CompSci 516
Database Systems

Lecture 5
Relational Algebra and Relational Calculus

Instructor: Sudeepa Roy
Announcements (Thurs, 1/20)

• Do not forget your mask in class!
• Project details posted on Sakai
  – Standard, semi-standard, open options
• Let me know ASAP if you have not found a project team or in a < 4-member team
  – Team members due Tuesday 1/25
  – Proposal due Thursday 2/3
• HW1 due in < 2 weeks
  – Tuesday 2/1
  – No more extensions – please continue working on it!
• If you are not on Ed or Gradescope, let me know soon
Today’s topics

• Relational Algebra (RA) and Relational Calculus (RC)

• Reading material
  – [RG] Chapter 4 (RA, RC)
  – [GUW] Chapters 2.4, 5.1, 5.2

Acknowledgement:
The following slides have been created adapting the instructor material of the [RG] book provided by the authors Dr. Ramakrishnan and Dr. Gehrke.
Relational Query Languages
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 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:
  – Relational Algebra: More operational, very useful for representing execution plans
  – Relational Calculus: Lets users describe what they want, rather than how to compute it (Non-operational, declarative, or procedural)

• Note: Declarative (RC, SQL) vs. Operational (RA)
Preliminaries (recap)

• 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
    • 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

• Positional vs. named-field notation:
  – Positional notation easier for formal definitions, named-field notation more readable
Example Schema and Instances

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

<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>

<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>
Logic Notations

• $\exists$ There exists
• $\forall$ For all
• $\land$ Logical AND
• $\lor$ Logical OR
• $\neg$ NOT
• $\Rightarrow$ Implies
Relational Algebra (RA)
Relational Algebra

• Takes one or more relations as input, and produces a relation as output
  – operator
  – operand
  – semantic
  – so an algebra!

• Since each operation returns a relation, operations can be composed
  – Algebra is “closed”
Relational Algebra

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

• Additional operations:
  – Intersection \((\cap)\)
  – join \(\Join\)
  – division\((/\)\)
  – renaming \((\rho)\)
  – Not essential, but (very) useful.
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 (performance)
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>

<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>58</td>
<td>rusty</td>
<td>10</td>
<td>35.0</td>
</tr>
</tbody>
</table>
Composition of Operators

- Result relation can be the input for another relational algebra operation
  - Operator composition

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

\[
\pi_{sname, rating} (\sigma_{rating > 8} (S2))
\]

<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>58</td>
<td>rusty</td>
<td>10</td>
<td>35.0</td>
</tr>
</tbody>
</table>
Union, Intersection, Set-Difference

<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>

• All of these operations take two input relations, which must be union-compatible:
  – Same number of fields.
  – ‘Corresponding’ fields have the same type
  – Same schema as the inputs
### Union, Intersection, Set-Difference

<table>
<thead>
<tr>
<th>(S_1)</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>

<table>
<thead>
<tr>
<th>(S_2)</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>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>

- **Note:** no duplicate
  - “Set semantic”
  - SQL: **UNION**
  - SQL allows “bag semantic” as well: **UNION ALL**
## Union, Intersection, Set-Difference

### $S_1$

<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>

### $S_2$

<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>

### $S_1 - S_2$

<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>58</td>
<td>rusty</td>
<td>10</td>
<td>35.0</td>
</tr>
</tbody>
</table>

### $S_1 \cap S_2$

<table>
<thead>
<tr>
<th>sid</th>
<th>sname</th>
<th>rating</th>
<th>age</th>
</tr>
</thead>
<tbody>
<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>
Cross-Product

- Each row of S1 is paired with each row of R.
- Result schema has one field per field of S1 and R, with field names `inherited` if possible.
  - Conflict: Both S1 and R have a field called `sid`.

<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>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>
Renaming Operator $\rho$

$$(\rho_{\text{sid} \rightarrow \text{sid1}} \ S1) \times (\rho_{\text{sid} \rightarrow \text{sid1}} \ R1)$$

or

$$\rho(C(1 \rightarrow \text{sid1}, 5 \rightarrow \text{sid2}), \ S1 \times R1)$$

<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>10/10/96</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>

- In general, can use $\rho(<\text{Temp}>, <\text{RA-expression}>)$
Joins

\[ R \bowtie_c S = \sigma_c (R \times S) \]

<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>103</td>
<td>11/12/96</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>
</tbody>
</table>

- Result schema same as that of cross-product.
- Fewer tuples than cross-product, might be able to compute more efficiently
Find names of sailors who’ve reserved boat #103

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

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

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

• Solution 2: \[ \pi_{sname}(\sigma_{bid=103}(\text{Reserves} \bowtie \text{Sailors})) \]
Expressing an RA expression as a Tree

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

Also called a logical query plan

 End of Lecture-5
Find sailors who’ve reserved a red or a green boat

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

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

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

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

Can also define Tempboats using union
Try the “AND” version yourself
What about aggregates?

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

• Extended relational algebra
• \( \gamma_{\text{age, avg(rating)}} \rightarrow \text{avgr} \) Sailors
• Also extended to “bag semantic”: allow duplicates
  – Take into account cardinality
  – R and S have tuple t resp. m and n times
  – \( R \cup S \) has t m+n times
  – \( R \cap S \) has t \( \min(m, n) \) times
  – \( R - S \) has t \( \max(0, m-n) \) times
  – sorting(\( \tau \)), duplicate removal (\( \delta \)) operators
Relational Calculus (RC)
Relational Calculus

- Equivalent to “First-Order Logic”
- RA is procedural
  - $\pi_A(\sigma_{A=a} R)$ and $\sigma_{A=a}(\pi_A R)$ are equivalent but different expressions
- RC
  - non-procedural and declarative
  - describes a set of answers without being explicit about how they should be computed
- TRC (tuple relational calculus)
  - variables correspond to tuples:
    \[
    \{P \mid \exists S \in \text{Sailors} (S.Name = \text{‘Bob’}) \land P.Sid = S.Sid\}
    \]
  - we will primarily do TRC
- DRC (domain relational calculus)
  - variables range over attribute values, equivalent to TRC
    \[
    \{x \mid \exists y, z (x, \text{‘Bob’}, y, z) \in \text{Sailors}\}
    \]
    or \[
    \{x \mid \exists y, z, w (x, w, y, z) \in \text{Sailors} \land w = \text{‘Bob’}\}
    \]
    or \[
    \{x \mid \exists y, z, w \text{Sailors}(x, w, y, z) \land w = \text{‘Bob’}\}
    \]
TRC: example

\[ \exists S \in \text{Sailors} \ (S.\text{rating} > 7 \land P.\text{sname} = S.\text{sname} \land P.\text{age} = S.\text{age}) \]

- Find the name and age of all sailors with a rating above 7

- P is a tuple variable
  - with exactly two fields sname and age (schema of the output relation)
  - \( P.\text{sname} = S.\text{sname} \land P.\text{age} = S.\text{age} \) gives values to the fields of an answer tuple

- Use parentheses, \( \forall \exists \land \lor \land > < = \neq \neg \) etc as necessary

- A \( \Rightarrow \) B is very useful too
  - next slide
A \Rightarrow B

• A “implies” B
• Equivalently, if A is true, B must be true
• Equivalently, \neg A \lor B, i.e.
  – either A is false (then B can be anything)
  – otherwise (i.e. A is true) B must be true
Useful Logical Equivalences

- $\forall x \; P(x) = \neg \exists x \; [\neg P(x)]$
- $\neg (P \lor Q) = \neg P \land \neg Q$
- $\neg (P \land Q) = \neg P \lor \neg Q$
- Similarly, $\neg (\neg P \lor Q) = P \land \neg Q$ etc.
- $A \implies B = \neg A \lor B$
TRC: example

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

• Find the names of sailors who have reserved at least two boats
TRC: example

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

• Find the names of sailors who have reserved at least two boats

\{ P \mid \exists S \in \text{Sailors} (\exists R1 \in \text{Reserves} \exists R2 \in \text{Reserves} (S.sid = R1.sid \land S.sid = R2.sid \land R1.bid \neq R2.bid) \land P.sname = S.sname) \}
TRC: example

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

• Find the names of sailors who have reserved all boats
TRC: example

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

• Find the names of sailors who have reserved all boats

\{P \ | \ \exists S \in \text{Sailors} \ [\forall B \in \text{Boats} \ (\exists R \in \text{Reserves} \ (S.\text{sid} = R.\text{sid} \land R.\text{bid} = B.\text{bid}))] \land (P.\text{sname} = S.\text{sname})\}
• Find the names of sailors who have reserved all red boats
**TRC: example**

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

• Find the names of sailors who have reserved all red boats

\[
\{P \mid \exists S \in \text{Sailors} \ (\forall B \in \text{Boats} \ (B.\text{color} = 'red' \Rightarrow (\exists R \in \text{Reserves} \ (S.\text{sid} = R.\text{sid} \land R.\text{bid} = B.\text{bid})))) \land P.\text{sname} = S.\text{sname})\}
\]

Recall that \(A \Rightarrow B\) is logically equivalent to \(\neg A \lor B\)
so \(\Rightarrow\) can be avoided, but it is cleaner and more intuitive

Feel free to use \(\neg A \lor B\)
TRC & DRC: example

Sailors(sid, sname, rating, age)
Boats(bid, bname, color)
Reserves(sid, bid, day)
• Find the name and age of all sailors with a rating above 7

TRC:
{P | \exists S \in \text{Sailors} \ (S.\text{rating} > 7 \land P.\text{name} = S.\text{name} \land P.\text{age} = S.\text{age})}

DRC:
{<N, A> | \exists <I, N, T, A> \in \text{Sailors} \land T > 7}

• Variables are now domain variables
• We will use use TRC
  – both are equivalent
The famous “Beers” database

Drinkers **Frequent** Bars
“X” times a week

Bars
Each has an address

Bars **Serve** Beers
At price “Y”

Drinkers **Likes** Beers

Drinkers
Each has an address

Beers
Each has a brewer
### Beers as a Relational Database

#### Bar

<table>
<thead>
<tr>
<th>name</th>
<th>address</th>
</tr>
</thead>
<tbody>
<tr>
<td>The Edge</td>
<td>108 Morris Street</td>
</tr>
<tr>
<td>Satisfaction</td>
<td>905 W. Main Street</td>
</tr>
</tbody>
</table>

#### Beer

<table>
<thead>
<tr>
<th>Name</th>
<th>Brewer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Budweiser</td>
<td>Anheuser-Busch Inc.</td>
</tr>
<tr>
<td>Corona</td>
<td>Grupo Modelo</td>
</tr>
<tr>
<td>Dixie</td>
<td>Dixie Brewing</td>
</tr>
</tbody>
</table>

#### Drinker

<table>
<thead>
<tr>
<th>name</th>
<th>address</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amy</td>
<td>100 W. Main Street</td>
</tr>
<tr>
<td>Ben</td>
<td>101 W. Main Street</td>
</tr>
<tr>
<td>Dan</td>
<td>300 N. Duke Street</td>
</tr>
</tbody>
</table>

#### Serves

<table>
<thead>
<tr>
<th>bar</th>
<th>beer</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>The Edge</td>
<td>Budweiser</td>
<td>2.50</td>
</tr>
<tr>
<td>The Edge</td>
<td>Corona</td>
<td>3.00</td>
</tr>
<tr>
<td>Satisfaction</td>
<td>Budweiser</td>
<td>2.25</td>
</tr>
</tbody>
</table>

#### Frequents

<table>
<thead>
<tr>
<th>drinker</th>
<th>bar</th>
<th>times_a_week</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ben</td>
<td>Satisfaction</td>
<td>2</td>
</tr>
<tr>
<td>Dan</td>
<td>The Edge</td>
<td>1</td>
</tr>
<tr>
<td>Dan</td>
<td>Satisfaction</td>
<td>2</td>
</tr>
</tbody>
</table>

#### Likes

<table>
<thead>
<tr>
<th>drinker</th>
<th>beer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amy</td>
<td>Corona</td>
</tr>
<tr>
<td>Dan</td>
<td>Budweiser</td>
</tr>
<tr>
<td>Dan</td>
<td>Corona</td>
</tr>
<tr>
<td>Ben</td>
<td>Budweiser</td>
</tr>
</tbody>
</table>

See online database for more tuples.
More Examples: RC

UNDERSTAND THE DIFFERENCE IN ANSWERS FOR ALL FOUR DRINKERS

Drinker Category 1

Find drinkers that frequent some bar that serves some beer they like.

…
Drinker Category 1

Find drinkers that frequent some bar that serves some beer they like.

\{x \mid \exists F \in \text{Frequents} \ (F\.drinker = x\.drinker \land \exists S \in \text{Serves} \ \exists L \in \text{Likes} \\
(F\.drinker = L\.drinker) \land (F\.bar = S\.bar) \land (S\.beer = L\.beer)\}\}
Find drinkers that frequent some bar that serves some beer they like.

\[
\{ x \mid \exists F \in \text{Frequents} \ (F.\text{drinker} = x.\text{drinker} \land \exists S \in \text{Serves} \ \exists L \in \text{Likes} \\
\quad (F.\text{drinker} = L.\text{drinker}) \land (F.\text{bar} = S.\text{bar}) \land (S.\text{beer} = L.\text{beer})) \}
\]

Find drinkers that frequent only bars that serves some beer they like.

\[
\text{...}
\]
Drinker Category 2

Find drinkers that frequent **some** bar that serves **some** beer they like.

\{x \mid \exists F \in \text{Frequents} \ (F.\text{drinker} = x.\text{drinker}) \land \exists S \in \text{Serves} \ \exists L \in \text{Likes} \ (F.\text{drinker} = L.\text{drinker}) \land (F.\text{bar} = S.\text{bar}) \land (S.\text{beer} = L.\text{beer})\}\}

Find drinkers that frequent **only** bars that serve **some** beer they like.

\{x \mid \exists F \in \text{Frequents} \ (F.\text{drinker} = x.\text{drinker}) \land [ \forall F1 \in \text{Frequents} \ (F.\text{drinker} = F1.\text{drinker}) \Rightarrow \exists S \in \text{Serves} \ \exists L \in \text{Likes} \ [(F1.\text{bar} = S.\text{bar}) \land (F1.\text{drinker} = L.\text{drinker}) \land (S.\text{beer} = L.\text{beer})] ]\}\}
Find drinkers that frequent **some** bar that serves **some** beer they like.

\[
\{ x \mid \exists F \in \text{Frequents} \ (F.\text{drinker} = x.\text{drinker}) \land \exists S \in \text{Serves} \ \exists L \in \text{Likes} \\
\text{(F.\text{drinker} = L.\text{drinker})} \land (F.\text{bar} = S.\text{bar}) \land (S.\text{beer} = L.\text{beer}) \} 
\]

Find drinkers that frequent **only** bars that serve **some** beer they like.

\[
\{ x \mid \exists F \in \text{Frequents} \ (F.\text{drinker} = x.\text{drinker}) \land [ \ \forall F1 \in \text{Frequents} \ (F.\text{drinker} = F1.\text{drinker}) \\
\Rightarrow \exists S \in \text{Serves} \ \exists L \in \text{Likes} \ [(F1.\text{bar} = S.\text{bar}) \land (F1.\text{drinker} = L.\text{drinker}) \land (S.\text{beer} = L.\text{beer})] \} 
\]

Find drinkers that frequent **some** bar that serves **only** beers they like.

\[
\ldots 
\]
Drinker Category 3

Find drinkers that frequent some bar that serves some beer they like.

\[
\{x \mid \exists F \in \text{Frequents} \ (F.\text{drinker} = x.\text{drinker}) \land \exists S \in \text{Serves} \ \exists L \in \text{Likes} \\
\quad (F.\text{drinker} = L.\text{drinker}) \land (F.\text{bar} = S.\text{bar}) \land (S.\text{beer} = L.\text{beer})\}\]

Find drinkers that frequent only bars that serve some beer they like.

\[
\{x \mid \exists F \in \text{Frequents} \ (F.\text{drinker} = x.\text{drinker}) \land [ \ \forall F1 \in \text{Frequents} \ (F.\text{drinker} = F1.\text{drinker}) \Rightarrow \exists S \in \text{Serves} \ \exists L \in \text{Likes} [ (F1.\text{bar} = S.\text{bar}) \land (F1.\text{drinker} = L.\text{drinker}) \land (S.\text{beer} = L.\text{beer})] ]\}

Find drinkers that frequent some bar that serves only beers they like.

\[
\{x \mid \exists F \in \text{Frequents} \ (F.\text{drinker} = x.\text{drinker}) \land [ \ \forall S \in \text{Serves} \ (F.\text{bar} = S.\text{bar}) \Rightarrow \exists L \in \text{Likes} [ (F.\text{drinker} = L.\text{drinker}) \land (S.\text{beer} = L.\text{beer})] ]\}
Find drinkers that frequent **some** bar that serves **some** beer they like.

\[
\{ x \mid \exists F \in \text{Frequents} \ (F.\text{drinker} = x.\text{drinker}) \land \exists S \in \text{Serves} \ \exists L \in \text{Likes} \\
(\exists S \in \text{Serves} \ \exists L \in \text{Likes} \ (F.\text{drinker} = L.\text{drinker}) \land (F.\text{bar} = S.\text{bar}) \land (S.\text{beer} = L.\text{beer})) \}
\]

Find drinkers that frequent **only** bars that serve **some** beer they like.

\[
\{ x \mid \exists F \in \text{Frequents} \ (F.\text{drinker} = x.\text{drinker}) \land \forall F1 \in \text{Frequents} \ (F.\text{drinker} = F1.\text{drinker}) \\
\Rightarrow \exists S \in \text{Serves} \ \exists L \in \text{Likes} \ ((F1.\text{bar} = S.\text{bar}) \land (F1.\text{drinker} = L.\text{drinker}) \land (S.\text{beer} = L.\text{beer})) \}
\]

Find drinkers that frequent **some** bar that serves **only** beers they like.

\[
\{ x \mid \exists F \in \text{Frequents} \ (F.\text{drinker} = x.\text{drinker}) \land \forall S \in \text{Serves} \ (F.\text{bar} = S.\text{bar}) \Rightarrow \\
\exists L \in \text{Likes} \ ((F.\text{drinker} = L.\text{drinker}) \land (S.\text{beer} = L.\text{beer})) \}
\]

Find drinkers that frequent **only** bars that serve **only** beer they like.

...
Drinker Category 4

Find drinkers that frequent **some** bar that serves **some** beer they like.

\[
\{ x \mid \exists F \in \text{Frequents} \ (F.\text{drinker} = x.\text{drinker}) \land \exists S \in \text{Serves} \ \exists L \in \text{Likes} \\
(F.\text{drinker} = L.\text{drinker}) \land (F.\text{bar} = S.\text{bar}) \land (S.\text{beer} = L.\text{beer}) \}\}
\]

Find drinkers that frequent **only** bars that serve **some** beer they like.

\[
\{ x \mid \exists F \in \text{Frequents} \ (F.\text{drinker} = x.\text{drinker}) \land [ \ \forall F1 \in \text{Frequents} \ (F.\text{drinker} = F1.\text{drinker}) \\
\Rightarrow \exists S \in \text{Serves} \ \exists L \in \text{Likes} [(F1.\text{bar} = S.\text{bar}) \land (F1.\text{drinker} = L.\text{drinker}) \land (S.\text{beer} = L.\text{beer})] ] \}
\]

Find drinkers that frequent **some** bar that serves **only** beers they like.

\[
\{ x \mid \exists F \in \text{Frequents} \ (F.\text{drinker} = x.\text{drinker}) \land [ \ \forall S \in \text{Serves} \ (F.\text{bar} = S.\text{bar}) \Rightarrow \exists L \in \text{Likes} [(F.\text{drinker} = L.\text{drinker}) \land (S.\text{beer} = L.\text{beer})] ] \}
\]

Find drinkers that frequent **only** bars that serve **only** beer they like.

\[
\{ x \mid \exists F \in \text{Frequents} \ (F.\text{drinker} = x.\text{drinker}) \land [ \ \forall F1 \in \text{Frequents} \ (F.\text{drinker} = F1.\text{drinker}) \\
\Rightarrow [ \ \forall S \in \text{Serves} \ (F1.\text{bar} = S.\text{bar}) \Rightarrow \exists L \in \text{Likes} [(F.\text{drinker} = L.\text{drinker}) \land (S.\text{beer} = L.\text{beer})] ] ] \}
\]
Why should we care about RC

- RC expression may be much simpler than SQL queries
  - and easier to check for correctness than SQL
  - power to use $\forall$ and $\Rightarrow$
  - then you can systematically go to a “correct” SQL or RA query (example coming soon)

- Note:
  - RC is declarative, like SQL, and unlike RA (which is operational)
  - Gives foundation of database queries in first-order logic
  - you cannot express all aggregates in RC, e.g., cardinality of a relation or sum (possible in extended RA and SQL)
  - still can express conditions like “at least two tuples” (or any constant)
From RC to SQL

Query: Find drinkers that like some beer (so much) that they frequent all bars that serve it

\{x \mid \exists L \in \text{Likes}\ (L.drinker = x.drinker) \land [\ \forall S \in \text{Serves}\ (L.beer = S.beer) \Rightarrow \\
\exists F \in \text{Frequents}\ [(F.drinker = L.drinker) \land (F.bar = S.bar)] ]}
Query: Find drinkers that like some beer so much that they frequent all bars that serve it

\[ \{ x | \exists L \in \text{Likes} (L.\text{drinker} = x.\text{drinker}) \land [ \forall S \in \text{Serves} [ (L.\text{beer} = S.\text{beer}) \implies \exists F \in \text{Frequents} [(F.\text{drinker} = L.\text{drinker}) \land (F.\text{bar} = S.\text{bar})] ] ] \} \]

\[ \equiv \{ x | \exists L \in \text{Likes} (L.\text{drinker} = x.\text{drinker}) \land [ \forall S \in \text{Serves} [ \neg (L.\text{beer} = S.\text{beer}) \lor [ \exists F \in \text{Frequents} [(F.\text{drinker} = L.\text{drinker}) \land (F.\text{bar} = S.\text{bar})] ] ] \} \]

Step 1: Replace \( \forall \) with \( \exists \) using de Morgan’s Laws

\[ Q(x) = \exists y. \text{Likes}(x, y) \land [ \neg \exists S \in \text{Serves} [(L.\text{beer} = S.\text{beer}) \land \neg [ \exists F \in \text{Frequents} [(F.\text{drinker} = L.\text{drinker}) \land (F.\text{bar} = S.\text{bar})] ] ] ] \]

SQL or RA does not have \( \forall \)!

Now you got all \( \exists \) and \( \neg \) expressible in RA/SQL
Query: Find drinkers that like some beer so much that they frequent all bars that serve it

$$\exists L \in \text{Likes} \land \neg \exists S \in \text{Serves} [(L.\text{beer} = S.\text{beer}) \land \neg \exists F \in \text{Frequents} [(F.\text{drinker} = L.\text{drinker}) \land (F.\text{bar} = S.\text{bar})]]$$

Step 2: Translate into SQL

```
SELECT DISTINCT L.drinker
FROM Likes L
WHERE not exists
  (SELECT S.bar
   FROM Serves S
   WHERE L.beer=S.beer
   AND not exists (SELECT *
                    FROM Frequents F
                    WHERE F.drinker=L.drinker
                          AND F.bar=S.bar))
```
• You learnt three query languages for the Relational DB model
  – SQL
  – RA
  – RC

• All have their own purposes

• You should be able to write a query in all three languages and
  convert from one to another
  – However, you have to be careful, not all “valid” expressions in one may
    be expressed in another
  – \{S \mid \neg (S \in \text{Sailors})\} – infinitely many tuples – an “unsafe” query
  – More when we do “Datalog”, also see Ch. 4.4 in [RG]
Announcements (Tues, 1/25)

• Team info due today on gradescope
  – One “group submission” per team (add everyone’s name)
  – Graded as Communication (2% total – everything that does not belong to other categories)

• HW1 due next week 2/1 (Tues)
  – Check out Ed for questions and discussions

• Quizzes start from this week!
  – In-class component (attempt in class = full point, discussed in class) and take-home component (1 week)
  – Useful for preparing for exams
  – Lowest score will be dropped