CompSci 516
Database Systems

Lecture 7
Relational Calculus (revisit)
And
Normal Forms

Instructor: Sudeepa Roy

Announcements

- HW1 Deadlines!
  - Today: parser and Q1-Q3
  - Q4: next Tuesday
  - Q5 (3 RA questions will be posted today): next Thursday

- 2 late days with penalty apply for individual deadlines
  - If you are still parsing XML
    - Remember to start early next time from first day
    - HW2 and HW3 typically take more time and effort!

Today's topic

- Revisit RC
- Finish Normalization

- From Thursday: Database Internals

Acknowledgement:
The following slides have been created adapting the instructor material of the [RG] book provided by the authors Dr. Ramakrishnan and Dr. Gehrke, and with the help of slides by Dr. Magda Balazinska and Dr. Dan Suciu

Relational Calculus (RC)
(Revisit from Lecture 4)

The RC in this lecture is called Tuple Relational Calculus (TRC).
There is an equivalent form called Domain Relational Calculus (DRC) that we are not considering

Logic Notations

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

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

Useful Logical Equivalences

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

$\exists$	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.

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

RC: 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

RC: 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 | ∃ S ∈ Sailors (∃ R1 ∈ Reserves ∃ R2 ∈ Reserves (S.sid = R1.sid ∧ S.sid = R2.sid ∧ R1.bid ≠ R2.bid) ∧ P.sname = S.sname)}

RC: example

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

- Find the names of sailors who have reserved all boats

RC: 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 | ∃ S ∈ Sailors [∀B ∈ Boats (∃ R ∈ Reserves (S.sid = R.sid ∧ R.bid = B.bid))] ∧ (P.sname = S.sname)}

RC: 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?

RC: 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 | ∃ S ∈ Sailors (∀B ∈ Boats (B.color = 'red' ⇒ (∃ R ∈ 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

More Examples: RC

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

UNDERSTAND THE DIFFERENCE IN ANSWERS
FOR ALL FOUR DRINKERS

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 some bar that serves some beer they like.

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

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

Drinker Category 2

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

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

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

…

Free HW question hint!

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

Drinker Category 2

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

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

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

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

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

Drinker Category 3

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

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

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.

…

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

Drinker Category 3

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

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

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.

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

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

Drinker Category 4

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

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

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

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

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

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

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 some bar that serves some beer they like.

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

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

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

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

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

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

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

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 or RA query

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

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

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

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

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

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

$\neg(\neg P \vee Q)$ 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

Database Normalization

Recap from Lecture-5

ssn (S)	name (N)	lot (L)	rating (R)	hourly-wage (W)	hours-worked (H)
111-11-1111	Attishoo	48	8	10	40
222-22-2222	Smiley	22	8	10	30
333-33-3333	Smethurst	35	5	7	30
444-44-4444	Guldu	35	5	7	32
555-55-5555	Madayan	35	8	10	40

Redundancy is bad!
(well…not always?)

1. Redundant storage
2. Update anomalies
3. Insertion anomalies
4. Deletion anomalies

Solution: Decomposition!

Be careful about "Lossy decomposition"!
(on blackboard)

Schema is forcing to store (complex) associations among tuples

Nulls may or may not help

Decompositions should be used judiciously

1. Do we need to decompose a relation?
   - Several "normal forms" exist to identify possible redundancy at different granularity
   - If a relation is not in one of them, may need to decompose further

2. What are the problems with decomposition?
   - Bad decompositions: e.g., Lossy decompositions
   - Performance issues -- decomposition may both
     - help performance (for updates, some queries accessing part of data), or
     - hurt performance (new joins may be needed for some queries)

Functional Dependencies (FDs)

- A functional dependency (FD) X → Y holds over relation R if, for every allowable instance *r* of R:
  - i.e., given two tuples in *r*, if the X values agree, then the Y values must also agree
  - X and Y are *sets* of attributes
  - $t1 \in r,\ t2 \in r,\ \Pi_X(t1) = \Pi_X(t2)$ implies $\Pi_Y(t1) = \Pi_Y(t2)$

A	B	C	D
a1	b1	c1	d1
a1	b1	c1	d2
a1	b2	c2	d1
a2	b1	c3	d1

What is a (possible) FD here?

Functional Dependencies (FDs)

- A functional dependency (FD) X → Y holds over relation R if, for every allowable instance *r* of R:
  - i.e., given two tuples in *r*, if the X values agree, then the Y values must also agree
  - X and Y are *sets* of attributes
  - $t1 \in r,\ t2 \in r,\ \Pi_X(t1) = \Pi_X(t2)$ implies $\Pi_Y(t1) = \Pi_Y(t2)$

A	B	C	D
a1	b1	c1	d1
a1	b1	c1	d2
a1	b2	c2	d1
a2	b1	c3	d1

What is a (possible) FD here?

AB → C

Note that, AB is not a key

Can we detect FDs from an instance?

- An FD is a statement about all allowable relation instances
  - Must be identified based on semantics of application
  - Given some allowable instance *r1* of R, we can check if it violates some FD *f*, but we cannot tell if *f* holds over R

FD from a key

- Consider a relation R(A,B, C, D) where AB is a key
- Which FD must hold on R?

- AB → ABCD

- However, S → ABCD does not mean S is a key. Why?
  - S can be a superkey!
  - E.g., ABC → ABCD in R, but ABC is not a key

Armstrong's Axioms

- X, Y, Z are sets of attributes

1. Reflexivity: If $X \supseteq Y$, then $X \rightarrow Y$, e.g., $ABC \rightarrow AB$
2. Augmentation: If $X \rightarrow Y$, then $XZ \rightarrow YZ$ for any Z,
   - e.g., $AB \rightarrow C \Rightarrow ABDE \rightarrow CDE$
3. Transitivity: If $X \rightarrow Y$ and $Y \rightarrow Z$, then $X \rightarrow Z$
   - e.g., $AB \rightarrow C$ and $C \rightarrow D \Rightarrow AB \rightarrow D$

A	B	C	D
a1	b1	c1	d1
a1	b1	c1	d2
a1	b2	c2	d1
a2	b1	c3	d1

Apply these rules on $AB \rightarrow C$ and check

- Additional rules that follow from Armstrong's Axioms

4. Union: If $X \rightarrow Y$ and $X \rightarrow Z$, then $X \rightarrow YZ$
5. Decomposition: If $X \rightarrow YZ$, then $X \rightarrow Y$ and $X \rightarrow Z$

$A \rightarrow B$ and $A \rightarrow C$
$\Rightarrow A \rightarrow BC$

$A \rightarrow BC$
$\Rightarrow A \rightarrow B, A \rightarrow C$

Closure of a set of FDs

- Given some FDs, we can usually infer additional FDs:
  - SSN → DEPT, and DEPT → LOT implies SSN → LOT
- An FD f is implied by a set of FDs F if f holds whenever all FDs in F hold.
- F^+ = closure of FDs F is the set of all FDs that are implied by F
- S^+ = closure of attributes S is the set of all attributes that are implied by S according to F^+

Armstrong's Axioms are sound and complete inference rules for FDs

- sound: they only generate FDs in closure F^+ for F
- complete: by repeated application of these rules, all FDs in F^+ will be generated

Computing Attribute Closure

Algorithm:

- closure = X
- Repeat until no change
  - if there is an FD U → V in F such that U ⊆ closure, then closure = closure ∪ V

Let's do the example first, Then look at the algo yourself

Does F = {A → B, B → C, C D → E } imply

1. A → E? (i.e, is A → E in the closure F^+ , or E in A^+?)
2. AD → E?

On blackboard

Normal Forms

- Question: given a schema, how to decide whether any schema refinement is needed at all?
- If a relation is in a certain normal forms, it is known that certain kinds of problems are avoided/minimized
- Helps us decide whether decomposing the relation is something we want to do

FDs play a role in detecting redundancy

Example

- Consider a relation R with 3 attributes, ABC
  - No FDs hold: There is no redundancy here – no decomposition needed
  - Given A → B: Several tuples could have the same A value, and if so, they'll all have the same B value ⇒ redundancy ⇒ decomposition may be needed if A is not a key

- Intuitive idea:
  - if there is any non-key dependency, e.g. A → B, decompose!