CompSci 516
Database Systems
(Incomplete Notes)
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
  - if not on piazza, send me an email

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

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

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)

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

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

Selection

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

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

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

Composition of Operators

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

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

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

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

$$S1 \cup S2$$

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

$$S1 \cup S2$$

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

$S1 - S2$ $S1 \cap S2$

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

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

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

Joins

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

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

- 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

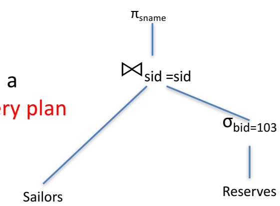
Sailors(<u>sid</u>, sname, rating, age)
Boats(<u>bid</u>, bname, color)
Reserves(<u>sid, bid, day</u>)

- Solution 1:
- Solution 2:

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:

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
- $\gamma_{age,\ avg(rating) \rightarrow avgr}$ 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

∃ There exists

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

- 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, $\neg A \vee B$, i.e.
  - either A is false (then B can be anything)
  - otherwise (i.e. A is true) B must be true

Useful Logical Equivalences

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

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

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

  } 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(<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 <u>at least two boats</u>

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

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
- Called the "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!

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

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 | $\exists$ S ϵ Sailors (S.rating > 7 $\wedge$ P.name = S.name $\wedge$ P.age = S.age)}

DRC:
{<N, A> | $\exists$ <I, N, T, A> ϵ Sailors $\wedge$ 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

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 | $\exists$Y ϵ Frequents Z ϵ Serves W ϵ Likes (T.drinker = x.drinker $\wedge$ T.bar = Z.bar $\wedge$ W.beer = ……}

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) = ∃y. ∃z. Frequents(x, y)∧Serves(y,z)∧Likes(x,z)

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

Q(x) = …

Duke CS, Fall 2018 CompSci 516: Database Systems 43

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) = ∃y. ∃z. Frequents(x, y)∧Serves(y,z)∧Likes(x,z)

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

Duke CS, Fall 2018 CompSci 516: Database Systems 44

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) = ∃y. ∃z. Frequents(x, y)∧Serves(y,z)∧Likes(x,z)

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

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

Duke CS, Fall 2018 CompSci 516: Database Systems 45

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) = ∃y. ∃z. Frequents(x, y)∧Serves(y,z)∧Likes(x,z)

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

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

Duke CS, Fall 2018 CompSci 516: Database Systems 46

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) = ∃y. ∃z. Frequents(x, y)∧Serves(y,z)∧Likes(x,z)

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

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

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

Duke CS, Fall 2018 CompSci 516: Database Systems 47

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) = ∃y. ∃z. Frequents(x, y)∧Serves(y,z)∧Likes(x,z)

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

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

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

Duke CS, Fall 2018 CompSci 516: Database Systems 48

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 ∀ and ⇒
  - 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 ∀ with ∃ using de Morgan's Laws

∀x P(x) same as ¬∃x ¬P(x)

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

¬(¬P ∨ Q) same as P ∧ ¬ Q

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

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

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))
```

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 | ¬ (S ϵ Sailors)} – infinitely many tuples – an "unsafe" query
  - More when we do "Datalog", also see Ch. 4.4 in [RG]