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)

S1

sid	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0

S2

sid	sname	rating	age
28	yuppy	9	35.0
31	lubber	8	55.5
44	guppy	5	35.0
58	rusty	10	35.0

R1

sid	bid	day
22	101	10/10/96
58	103	11/12/96

Logic Notations

- $\exists$ There exists
- $\forall$ For all
- $\wedge$ Logical AND
- $\vee$ 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 (σ) Selects a subset of rows from relation
  - Projection (π) 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 (∪) Tuples in reln. 1 or in reln. 2.
- Additional operations:
  - Intersection (∩)
  - join ⋈
  - division(/)
  - renaming (ρ)
  - Not essential, but (very) useful.

Projection

S2

<u>sid</u>	sname	rating	age
28	yuppy	9	35.0
31	lubber	8	55.5
44	guppy	5	35.0
58	rusty	10	35.0

- 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)

sname	rating
yuppy	9
lubber	8
guppy	5
rusty	10

$$\pi_{sname,rating}(S2)$$

age
35.0
55.5

$$\pi_{age}(S2)$$

Selection

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

S2

sid	sname	rating	age
28	yuppy	9	35.0
31	lubber	8	55.5
44	guppy	5	35.0
58	rusty	10	35.0

sid	sname	rating	age
28	yuppy	9	35.0
58	rusty	10	35.0

$$\sigma_{rating>8}(S2)$$

Composition of Operators

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

sid	sname	rating	age
28	yuppy	9	35.0
58	rusty	10	35.0

$$\sigma_{rating>8}(S2)$$

sname	rating
yuppy	9
rusty	10

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

Union, Intersection, Set-Difference

S1

sid	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0

S2

sid	sname	rating	age
28	yuppy	9	35.0
31	lubber	8	55.5
44	guppy	5	35.0
58	rusty	10	35.0

- 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

sid	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0
44	guppy	5	35.0
28	yuppy	9	35.0

$S1 \cup S2$

Union, Intersection, Set-Difference

S1

sid	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0

S2

sid	sname	rating	age
28	yuppy	9	35.0
31	lubber	8	55.5
44	guppy	5	35.0
58	rusty	10	35.0

- Note: no duplicate
  - "Set semantic"
  - SQL: UNION
  - SQL allows "bag semantic" as well: UNION ALL

sid	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0
44	guppy	5	35.0
28	yuppy	9	35.0

$S1 \cup S2$

Union, Intersection, Set-Difference

S1

sid	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0

S2

sid	sname	rating	age
28	yuppy	9	35.0
31	lubber	8	55.5
44	guppy	5	35.0
58	rusty	10	35.0

sid	sname	rating	age
22	dustin	7	45.0

$S1 - S2$

sid	sname	rating	age
31	lubber	8	55.5
58	rusty	10	35.0

$S1 \cap S2$

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.

sid	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0

sid	bid	day
22	101	10/10/96
58	103	11/12/96

(sid)	sname	rating	age	(sid)	bid	day
22	dustin	7	45.0	22	101	10/10/96
22	dustin	7	45.0	58	103	11/12/96
31	lubber	8	55.5	22	101	10/10/96
31	lubber	8	55.5	58	103	11/12/96
58	rusty	10	35.0	22	101	10/10/96
58	rusty	10	35.0	58	103	11/12/96

Renaming Operator ρ

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

or

$\rho(C(1 \to sid1, 5 \to sid2), S1 \times R1)$

C is the new relation name

(sid)	sname	rating	age	(sid)	bid	day
22	dustin	7	45.0	22	101	10/10/96
22	dustin	7	45.0	58	103	11/12/96
31	lubber	8	55.5	22	101	10/10/96
31	lubber	8	55.5	58	103	11/12/96
58	rusty	10	35.0	22	101	10/10/96
58	rusty	10	35.0	58	103	11/12/96

- In general, can use ρ(<Temp>, <RA-expression>)

Joins

$$R \bowtie_c S = \sigma_c (R \times S)$$

(sid)	sname	rating	age	(sid)	bid	day
22	dustin	7	45.0	58	103	11/12/96
31	lubber	8	55.5	58	103	11/12/96

$$S1 \bowtie_{S1.sid < R1.sid} R1$$

- 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(<u>sid</u>, sname, rating, age)

Boats(<u>bid</u>, bname, color)

Reserves(<u>sid, bid, day</u>)

Find names of sailors who've reserved boat #103

Sailors(<u>sid</u>, sname, rating, age)

Boats(<u>bid</u>, bname, color)

Reserves(<u>sid, bid, day</u>)

No join conditions?
"Natural Join"
= on all common attributes
+
Duplicate columns removed

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

- Solution 2: $\pi_{sname}(\sigma_{bid=103}(\text{Reserves} \bowtie Sailors))$

Expressing an RA expression as a Tree

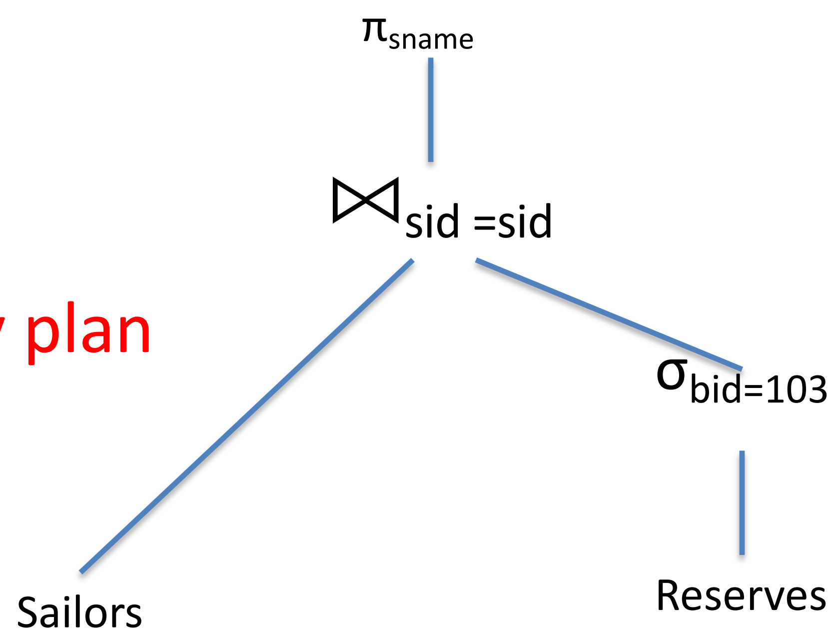
Sailors(sid, sname, rating, age)

Boats(bid, bname, color)

Reserves(sid, bid, day)

End of Lecture-5

Also called a
logical query plan



$$\pi_{sname}((\sigma_{bid=103} Reserves) \bowtie Sailors)$$

Start of Lecture-6

Find sailors who've reserved a red or a green boat

Sailors(<u>sid</u>, sname, rating, age)
Boats(<u>bid</u>, bname, color)
Reserves(<u>sid, bid, day</u>)

Use of rename operation

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

$$\rho\ (Tempboats, (\sigma_{color='red' \vee color='green'}\ Boats))$$

$$\pi_{sname}(Tempboats \bowtie \mathrm{Re}serves \bowtie Sailors)$$

Can also define Tempboats using union
Try the "AND" version yourself

What about aggregates?

Sailors(<u>sid</u>, sname, rating, age)

Boats(<u>bid</u>, bname, color)

Reserves(<u>sid, bid, day</u>)

- 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 ∪ S has t m+n times
  - R ∩ S has t min(m, n) times
  - R – S has t max(0, m-n) times
  - sorting(τ), duplicate removal (δ) 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 $\{x \mid \exists y, z, w\ \text{Sailors}(x, w, y, z) \land w = \text{‘Bob’}\}$

Sailors(<u>sid</u>, sname, rating, age)

Output sid-s of sailors with name = ‘Bob’

TRC: example

Sailors(<u>sid</u>, sname, rating, age)

Boats(<u>bid</u>, bname, color)

Reserves(<u>sid, bid, day</u>)

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

∃	There exists

{P | ∃ S ϵ 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

$A \Rightarrow B$

- 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

$\exists$	There exists
$\forall$	For all
$\wedge$	Logical AND
$\vee$	Logical OR
$\neg$	NOT

- $\forall x\, P(x) \;=\; \neg\exists x\; [\neg P(x)]$

- $\neg(P \vee Q) \;=\; \neg P \wedge \neg Q$
- $\neg(P \wedge Q) \;=\; \neg P \vee \neg Q$
  - Similarly, $\neg(\neg P \vee Q) = P \wedge \neg Q$ etc.

$\left.\begin{matrix} \\ \\ \end{matrix}\right\}$ de Morgan's laws

- $A \Rightarrow B \;=\; \neg A \vee B$

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(<u>sid</u>, sname, rating, age)

Boats(<u>bid</u>, bname, color)

Reserves(<u>sid, bid, day</u>)

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

$\{P \mid \exists\, S \in Sailors\ (\exists\, R1 \in Reserves\ \exists\, R2 \in Reserves\ (S.sid = R1.sid \land S.sid = R2.sid \land R1.bid \neq R2.bid) \land P.sname = S.sname)\}$

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(<u>sid</u>, sname, rating, age)
Boats(<u>bid</u>, bname, color)
Reserves(<u>sid, bid, day</u>)

- Find the names of sailors who have reserved all boats

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

TRC: example

Sailors(<u>sid</u>, sname, rating, age)

Boats(<u>bid</u>, bname, color)

Reserves(<u>sid, bid, day</u>)

- Find the names of sailors who have reserved all <u>red</u> boats

How will you change the previous TRC expression?

TRC: example

Sailors(<u>sid</u>, sname, rating, age)
Boats(<u>bid</u>, bname, color)
Reserves(<u>sid, bid, day</u>)

- Find the names of sailors who have reserved all <u>red</u> boats

{P | ∃ S ϵ Sailors (∀B ϵ Boats (B.color = ‘red’ ⇒ (∃ R ϵ Reserves (S.sid = R.sid ⋀ R.bid = B.bid))) ⋀ P.sname = S.sname)}

Recall that A ⇒ B is logically equivalent to ¬ A ⋁ B
so ⇒ can be avoided, but it is cleaner and more intuitive

Feel free to use ¬ A ⋁ B

TRC & DRC: example

Sailors(<u>sid</u>, sname, rating, age)

Boats(<u>bid</u>, bname, color)

Reserves(<u>sid, bid, day</u>)

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

TRC:

{P | ∃ S ϵ Sailors (S.rating > 7 ⋀ P.name = S.name ⋀ P.age = S.age)}

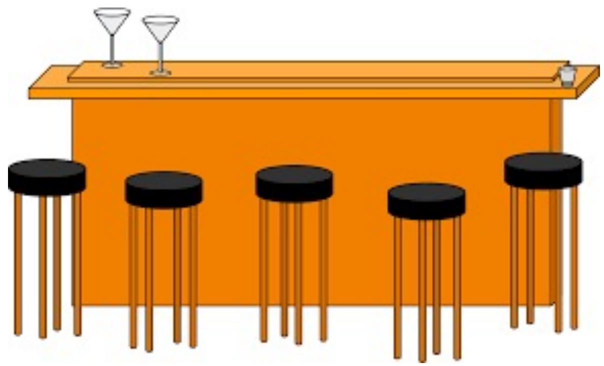
DRC:

{<N, A> | ∃ <I, N, T, A> ϵ Sailors ⋀ T > 7}

- Variables are now domain variables
- We will use use TRC
  - both are equivalent

The famous “Beers” database

Bars
Each has an address



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

Bars **Serve** Beers
At price “Y”

Drinkers **Likes** Beers

Drinkers
Each has an address

Beers
Each has a brewer

See online database for more tuples

"Beers" as a Relational Database

Bar

name	address
The Edge	108 Morris Street
Satisfaction	905 W. Main Street

Serves

bar	beer	price
The Edge	Budweiser	2.50
The Edge	Corona	3.00
Satisfaction	Budweiser	2.25

Beer

Name	brewer
Budweiser	Anheuser-Busch Inc.
Corona	Grupo Modelo
Dixie	Dixie Brewing

drinker	bar	times_a_week
Ben	Satisfaction	2
Dan	The Edge	1
Dan	Satisfaction	2

Frequents

Drinker

name	address
Amy	100 W. Main Street
Ben	101 W. Main Street
Dan	300 N. Duke Street

drinker	beer
Amy	Corona
Dan	Budweiser
Dan	Corona
Ben	Budweiser

Likes

More Examples: RC

UNDERSTAND THE DIFFERENCE IN ANSWERS
FOR ALL FOUR DRINKERS

Acknowledgement: examples and slides by Profs. Balazinska and Suciu, and the [GUW] book

Likes(drinker, beer)
Frequents(drinker, bar)
Serves(bar, beer)

Drinker Category 1

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

```
…
```

Likes(drinker, beer)
Frequents(drinker, bar)
Serves(bar, beer)

Drinker Category 1

Find drinkers that frequent <u>some</u> bar that serves <u>some</u> beer they like.

{x | ∃F ϵ Frequents (F.drinker = x.drinker ∧ ∃ S ϵ Serves ∃ L ϵ Likes
(F.drinker = L.drinker) ∧ (F.bar = S.bar) ∧ (S.beer =L.beer))}

Likes(drinker, beer)
Frequents(drinker, bar)
Serves(bar, beer)

Drinker Category 2

Find drinkers that frequent <u>some</u> bar that serves <u>some</u> beer they like.

{x | ∃F ϵ Frequents (F.drinker = x.drinker ∧ ∃ S ϵ Serves ∃ L ϵ Likes
(F.drinker = L.drinker) ∧ (F.bar = S.bar) ∧ (S.beer =L.beer))}

Find drinkers that frequent <u>only</u> bars that serves <u>some</u> beer they like.

…

Likes(drinker, beer)
Frequents(drinker, bar)
Serves(bar, beer)

Drinker Category 2

Find drinkers that frequent <u>some</u> bar that serves <u>some</u> beer they like.

{x | ∃F ϵ Frequents (F.drinker = x.drinker ∧ ∃ S ϵ Serves ∃ L ϵ Likes
(F.drinker = L.drinker) ∧ (F.bar = S.bar) ∧ (S.beer =L.beer))}

Find drinkers that frequent <u>only</u> bars that serve <u>some</u> beer they like.

{x | ∃F ϵ Frequents (F.drinker = x.drinker) ∧ [∀ F1 ϵ Frequents (F.drinker = F1.drinker)
⇒ ∃ S ϵ Serves ∃ L ϵ Likes [(F1.bar = S.bar) ∧ (F1.drinker = L.drinker) ∧ (S.beer =L.beer)]]}

Likes(drinker, beer)
Frequents(drinker, bar)
Serves(bar, beer)

Drinker Category 3

Find drinkers that frequent <u>some</u> bar that serves <u>some</u> 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 <u>only</u> bars that serve <u>some</u> 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 <u>some</u> bar that serves <u>only</u> beers they like.

…

Likes(drinker, beer)
Frequents(drinker, bar)
Serves(bar, beer)

Drinker Category 3

Find drinkers that frequent <u>some</u> bar that serves <u>some</u> beer they like.

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

Find drinkers that frequent <u>only</u> bars that serve <u>some</u> beer they like.

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

Find drinkers that frequent <u>some</u> bar that serves <u>only</u> beers they like.

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

Likes(drinker, beer)
Frequents(drinker, bar)
Serves(bar, beer)

Drinker Category 4

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

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

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

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

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

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

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

…

Likes(drinker, beer)
Frequents(drinker, bar)
Serves(bar, beer)

Drinker Category 4

Find drinkers that frequent <u>some</u> bar that serves <u>some</u> 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 <u>only</u> bars that serve <u>some</u> 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 <u>some</u> bar that serves <u>only</u> 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 <u>only</u> bars that serve <u>only</u> 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)

Likes(drinker, beer)
Frequents(drinker, bar)
Serves(bar, beer)

Drinker category 5!

From RC to SQL

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

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

Likes(drinker, beer)
Frequents(drinker, bar)
Serves(bar, beer)

From RC to SQL (or RA)

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

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

$\equiv\ \{x \mid \exists\, L \in \text{Likes}\ (L.\text{drinker} = x.\text{drinker}) \wedge [\ \forall\, S \in \text{Serves}\ [\neg\ (L.\text{beer} = S.\text{beer}) \vee [\exists\, F \in \text{Frequents}\ [(F.\text{drinker} = L.\text{drinker}) \wedge (F.\text{bar} = S.\text{bar})]\ ]\ ]\}$

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

$\forall x\ P(x)$ same as $\neg\exists x\ \neg P(x)$

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

$\neg(\neg P \vee Q)$ same as $P \wedge \neg\ Q$

SQL or RA does not have $\forall$!

Now you got all $\exists$ and $\neg$ expressible in RA/SQL

Likes(drinker, beer)
Frequents(drinker, bar)
Serves(bar, beer)

From RC to SQL

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

$\exists$ L $\in$ Likes $\wedge$ $\neg\exists$ S $\in$ Serves [(L.beer = S.beer) $\wedge$
$\neg$ [$\exists$ F $\in$ Frequents [(F.drinker = L.drinker) $\wedge$ (F.bar = S.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))
```

We will see a "methodical and correct" translation trough "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
  - $\{S \mid \neg (S \in Sailors)\}$ – 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