CompSci 516
Data Intensive Computing Systems

Lecture 4
Relational Algebra
and
Relational Calculus

Instructor: Sudeepa Roy

Announcement

- Deadlines extended by a week for project proposal and midterm report

- Proposal (writeup): 09/28 (Wed)
  - But send me an email(s) mentioning your group members and describing your project by 09/21
  - Project ideas to be posted by this weekend
- Midterm report: 10/28 (Fri)
- Final report: 11/28 (Mon)

Today's topics

- Finish SQL (from Lecture 2)
- 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

- 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)
- Note: Declarative (RC, SQL) vs. Operational (RA)

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
    - 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

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

sid	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

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

sname	rating
yuppy	9
rusty	10

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

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

$S1 \cup S2$

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

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 R1.
- Result schema has one field per field of S1 and R1, with field names `inherited' if possible.
  - Conflict: Both S1 and R1 have a field called sid.

(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

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

or

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

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>)

- Solution 1: $\pi_{sname}((\sigma_{bid=103} \text{Reserves}) \bowtie Sailors)$
- Solution 2: $\pi_{sname}(\sigma_{bid=103}(\text{Reserves} \bowtie Sailors))$

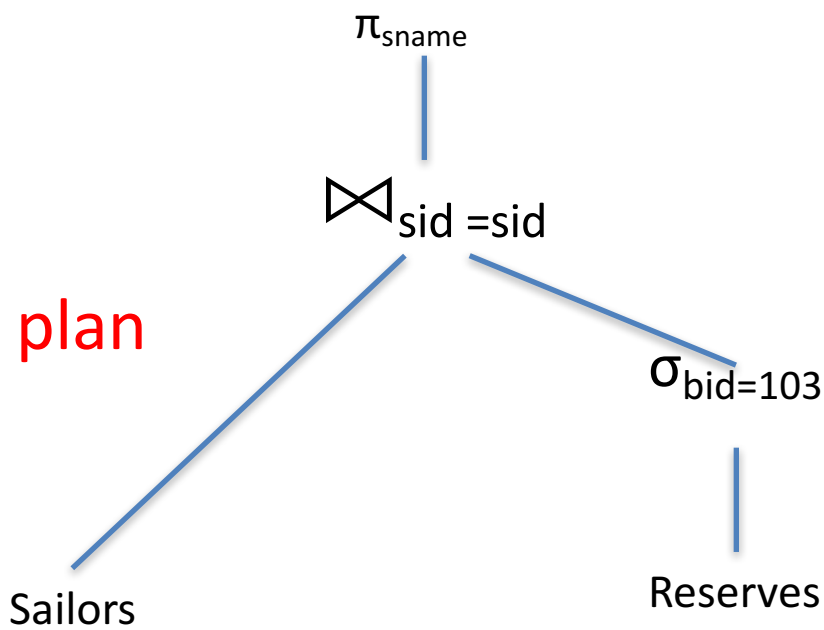
Expressing an RA expression as a Tree

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

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

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

Also called a
logical query plan



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

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:

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

$$\pi_{sname}(Tempboats \bowtie Reserves \bowtie Sailors)$$

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

- 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(<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

$\exists$ There exists

$\{P \mid \exists\ S \in Sailors\ (S.rating > 7 \wedge P.name = S.name \wedge P.age = S.age)\}$

- P is a tuple variable
  - with exactly two fields name and age (schema of the output relation)
  - P.name = S.name $\wedge$ P.age = S.age gives values to the fields of an answer tuple
- Use parentheses, $\forall$ $\exists$ $\vee$ $\wedge$ $>$ $<$ $=$ $\neq$ $\neg$ 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, $\neg A \vee B$, 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

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

- $\neg(P \vee Q) \quad = \quad \neg P \wedge \neg Q$
- $\neg(P \wedge Q) \quad = \quad \neg P \vee \neg Q$

de Morgan's laws

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

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

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

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

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

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
- Division operation in RA!

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

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(<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

$\{P \mid \exists\ S \in Sailors\ \forall B \in Boats\ (B.color = \text{'red'} \Rightarrow (\exists\ R \in Reserves\ (S.sid = R.sid \wedge R.bid = B.bid \wedge P.name = S.name)))\}$

Recall that $A \Rightarrow B$ is logically equivalent to $\neg A \vee B$

so $\Rightarrow$ can be avoided, but it is cleaner and more intuitive

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:
$\{P \mid \exists\ S \in Sailors\ (S.rating > 7 \wedge P.name = S.name \wedge P.age = S.age)\}$

DRC:
$\{\langle N, A\rangle \mid \exists\ \langle I, N, T, A\rangle \in Sailors \wedge T > 7\}$

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

More Examples: RC

- The famous "Drinker-Beer-Bar" example!

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 <u>some</u> bar that serves <u>some</u> 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.

$Q(x) = \exists y.\ \exists z.\ Frequents(x, y) \wedge Serves(y,z) \wedge Likes(x,z)$

a shortcut for
$\{x \mid \exists Y \in Frequents\ \ Z \in Serves\ W \in Likes\ (T.drinker = x.drinker \wedge T.bar = Z.bar \wedge W.beer = \ldots\ldots\}$

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

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.

$Q(x) = \exists y.\ \exists z.\ Frequents(x, y) \wedge Serves(y,z) \wedge Likes(x,z)$

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

$Q(x) = \ldots$

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.

$$Q(x) = \exists y.\ \exists z.\ Frequents(x, y) \wedge Serves(y,z) \wedge Likes(x,z)$$

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

$$Q(x) = \forall y.\ Frequents(x, y) \Rightarrow (\exists z.\ Serves(y,z) \wedge Likes(x,z))$$

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.

$Q(x) = \exists y.\ \exists z.\ Frequents(x, y) \wedge Serves(y,z) \wedge Likes(x,z)$

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

$Q(x) = \forall y.\ Frequents(x, y) \Rightarrow (\exists z.\ Serves(y,z) \wedge Likes(x,z))$

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

$Q(x) = \ldots$

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.

$$Q(x) = \exists y.\ \exists z.\ Frequents(x, y) \wedge Serves(y,z) \wedge Likes(x,z)$$

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

$$Q(x) = \forall y.\ Frequents(x, y) \Rightarrow (\exists z.\ Serves(y,z) \wedge Likes(x,z))$$

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

$$Q(x) = \exists y.\ Frequents(x, y) \wedge \forall z.(Serves(y,z) \Rightarrow Likes(x,z))$$

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.

$Q(x) = \exists y.\ \exists z.\ Frequents(x, y) \wedge Serves(y,z) \wedge Likes(x,z)$

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

$Q(x) = \forall y.\ Frequents(x, y) \Rightarrow (\exists z.\ Serves(y,z) \wedge Likes(x,z))$

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

$Q(x) = \exists y.\ Frequents(x, y) \wedge \forall z.(Serves(y,z) \Rightarrow Likes(x,z))$

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

$Q(x) = \ldots$

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.

$$Q(x) = \exists y.\ \exists z.\ Frequents(x, y) \wedge Serves(y,z) \wedge Likes(x,z)$$

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

$$Q(x) = \forall y.\ Frequents(x, y) \Rightarrow (\exists z.\ Serves(y,z) \wedge Likes(x,z))$$

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

$$Q(x) = \exists y.\ Frequents(x, y) \wedge \forall z.(Serves(y,z) \Rightarrow Likes(x,z))$$

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

$$Q(x) = \forall y.\ Frequents(x, y) \Rightarrow \forall z.(Serves(y,z) \Rightarrow Likes(x,z))$$

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

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

$$Q(x) = \exists y.\ Likes(x, y) \wedge \forall z.(Serves(z,y) \Rightarrow Frequents(x,z))$$

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

$$Q(x) = \exists y.\ Likes(x, y) \wedge \forall z.(Serves(z,y) \Rightarrow Frequents(x,z))$$

$$\equiv \quad Q(x) = \exists y.\ Likes(x, y) \wedge \forall z.(\neg\ Serves(z,y) \vee Frequents(x,z))$$

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

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

$$Q(x) = \exists y.\ Likes(x, y) \wedge \neg\exists z.(Serves(z,y) \wedge \neg Frequents(x,z))$$

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

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

From RC to SQL

$Q(x) = \exists y.\ Likes(x, y) \wedge \neg \exists z.(Serves(z,y) \wedge \neg Frequents(x,z))$

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))
```

this slide can be skipped until we do Datalog
will revisit "Safe queries" then

The "correct" intermediate steps

- Write the query in RC
- If you have a variable under "negation", also add the "domain", i.e. where the variable appears without a negation
  - e.g. if you have $\neg H(x, y)$ for a subquery,
  - where x and y can only come from a relation R(x, y)
  - make it $R(x, y) \wedge \neg H(x, y)$

  - This is to make the query "safe" and "domain independent" – we will discuss this when we do Datalog
  - Intuitively, if you are trying to find "sailors that do not satisfy some criteria" you have to specify the domain of sailors, say from the sailor table, otherwise you are looking at an infinite space

this slide can be skipped until we do Datalog
will revisit "Safe queries" then

The "correct" intermediate step

Make all *subqueries* with negation domain independent
i.e. say where x is coming from

$$Q(x) = \exists y.\ Likes(x, y) \wedge \neg\exists z.(Likes(x,y) \wedge Serves(z,y) \wedge \neg Frequents(x,z))$$

```
SELECT DISTINCT L.drinker FROM Likes L
WHERE not exists
  (SELECT * FROM Likes L2, Serves S
   WHERE L2.drinker=L.drinker and L2.beer=L.beer
         and L2.beer=S.beer
         and not exists (SELECT * FROM Frequents F
                         WHERE F.drinker=L2.drinker
                             and F.bar=S.bar))
```

which can be simplified
to the previous query

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 | ¬ (S ϵ Sailors)} – infinitely many tuples – an "unsafe" query
  - More when we do "Datalog", also see Ch. 4.4 in [RG]
- Next topic: DBMS internals
  - storage/indexing, query execution, algorithms, optimization

Additional Slides
Division in RA

Division

- Not supported as a primitive operator, but useful for expressing queries like:

  Find sailors who have reserved **all** boats.

- Let A have 2 fields, x and y; B have only field y:
  - A/B = {<x> : $\forall$ <y> ϵ B $\exists$ <x, y> ϵ A}
  - i.e., **A/B contains all x tuples (sailors) such that for every y tuple (boat) in B, there is an xy tuple in A.**
  - Equivalently,
  - If the set of y values (boats) associated with an x value (sailor) in A contains all y values in B, the x value is in A/B.

Examples of Division A/B

sno	pno
s1	p1
s1	p2
s1	p3
s1	p4
s2	p1
s2	p2
s3	p2
s4	p2
s4	p4

A

pno
p2

B1

pno
p2
p4

B2

pno
p1
p2
p4

B3

sno
s1
s2
s3
s4

A/B1

sno
s1
s4

A/B2

sno
s1

A/B3

Expressing A/B Using Basic Operators

- Division is not essential op; just a useful shorthand.
  - Also true of joins = selection on cross products
  - but joins are so common that systems implement joins specially
- Idea: For A/B, compute all x values that are "not disqualified" by some y value in B
  - x value is disqualified if by attaching y value from B, we obtain an xy tuple that is not in A
- For A(x, y) and B(y):
  - see example first on the next slide

A/B: $$\pi_x(A) - \pi_x((\pi_x(A) \times B) - A)$$

Example: A/B illustration

sno	pno
s1	p1
s1	p2
s1	p3
s1	p4
s2	p1
s2	p2
s3	p2
s4	p2
s4	p4

A

pno
p2
p4

$B2$

sno
s1
s4

$A/B2$

- What is $\pi_x((\pi_x(A) \times B) - A)$
  - First cross product:
    - {s1, s2, s3, s4} × {p2, p4}
  - Then missing combinations in A:
    - {<s2, p4>, <s3, p4>}
  - Then "disqualified" A tuples
    - {s2, s3}
- What is

$$\pi_x(A) - \pi_x((\pi_x(A) \times B) - A)$$

  - The "qualified" A tuples
    - {s1, s2, s3, s4} – {s2, s3}

      = {s1, s4}

Find the names of sailors who've reserved all boats

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

- Uses division; schemas of the input relations to / must be carefully chosen:

$$\rho\ (Tempsids, (\pi_{sid,bid} Reserves) / (\pi_{bid} Boats))$$

$$\pi_{sname}(Tempsids \bowtie Sailors)$$

- To find sailors who've reserved all 'Interlake' boats:

$$\ldots\ldots \quad / \ \pi_{bid}(\sigma_{bname='Interlake'} Boats)$$