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

Relational Calculus (RC)
(Revisit from Lecture 4)

Logic Notations
• $\exists$ There exists
• $\forall$ For all
• $\land$ Logical AND
• $\lor$ Logical OR
• $\neg$ NOT
• $\Rightarrow$ Implies

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.

TRC: example

$\exists$ There exists

$\exists$ S $\in$ Sailors $(S.rating > 7 \land P.sname = S.sname \land P.age = S.age)$

• Find the name and age of all sailors with a rating above 7
• P is a tuple variable
  – with exactly two fields sname and age (schema of the output relation)
  – P.sname = S.sname \land P.age = S.age gives values to the fields of an answer tuple
• Use parentheses, $\exists$ $\forall$ $\land$ $\lor$ $\neg$ $<$ $=$ $\rightarrow$ etc as necessary
• A $\Rightarrow$ B is very useful too
A ⇒ B

• A "implies" B
• Equivalently, if A is true, B must be true
• Equivalently, ¬ A V 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

• ∀ x P(x) = ¬∃ x [¬P(x)]
• ¬(P \lor Q) = ¬ P \land ¬ Q
• ¬(P \land Q) = ¬ P \lor ¬ Q
  – Similarly, ¬(¬P \lor Q) = P \land ¬ Q etc.

• A ⇒ B = ¬ A \lor 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(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(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(sid, sname, rating, age)
Boats(bid, bname, color)
Reserves(sid, bid, day)

• Find the names of sailors who have reserved all boats
TRC: example

- Find the names of sailors who have reserved all red boats

How will you change the previous TRC expression?

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.

Drinker Category 2

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.

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

Drinker Category 3

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

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

Drinker Category 4

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

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

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 \( \lor \) and \( \Rightarrow \)
  - then you can systematically go to a "correct" SQL or RA query
Drinker category S!

From RC to SQL

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

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

Step 2: Translate into SQL

```sql
SELECT DISTINCT L.drinker
FROM Likes L
WHERE not exists
    (SELECT * 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

<table>
<thead>
<tr>
<th>Roll</th>
<th>Name</th>
<th>ID</th>
<th>Age</th>
<th>Hours</th>
<th>Wage</th>
</tr>
</thead>
<tbody>
<tr>
<td>113</td>
<td>Wilkinson</td>
<td>2002</td>
<td>20</td>
<td>30</td>
<td>500</td>
</tr>
<tr>
<td>222</td>
<td>Singer</td>
<td>2003</td>
<td>21</td>
<td>35</td>
<td>600</td>
</tr>
<tr>
<td>333</td>
<td>Stewart</td>
<td>2004</td>
<td>22</td>
<td>40</td>
<td>700</td>
</tr>
<tr>
<td>444</td>
<td>Smith</td>
<td>2005</td>
<td>23</td>
<td>45</td>
<td>800</td>
</tr>
</tbody>
</table>

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

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

Be careful about "Lossy decomposition"!

On blackboard

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)

- Deletions and nulls may or may not help
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 \( \Pi_Y(t_1) = \Pi_Y(t_2) \)

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 \( r \) of \( R \), we can check if it violates some FD \( f \), but we cannot tell if \( f \) holds over \( R \)

- \( K \) is a candidate key for \( R \) means that \( K \rightarrow R \)
  - denoting \( R = \) all attributes of \( R \)
  - However, \( S \rightarrow R \) does not require \( S \) to be minimal
  - e.g. \( S \) can be a superkey

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 \( C \rightarrow D \) \( \Rightarrow AB \rightarrow D \)

- 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 \( X \rightarrow Z \)

*Armstrong’s Axioms are sound and complete inference rules for FDs*

FD from a key

- Consider a relation \( R(AB, C, D) \) where \( AB \) is a key
- Which FD must hold on \( R \)?
  - \( AB \rightarrow ABCD \)
- However, \( S \rightarrow ABCD \) does not mean \( S \) is a key. Why?
  - \( S \) can be a superkey!
  - E.g., \( ABC \rightarrow ABCD \) in \( R \), but \( ABC \) is not a key

Closure of a set of FDs

- Given some FDs, we can usually infer additional FDs:
  - \( SSN \rightarrow DEPT \), and \( DEPT \rightarrow LOT \) implies \( SSN \rightarrow LOT \)

- An FD \( f \) is implied by a set of FDs \( F \) if \( f \) holds whenever all FDs in \( F \) hold.

- \( F^+ = \) closure of \( F \) 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

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 \( r \) of \( R \), we can check if it violates some FD \( f \), but we cannot tell if \( f \) holds over \( R \)

- \( K \) is a candidate key for \( R \) means that \( K \rightarrow R \)
  - denoting \( R = \) all attributes of \( R \)
  - However, \( S \rightarrow R \) does not require \( S \) to be minimal
  - e.g. \( S \) can be a superkey
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 → 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?

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

Normal Forms

R is in 4NF
⇒ R is in BCNF
⇒ R is in 3NF
⇒ R is in 2NF (a historical one)
⇒ R is in 1NF (every field has atomic values)

Only BCNF and 4NF are covered in the class

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!

Boyce-Codd Normal Form (BCNF)

• Relation R with FDs F is in BCNF if, for all X → A in F
  – A ∈ X (called a trivial FD), or
  – X contains a key for R
    • i.e. X is a superkey

BCNF decomposition algorithm

• Find a BCNF violation
  – That is, a non-trivial FD X → Y in R where X is not a super key of R
• Decompose R into R₁ and R₂, where
  – R₁ has attributes X U Y
  – R₂ has attributes X U Z, where Z contains all attributes of R that are in neither X nor Y
• Repeat until all relations are in BCNF
• Also gives a lossless decomposition!
  – Check yourself
BCNF decomposition example - 1

- CSJDQV, key C, F = {JP → C, SD → P, J → S}
  - To deal with SD → P, decompose into SDP, CSJDQV.
  - To deal with J → S, decompose CSJDQV into JS and CJDQV.

- Is JP → C a violation of BCNF?

- Note:
  - several dependencies may cause violation of BCNF
  - The order in which we pick them may lead to very different sets of relations
  - there may be multiple correct decompositions (can pick J → S first)

BCNF decomposition example - 2

UserJoinsGroup (uid, uname, twitterid, gid, fromDate)

BCNF violation: uid → twitterid, twitterid → uid

User (uid, uname, twitterid)

Member (uid, gid, fromDate)

BCNF

Recap

- Functional dependencies: a generalization of the key concept
- Non-key functional dependencies: a source of redundancy
- BCNF decomposition: a method for removing redundancies
  - BCNF decomposition is a lossless join decomposition
  - BCNF: schema in this normal form has no redundancy due to FD’s

BCNF = no redundancy?

- User (uid, gid, place)
  - A user can belong to multiple groups
  - A user can register places she’s visited
  - Groups and places have nothing to do with other
  - FD’s?
    - None
    - BCNF?
      - Yes
      - Redundancies?
        - Tons!

Multivalued dependencies

- A multivalued dependency (MVD) has the form X → Y, where X and Y are sets of attributes in a relation R

  \[
  \begin{array}{ccc}
  X & Y & Z \\
  a & b_1 & c_1 \\
  a & b_2 & c_2 \\
  a & b_1 & c_3 \\
  \vdots & \vdots & \vdots \\
  \end{array}
  \]

  \[ X \rightarrow Y \]
  means that whenever two rows in R agree on all the attributes of X, then we can swap their Y components and get two rows that are also in R
MVD examples

User (uid, gid, place)
• uid ↞ gid
• uid ↞ place
  – Intuition: given uid, attributes gid and place are “independent”
• uid, gid ↞ place
  – Trivial: LHS U RHS = all attributes of R
• uid, gid ↞ uid
  – Trivial: LHS ⊇ RHS

An elegant solution: “chase”

• Given a set of FD’s and MVD’s 𝒫, does another dependency 𝑑 (FD or MVD) follow from 𝒫?

• Procedure
  – Start with the premise of 𝑑, and treat them as “seed” tuples in a relation
  – Apply the given dependencies in 𝒫 repeatedly
    • If we apply an FD, we infer equality of two symbols
    • If we apply an MVD, we infer more tuples
  – If we infer the conclusion of 𝑑, we have a proof
  – Otherwise, if nothing more can be inferred, we have a counterexample

4NF

• A relation 𝑅 is in Fourth Normal Form (4NF) if
  – For every non-trivial MVD 𝑋 ↞ 𝑌 in 𝑅, 𝑋 is a superkey
  – That is, all FD’s and MVD’s follow from “key ↞ other attributes” (i.e., no MVD’s and no FD’s besides key functional dependencies)

• 4NF is stronger than BCNF
  – Because every FD is also a MVD

Proof by chase

• In 𝑅(𝐴, 𝐵, 𝐶, 𝐷), does 𝐴 → 𝐵 and 𝐵 → 𝐶 imply that 𝐴 → 𝐶?

Have: 𝐴 𝐵 𝐶 𝐷

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b₁</td>
<td>c₁</td>
<td>d₁</td>
</tr>
<tr>
<td>a</td>
<td>b₂</td>
<td>c₂</td>
<td>d₂</td>
</tr>
</tbody>
</table>

Need: 𝐴 𝐵 𝐶 𝐷

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b₁</td>
<td>c₁</td>
<td>d₁</td>
</tr>
<tr>
<td>a</td>
<td>b₂</td>
<td>c₂</td>
<td>d₂</td>
</tr>
</tbody>
</table>

Another proof by chase

• In 𝑅(𝐴, 𝐵, 𝐶, 𝐷), does 𝐴 → 𝐵 and 𝐵 → 𝐶 imply that 𝐴 → 𝐶?

Have: 𝐴 𝐵 𝐶 𝐷

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b₁</td>
<td>c₁</td>
<td>d₁</td>
</tr>
<tr>
<td>a</td>
<td>b₂</td>
<td>c₂</td>
<td>d₂</td>
</tr>
</tbody>
</table>

Need: 𝑐₁ = 𝑐₂

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b₁</td>
<td>c₁</td>
<td>d₁</td>
</tr>
<tr>
<td>a</td>
<td>b₂</td>
<td>c₂</td>
<td>d₂</td>
</tr>
</tbody>
</table>

In general, with both MVD’s and FD’s, chase can generate both new tuples and new equalities

Counterexample by chase

• In 𝑅(𝐴, 𝐵, 𝐶, 𝐷), does 𝐴 ↞ 𝐵𝐶 and 𝐶𝐷 ↞ 𝐵 imply that 𝐴 → 𝐵?

Have: 𝐴 𝐵 𝐶 𝐷

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b₁</td>
<td>c₁</td>
<td>d₁</td>
</tr>
<tr>
<td>a</td>
<td>b₂</td>
<td>c₂</td>
<td>d₂</td>
</tr>
</tbody>
</table>

Need: 𝑏₁ = 𝑏₂

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b₁</td>
</tr>
<tr>
<td>a</td>
<td>b₂</td>
</tr>
</tbody>
</table>

Counterexample!
4NF decomposition algorithm

- Find a 4NF violation
  - A non-trivial MVD $X \rightarrow Y$ in $R$ where $X$ is not a superkey
- Decompose $R$ into $R_1$ and $R_2$, where
  - $R_1$ has attributes $X \cup Y$
  - $R_2$ has attributes $X \cup Z$ (where $Z$ contains $R$ attributes not in $X$ or $Y$)
- Repeat until all relations are in 4NF
- Almost identical to BCNF decomposition algorithm
- Any decomposition on a 4NF violation is lossless

4NF decomposition example

<table>
<thead>
<tr>
<th>User (uid, gid, place)</th>
<th>4NF violation: uid $\rightarrow$ gid</th>
</tr>
</thead>
<tbody>
<tr>
<td>uid</td>
<td>gid</td>
</tr>
<tr>
<td>142</td>
<td>dps</td>
</tr>
<tr>
<td>456</td>
<td>abc</td>
</tr>
<tr>
<td>456</td>
<td>gov</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Member (uid, gid)</th>
<th>4NF</th>
</tr>
</thead>
<tbody>
<tr>
<td>uid</td>
<td>gid</td>
</tr>
<tr>
<td>142</td>
<td>dps</td>
</tr>
<tr>
<td>456</td>
<td>abc</td>
</tr>
<tr>
<td>456</td>
<td>gov</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Visited (uid, place)</th>
<th>4NF</th>
</tr>
</thead>
<tbody>
<tr>
<td>uid</td>
<td>place</td>
</tr>
<tr>
<td>142</td>
<td>Springfield</td>
</tr>
<tr>
<td>456</td>
<td>Australia</td>
</tr>
<tr>
<td>456</td>
<td>Morocco</td>
</tr>
</tbody>
</table>

Other kinds of dependencies and normal forms

- Dependency preserving decompositions
- Join dependencies
- Inclusion dependencies
- 5NF, 3NF, 2NF
- See book if interested (not covered in class)

Summary

- Philosophy behind BCNF, 4NF:
  - Data should depend on the key, the whole key, and nothing but the key!
  - You could have multiple keys though
- Redundancy is not desired typically
  - not always, mainly due to performance reasons
- Functional/multivalued dependencies – capture redundancy
- Decompositions – eliminate dependencies (should not be lossy!)
- Normal forms
  - Guarantees certain non-redundancy
  - BCNF, and 4NF
- How to decompose into BCNF, 4NF
- Chase