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)
Logic Notations

• ∀  For all
• ∃  There exists
• ∧  Logical AND
• ∨  Logical OR
• ¬   NOT
• ⇒  Implies
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 ⇒ B

- A “implies” B
- Equivalently, if A is true, B must be true
- Equivalently, ¬A ∨ 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.

• $A \implies B = \neg A \lor B$
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(sid, sname, rating, age)
Boats(bid, bname, 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)\}
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(sid, sname, rating, age)
Boats(bid, bname, color)
Reserves(sid, bid, day)

• Find the names of sailors who have reserved all boats

\{\text{P} \mid \exists S \in \text{Sailors} [\forall B \in \text{Boats} (\exists R \in \text{Reserves} (S.\text{sid} = R.\text{sid} \land R.\text{bid} = B.\text{bid}))] \land (\text{P.sname} = S.\text{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 \mid \exists S \in \text{Sailors} (\forall B \in \text{Boats} (B.\text{color} = \text{‘red’} \Rightarrow (\exists R \in \text{Reserves} (S.\text{sid} = R.\text{sid} \land R.\text{bid} = B.\text{bid})))) \land P.\text{sname} = S.\text{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
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.

…
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}) \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})\}
Drinker Category 2

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

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

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

Free HW question hint!
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}) \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 only bars that serve some 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})] ]\}\}
Drinker Category 3

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

\[ \{ x | \exists F \in \text{Frequents} \ (F.\text{drinker} = x.\text{drinker}) \land \exists S \in \text{Serves} \ \exists L \in \text{Likes} \ (F.\text{drinker} = L.\text{drinker}) \land (F.\text{bar} = S.\text{bar}) \land (S.\text{beer} = L.\text{beer}) \} \]

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

\[ \{ x | \exists F \in \text{Frequents} \ (F.\text{drinker} = x.\text{drinker}) \land [ \forall F1 \in \text{Frequents} \ (F.\text{drinker} = F1.\text{drinker}) \implies \exists S \in \text{Serves} \ \exists L \in \text{Likes} \ [(F1.\text{bar} = S.\text{bar}) \land (F1.\text{drinker} = L.\text{drinker}) \land (S.\text{beer} = L.\text{beer})] ] \} \]

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

\[ \ldots \]
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} \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 only bars that serve some 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 some bar that serves only 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 some bar that serves some beer they like.

\[ \{ x | \exists F \in \text{Frequents} \ (F.drinker = x.drinker) \land \exists S \in \text{Serves} \exists L \in \text{Likes} \]
\[ \quad \quad \quad (F.drinker = L.drinker) \land (F.bar = S.bar) \land (S.beer = L.beer) \}\]

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

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

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

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

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

...
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}) \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 only bars that serve some 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 some bar that serves only 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 only bars that serve only 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 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
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}) \land [ \forall S \in \text{Serves} \ (L.\text{beer} = S.\text{beer}) \Rightarrow \\
\exists F \in \text{Frequents} \ [(F.\text{drinker} = L.\text{drinker}) \land (F.\text{bar} = S.\text{bar})] \} \}
From RC to SQL (or RA)

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

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

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

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

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

SQL or RA does not have \( \forall \)!
Now you got all \( \exists \) and \( \neg \) expressible in RA/SQL
From RC to SQL

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

$$\exists L \in \text{Likes} \land \neg \exists S \in \text{Serves} [((L.\text{beer} = S.\text{beer}) \land$$
$$\neg [\exists F \in \text{Frequents} [((F.\text{drinker} = L.\text{drinker}) \land (F.\text{bar} = S.\text{bar})]])$$

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 through “safe queries” in Datalog.
Database Normalization
Recap from Lecture-5

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

Redundancy is bad! (well...not always?)

<table>
<thead>
<tr>
<th>ssn (S)</th>
<th>name (N)</th>
<th>lot (L)</th>
<th>rating (R)</th>
<th>hourly-wage (W)</th>
<th>hours-worked (H)</th>
</tr>
</thead>
<tbody>
<tr>
<td>111-11-1111</td>
<td>Attishoo</td>
<td>48</td>
<td>8</td>
<td>10</td>
<td>40</td>
</tr>
<tr>
<td>222-22-2222</td>
<td>Smiley</td>
<td>22</td>
<td>8</td>
<td>10</td>
<td>30</td>
</tr>
<tr>
<td>333-33-3333</td>
<td>Smethurst</td>
<td>35</td>
<td>5</td>
<td>7</td>
<td>30</td>
</tr>
<tr>
<td>444-44-4444</td>
<td>Guldu</td>
<td>35</td>
<td>5</td>
<td>7</td>
<td>32</td>
</tr>
<tr>
<td>555-55-5555</td>
<td>Madayan</td>
<td>35</td>
<td>8</td>
<td>10</td>
<td>40</td>
</tr>
</tbody>
</table>
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) $X \rightarrow Y$ 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
  - $t_1 \in r$, $t_2 \in r$, $\Pi_X(t_1) = \Pi_X(t_2)$ implies $\Pi_Y(t_1) = \Pi_Y(t_2)$

What is a (possible) FD here?

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>a1</td>
<td>b1</td>
<td>c1</td>
<td>d1</td>
</tr>
<tr>
<td>a1</td>
<td>b1</td>
<td>c1</td>
<td>d2</td>
</tr>
<tr>
<td>a1</td>
<td>b2</td>
<td>c2</td>
<td>d1</td>
</tr>
<tr>
<td>a2</td>
<td>b1</td>
<td>c3</td>
<td>d1</td>
</tr>
</tbody>
</table>
Functional Dependencies (FDs)

- A functional dependency (FD) \( X \rightarrow Y \) 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
  - \( t_1 \in r, \ t_2 \in r, \ \Pi_X(t_1) = \Pi_X(t_2) \) implies \( \Pi_Y(t_1) = \Pi_Y(t_2) \)

What is a (possible) FD here?

\[
\begin{array}{cccc}
A & B & C & D \\
\hline
a1 & b1 & c1 & d1 \\
a1 & b1 & c1 & d2 \\
a1 & b2 & c2 & d1 \\
a2 & b1 & c3 & d1 \\
\end{array}
\]

\( AB \rightarrow C \)

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 ⊇ Y, then X → Y, e.g., ABC → AB
2. Augmentation: If X → Y, then XZ → YZ for any Z,
   – e.g., AB → C ⇒ ABDE → CDE
3. Transitivity: If X → Y and Y → Z, then X → Z
   – e.g., AB → C and C → D ⇒ AB → D

• Additional rules that follow from Armstrong’s Axioms

4. Union: If X → Y and X → Z, then X → YZ
5. Decomposition: If X → YZ, then X → Y and X → Z

A → B and A → C
⇒ A → BC
A → BC
⇒ A → B, A → C

Apply these rules on AB → C and check
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 U V

Does $F = \{ A \rightarrow B, B \rightarrow C, C D \rightarrow E \}$ imply
1. $A \rightarrow E$? (i.e., is $A \rightarrow E$ in the closure $F^+$, or $E$ in $A^+$?)
2. $AD \rightarrow E$?

On blackboard

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