>
<td>58</td>
<td>rusty</td>
<td>10</td>
<td>35.0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>name</th>
<th>rating</th>
</tr>
</thead>
<tbody>
<tr>
<td>uppy</td>
<td>9</td>
</tr>
</tbody>
</table>
rusty| 10     |
What do we want to get from two relations?

<table>
<thead>
<tr>
<th>R1</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>sid</td>
<td>bid</td>
<td>day</td>
</tr>
<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>

<table>
<thead>
<tr>
<th>S1</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>sid</td>
<td>sname</td>
<td>rating</td>
<td>age</td>
</tr>
<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>

What about: Who reserved boat 101?

Or: Find the name of the sailor who reserved boat 101.
Cross-Product

- Each row of S1 is paired with each row of R1.
- Result schema has one field per field of S1 and R1, with field names inherited.

<table>
<thead>
<tr>
<th>sid1</th>
<th>sname</th>
<th>rating</th>
<th>age</th>
<th>sid2</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>10/10/96</td>
</tr>
<tr>
<td>22</td>
<td>dustin</td>
<td>7</td>
<td>45.0</td>
<td>58</td>
<td>103</td>
<td>11/12/96</td>
</tr>
<tr>
<td>31</td>
<td>lubber</td>
<td>8</td>
<td>55.5</td>
<td>22</td>
<td>101</td>
<td>10/10/96</td>
</tr>
<tr>
<td>31</td>
<td>lubber</td>
<td>8</td>
<td>55.5</td>
<td>58</td>
<td>103</td>
<td>11/12/96</td>
</tr>
<tr>
<td>58</td>
<td>rusty</td>
<td>10</td>
<td>35.0</td>
<td>22</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>58</td>
<td>103</td>
<td>11/12/96</td>
</tr>
</tbody>
</table>

Renaming operator (because of naming conflict):
\[ \rho(sid \rightarrow sid1, S1) \times \rho(sid \rightarrow sid2, R1) \]
Why does this cross product help

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

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

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

* Note my use of “temporary” relation Temp.
Another example

• Find the name of the sailor having the highest rating.

\[ \text{AllR} = \pi_{\text{ratingA}} \rho(\text{rating} \rightarrow \text{ratingA}, S2) \]

\[ \text{Result} = \pi_{\text{Sname}} (\sigma_{\text{rating} < \text{ratingA}} (S2 \times \text{AllR})) \]

What’s in “Result”? 

Does it answer our query? 

<table>
<thead>
<tr>
<th>sid</th>
<th>sname</th>
<th>rating</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>28</td>
<td>yuppy</td>
<td>9</td>
<td>35.0</td>
</tr>
<tr>
<td>31</td>
<td>lubber</td>
<td>8</td>
<td>55.5</td>
</tr>
<tr>
<td>44</td>
<td>guppy</td>
<td>5</td>
<td>35.0</td>
</tr>
<tr>
<td>58</td>
<td>rusty</td>
<td>10</td>
<td>35.0</td>
</tr>
</tbody>
</table>
\[
\text{S2} \quad \times \quad \text{AllR} = \quad \text{Result}
\]

\[
\text{AllR} = \pi_{\text{ratingA}} \rho(\text{rating} \rightarrow \text{ratingA}, \ S2)
\]

\[
\text{Result} = \pi_{\text{Sname}} (\sigma_{\text{rating} < \text{ratingA}} (S2 \times \text{AllR}))
\]
Union, Intersection, Set-Difference

- All of these operations take two input relations, which must be union-compatible:
  - Same number of fields.
  - ‘Corresponding’ fields have the same type.

- What is the schema of result?

<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>58</td>
<td>rusty</td>
<td>10</td>
<td>35.0</td>
</tr>
<tr>
<td>44</td>
<td>guppy</td>
<td>5</td>
<td>35.0</td>
</tr>
<tr>
<td>28</td>
<td>yuppy</td>
<td>9</td>
<td>35.0</td>
</tr>
</tbody>
</table>

\[
S_1 \cup S_2
\]

\[
S_1 \cap S_2
\]
Back to our query

• Find the name of the sailor having the highest rating.

\[
\text{AllR} = \pi_{\text{ratingA}} \rho(\text{rating} \rightarrow \text{ratingA}, S2)
\]

\[
\text{Tmp} = \pi_{\text{Sid, Sname}} (\sigma_{\text{rating} < \text{ratingA}} (S2 \times \text{AllR}))
\]

\[
\text{Result} = \pi_{\text{Sname}} (\pi_{\text{Sid, Sname}} (S2) - \text{Tmp})
\]

* Why not project on Sid only for Tmp?
Relational Algebra (So far)

• Basic operations:
  – **Selection** (σ) Selects a subset of rows from relation.
  – **Projection** (π) Deletes unwanted columns from relation.
  – **Cross-product** (×) Allows us to combine two relations.
  – **Set-difference** (-) Tuples in reln. 1, but not in reln. 2.
  – **Union** (∪) Tuples in reln. 1 and tuples in reln. 2.

Also,
  – **Rename** (ρ) Changes names of the attributes
  – **Intersection** (∩) 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 $$\bowtie$$ 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…

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

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: $r \bowtie s$

• 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 = $(A, B, C, D, E)$

  • $r \bowtie s$ is defined as:

  $$\pi_{r.A, r.B, r.C, r.D, s.E} (\sigma_{r.B = s.B \ r.D = s.D} (r \times s))$$
Natural Join Operation – Example

• Relation instances $r, s$:

$$
\begin{array}{cccc}
A & B & C & D \\
\alpha & 1 & \alpha & a \\
\beta & 2 & \gamma & a \\
\gamma & 4 & \beta & b \\
\alpha & 1 & \gamma & a \\
\delta & 2 & \beta & b \\
\end{array}
$$

$\text{r}$

$$
\begin{array}{ccc}
B & D & E \\
1 & a & \alpha \\
3 & a & \beta \\
1 & a & \gamma \\
2 & b & \delta \\
3 & b & \in \\
\end{array}
$$

$s$

$$
\begin{array}{cccc}
A & B & C & D & E \\
\alpha & 1 & \alpha & a & \alpha \\
\alpha & 1 & \alpha & a & \gamma \\
\alpha & 1 & \gamma & a & \alpha \\
\alpha & 1 & \gamma & a & \gamma \\
\delta & 2 & \beta & b & \delta \\
\end{array}
$$

$r \bowtie s$

How did we get here?

Lets 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>sid</th>
<th>name</th>
<th>rating</th>
<th>age</th>
<th>bid</th>
<th>day</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>22</td>
<td>dustin</td>
<td>7</td>
<td>45.0</td>
<td>101</td>
<td>10/10/96</td>
</tr>
<tr>
<td>31</td>
<td>31</td>
<td>lubber</td>
<td>8</td>
<td>55.5</td>
<td>103</td>
<td>11/12/96</td>
</tr>
<tr>
<td>58</td>
<td>58</td>
<td>rusty</td>
<td>10</td>
<td>35.0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Earlier We Saw…

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

\[ Temp = \rho (sid \rightarrow sid_1, S_1) \times \rho (sid \rightarrow sid_2, R_1) \]

\[ Result = \pi_{\text{Sname}} (\sigma_{sid_1 = sid_2 \land bid = 101} (Temp)) \]

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

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

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

Or

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

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.

$$ r \bowtie_c s = \sigma_c(r \times s) $$

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 can use a Conditional-Join to achieve this. The join condition is:

\[ r.\text{age} < s.\text{age} \text{ AND } r.\text{Marr-Lic} = s.\text{Marr-Lic} \]

Here are the tables for the relation \( r \) and \( s \):

<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>

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

#### Table $r$

<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 $s$

<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>

#### Equi-Join Operation

$r \bowtie_s r.marr-Lic = s.marr-Lic$

#### Result 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.