CompSci 516
Database Systems

Lecture 4
Relational Algebra
and
Relational Calculus

Instructor: Sudeepa Roy
Announcements

• Reminder: HW1
  – Sakai: Resources -> HW -> HW1 folder
  – Due on 09/20 (Thurs), 11:55 pm, no late days
  – Start now!
  – Submission instructions for gradescope to be updated (will be notified through piazza)

• Your piazza and sakai accounts should be active
  – Last call! will use piazza for pop-up quizzes
  – if not on piazza, send me an email
  – Install piazza app on your phone (or bring a laptop)
Recap: SQL -- Lecture 2/3

- Creating/modifying relations
- Specifying integrity constraints
- Key/candidate key, superkey, primary key, foreign key
- Conceptual evaluation of SQL queries
- Joins
- Group bys and aggregates
- Nested queries
- NULLs
- Views

On whiteboard:
From last lecture
Correct/incorrect group-by queries
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>yappy</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

\[ \sigma_{\text{rating} > 8}(S2) \]
Composition of Operators

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

<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>

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

<table>
<thead>
<tr>
<th>sname</th>
<th>rating</th>
</tr>
</thead>
<tbody>
<tr>
<td>yuppy</td>
<td>9</td>
</tr>
<tr>
<td>rusty</td>
<td>10</td>
</tr>
</tbody>
</table>

\[ \pi_{\text{sname}, \text{rating}}(\sigma_{\text{rating} > 8}(S2)) \]
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
  - same schema as the inputs
Union, Intersection, Set-Difference

Note: no duplicate

- “Set semantic”
- SQL: UNION
- SQL allows “bag semantic” as well: UNION ALL

<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>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>28</td>
<td>yuppy</td>
<td>9</td>
<td>35.0</td>
</tr>
</tbody>
</table>
### 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>

<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>
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>

\[ S1 \bowtie_{S1.sid < R1.sid} R1 \]

- 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

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

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

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

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

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

- 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

• **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 take tuples as values
  - we will primarily do TRC

• **DRC (domain relational calculus)**
  - variables range over field values
TRC: 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

{P | ∃ S ∈ Sailors (S.rating > 7 ∧ P.sname = S.sname ∧ P.age = S.age)}

• P is a tuple variable
  – with exactly two fields sname and age (schema of the output relation)
  – P.sname = S.sname ∧ P.age = S.age gives values to the fields of an answer tuple

• Use parentheses, ∀ ∃ ∨ ∧ > < = ≠ ¬ etc as necessary

• A ⇒ B is very useful too
  – next slide
A \implies 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.

• $A \Rightarrow B = \neg A \lor B$
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$(sid, \text{ sname}, \text{ rating}, \text{ age})$
Boats$(bid, \text{ bname}, \text{ color})$
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
• Called the “Division” operation
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
• Division operation in RA!

\{P \mid \exists S \in \text{Sailors} [ \forall B \in \text{Boats} ( \exists R \in \text{Reserves} (S.sid = R.sid \land R.bid = B.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 red boats

How will you change the previous TRC expression?
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\.color = 'red') \Rightarrow (\exists R \in \text{Reserves} \ (S\.sid = R\.sid \land R\.bid = B\.bid)) \land P\.sname = S\.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
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 Sailors (S.rating > 7 \land P.name = S.name \land P.age = S.age)}

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

• Variables are now domain variables
• We will use use TRC
  – both are equivalent
• Another option to write coming soon!
More Examples: RC

• The famous “Drinker-Beer-Bar” example!

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.

\[ Q(x) = \exists y. \exists z. \text{Frequents}(x, y) \land \text{Serves}(y, z) \land \text{Likes}(x, z) \]

a shortcut for
\[ \{ x \mid \exists Y \in \text{Frequents} \exists Z \in \text{Serves} \exists W \in \text{Likes} ((T.\text{drinker} = x.\text{drinker}) \land (T.\text{bar} = Z.\text{bar}) \land (W.\text{beer} = Z.\text{beer}) \land (Y.\text{drinker} = W.\text{drinker}) \} \]

The difference is that in the first one, one variable = one attribute
in the second one, one variable = one tuple (Tuple RC)
Both are equivalent and feel free to use the one that is convenient to you
Drinker Category 2

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

\[ Q(x) = \exists y. \exists z. \text{Frequents}(x, y) \land \text{Serves}(y, z) \land \text{Likes}(x, z) \]

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

\[ Q(x) = \ldots \]
Drinker Category 2

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

$$Q(x) = \exists y. \exists z. \text{Frequents}(x, y) \land \text{Serves}(y, z) \land \text{Likes}(x, z)$$

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

$$Q(x) = \forall y. \text{Frequents}(x, y) \Rightarrow (\exists z. \text{Serves}(y, z) \land \text{Likes}(x, z))$$
Find drinkers that frequent \textit{some} bar that serves \textit{some} beer they like.

\[ Q(x) = \exists y. \exists z. \text{Frequents}(x, y) \land \text{Serves}(y, z) \land \text{Likes}(x, z) \]

Find drinkers that frequent \textit{only} bars that serve \textit{some} beer they like.

\[ Q(x) = \forall y. \text{Frequents}(x, y) \implies (\exists z. \text{Serves}(y, z) \land \text{Likes}(x, z)) \]

Find drinkers that frequent \textit{some} bar that serves \textit{only} beers they like.

\[ Q(x) = \ldots \]
Find drinkers that frequent **some** bar that serves **some** beer they like.

\[ Q(x) = \exists y. \exists z. \text{Frequents}(x, y) \land \text{Serves}(y, z) \land \text{Likes}(x, z) \]

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

\[ Q(x) = \forall y. \text{Frequents}(x, y) \Rightarrow (\exists z. \text{Serves}(y, z) \land \text{Likes}(x, z)) \]

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

\[ Q(x) = \exists y. \text{Frequents}(x, y) \land \forall z. (\text{Serves}(y, z) \Rightarrow \text{Likes}(x, z)) \]
Drinker Category 4

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

\[ Q(x) = \exists y. \exists z. \text{Frequents}(x, y) \land \text{Serves}(y, z) \land \text{Likes}(x, z) \]

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

\[ Q(x) = \forall y. \text{Frequents}(x, y) \Rightarrow (\exists z. \text{Serves}(y, z) \land \text{Likes}(x, z)) \]

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

\[ Q(x) = \exists y. \text{Frequents}(x, y) \land \forall z. (\text{Serves}(y, z) \Rightarrow \text{Likes}(x, z)) \]

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

\[ Q(x) = \ldots \]
Find drinkers that frequent some bar that serves some beer they like.

$$Q(x) = \exists y. \exists z. \text{Frequents}(x, y) \land \text{Serves}(y, z) \land \text{Likes}(x, z)$$

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

$$Q(x) = \forall y. \text{Frequents}(x, y) \Rightarrow (\exists z. \text{Serves}(y, z) \land \text{Likes}(x, z))$$

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

$$Q(x) = \exists y. \text{Frequents}(x, y) \land \forall z. (\text{Serves}(y, z) \Rightarrow \text{Likes}(x, z))$$

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

$$Q(x) = \forall y. \text{Frequents}(x, y) \Rightarrow \forall z. (\text{Serves}(y, z) \Rightarrow \text{Likes}(x, z))$$
Why should we care about RC

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

\[ Q(x) = \exists y. \text{Likes}(x, y) \land \forall z. (\text{Serves}(z, y) \Rightarrow \text{Frequents}(x, z)) \]
From RC to SQL

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

$$Q(x) = \exists y. \text{Likes}(x, y) \land \forall z. (\text{Serves}(z, y) \implies \text{Frequents}(x, z))$$

$$\equiv Q(x) = \exists y. \text{Likes}(x, y) \land \forall z. (\neg \text{Serves}(z, y) \lor \text{Frequents}(x, z))$$

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

$$Q(x) = \exists y. \text{Likes}(x, y) \land \neg \exists z. (\text{Serves}(z, y) \land \neg\text{Frequents}(x, z))$$
From RC to SQL

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

\[ Q(x) = \exists y. \text{Likes}(x, y) \land \neg \exists z. (\text{Serves}(z, y) \land \neg \text{Frequents}(x, z)) \]

Step 2: Translate into SQL

```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))
```

We will see a “methodical and correct” translation through “safe queries” in Datalog
Summary

• 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]