Normalization

Dr. Shrutilipi Bhattacharjee, Assistant Professor, Dept. of IT, NIT Karnataka, India

Third Normal Form

- A relation schema *R* is in **third normal form (3NF)** if for all:

  $$\alpha \rightarrow \beta \text{ in } F^+$$

  at least one of the following holds:
  - $\alpha \rightarrow \beta$ is trivial (i.e., $\beta \in \alpha$)
  - α is a superkey for *R*
  - Each attribute *A* in $\beta - \alpha$ is contained in a candidate key for *R*

    (**NOTE**: each attribute may be in a different candidate key)

- If a relation is in BCNF, it is in 3NF (since in BCNF one of the first two conditions above must hold)

- Third condition is a minimal relaxation of BCNF to ensure dependency preservation (will see why later)

3NF Example

- Consider a schema:

$$dept\_advisor(s\_ID, i\_ID, dept\_name)$$

- With function dependencies:

$$i\_ID \rightarrow dept\_name$$
$$s\_ID, dept\_name \rightarrow i\_ID$$

- Two candidate keys = {*s_ID, dept_name*}, {*s_ID, i_ID*}

- We have seen before that *dept_advisor* is not in BCNF

- *R*, however, is in 3NF
  - *s_ID, dept_name* is a superkey
  - $i\_ID \rightarrow dept\_name$ *and* *i_ID* is NOT a superkey, but:
    - $\{dept\_name\} - \{i\_ID\} = \{dept\_name\}$ and
    - *dept_name* is contained in a candidate key

Redundancy in 3NF

- Consider the schema R below, which is in 3NF
  - $R = (J, K, L)$
  - $F = \{JK \rightarrow L, L \rightarrow K\}$
  - And an instance table:

J	L	K
j_1	l_1	k_1
j_2	l_1	k_1
j_3	l_1	k_1
null	l_2	k_2

- What is wrong with the table?
  - Repetition of information
  - Need to use null values (e.g., to represent the relationship l_2, k_2, where there is no corresponding value for J)

Goals of Normalization

- Let R be a relation scheme with a set F of functional dependencies
- Decide whether a relation scheme R is in "good" form
- In the case that a relation scheme R is not in "good" form, need to decompose it into a set of relation scheme $\{R_1, R_2, ..., R_n\}$ such that:
  - Each relation scheme is in good form
  - The decomposition is a lossless decomposition
  - Preferably, the decomposition should be dependency preserving

How Good is BCNF?

- There are database schemas in BCNF that do not seem to be sufficiently normalized
- Consider a relation

inst_info (ID, child_name, phone)

  - Where an instructor may have more than one phone and can have multiple children
  - Instance of *inst_info*

ID	*child_name*	*phone*
99999	David	512-555-1234
99999	David	512-555-4321
99999	William	512-555-1234
99999	William	512-555-4321

- There are no non-trivial functional dependencies and therefore the relation is in BCNF

- Insertion anomalies, i.e., if we add a phone 981-992-3443 to 99999, we need to add two tuples:

(99999, David, 981-992-3443)
(99999, William, 981-992-3443)

Higher Normal Forms

- Therefore, it is better to decompose *inst_info* into:
  - *inst_child*:

ID	*child_name*
99999	David
99999	William

  - *inst_phone:*

ID	*phone*
99999	512-555-1234
99999	512-555-4321

- This suggests the need for higher normal forms, such as Fourth Normal Form (4NF), which we shall see later

Functional-Dependency Theory Roadmap

- We now consider the formal theory that tells us which functional dependencies are implied logically by a given set of functional dependencies
- We then develop algorithms to generate lossless decompositions into BCNF and 3NF
- We then develop algorithms to test if a decomposition is dependency-preserving

Closure of a Set of Functional Dependencies

- Given a set F set of functional dependencies, there are certain other functional dependencies that are logically implied by F
  - If $A \rightarrow B$ and $B \rightarrow C$, then we can infer that $A \rightarrow C$
  - etc.

- The set of ***all*** functional dependencies logically implied by F is the **closure** of F

- We denote the *closure* of F by $\boldsymbol{F^+}$

Closure of a Set of Functional Dependencies

- We can compute F^+, the closure of F, by repeatedly applying **Armstrong's Axioms:**
  - **Reflexive rule:** If $\beta \subseteq \alpha$, then $\alpha \rightarrow \beta$
  - **Augmentation rule:** If $\alpha \rightarrow \beta$, then $\gamma\alpha \rightarrow \gamma\beta$
  - **Transitivity rule:** If $\alpha \rightarrow \beta$, and $\beta \rightarrow \gamma$, then $\alpha \rightarrow \gamma$

- These rules are:
  - **Sound:** Generate only functional dependencies that actually hold, and
  - **Complete:** Generate all functional dependencies that hold

Example of F+

- $R = (A, B, C, G, H, I)$

  $F = \{ A \rightarrow B$

  $A \rightarrow C$

  $CG \rightarrow H$

  $CG \rightarrow I$

  $B \rightarrow H\}$

- Some members of F^+
  - $A \rightarrow H$
    - By transitivity from $A \rightarrow B$ *and* $B \rightarrow H$
  - $AG \rightarrow I$
    - By augmenting $A \rightarrow C$ with G, to get $AG \rightarrow CG$ and then transitivity with $CG \rightarrow I$
  - $CG \rightarrow HI$
    - By augmenting $CG \rightarrow I$ to infer $CG \rightarrow CGI$, and augmenting of $CG \rightarrow H$ to infer $CGI \rightarrow HI$, and then transitivity

Procedure for Computing F+

- To compute the closure of a set of functional dependencies F:

  $F^+ = F$
  repeat
      for each functional dependency f in F^+
          apply reflexivity and augmentation rules on f
          add the resulting functional dependencies to F^+
      for each pair of functional dependencies f_1 and f_2 in F^+
          if f_1 and f_2 can be combined using transitivity
              then add the resulting functional dependency to F^+
  until F^+ does not change any further

- **NOTE**: We shall see an alternative procedure for this task later

Closure of Functional Dependencies

- Additional derived rules:
  - **Union rule**: If $\alpha \rightarrow \beta$ holds and $\alpha \rightarrow \gamma$ holds, then $\alpha \rightarrow \beta\gamma$ holds
  - **Decomposition rule**: If $\alpha \rightarrow \beta\gamma$ holds, then $\alpha \rightarrow \beta$ holds and $\alpha \rightarrow \gamma$ holds
  - **Pseudotransitivity rule**: If $\alpha \rightarrow \beta$ holds and $\gamma\beta \rightarrow \delta$ holds, then $\alpha\gamma \rightarrow \delta$ holds

- The above rules can be inferred from Armstrong's axioms

Closure of Attribute Sets

- Given a set of attributes a, define the ***closure*** of a **under** *F* (denoted by a^+) as the set of attributes that are functionally determined by a under *F*

- Algorithm to compute a^+, the closure of a under *F*

result := a;
while (changes to *result*) **do**
  for each $\beta \rightarrow \gamma$ **in** *F* **do**
   begin
    if $\beta \subseteq$ *result* **then** *result* := *result* $\cup\ \gamma$
   end

Example of Attribute Set Closure

- $R = (A, B, C, G, H, I)$
- $F = \{A \rightarrow B$
  $A \rightarrow C$
  $CG \rightarrow H$
  $CG \rightarrow I$
  $B \rightarrow H\}$

- $(AG)^+$
  1. $result = AG$
  2. $result = ABCG$ $(A \rightarrow C$ and $A \rightarrow B)$
  3. $result = ABCGH$ $(CG \rightarrow H$ and $CG \subseteq AGBC)$
  4. $result = ABCGHI$ $(CG \rightarrow I$ and $CG \subseteq AGBCH)$

- Is AG a candidate key?
  - Is AG a super key?
    - Does $AG \rightarrow R$? == Is R $\supseteq$ $(AG)^+$
  - Is any subset of AG a superkey?
    - Does $A \rightarrow R$? == Is R $\supseteq$ $(A)^+$
    - Does $G \rightarrow R$? == Is R $\supseteq$ $(G)^+$
    - In general: Check for each subset of size $n - 1$

Uses of Attribute Closure

There are several uses of the attribute closure algorithm:

- Testing for superkey:
  - To test if α is a superkey, we compute α^{+}, and check if α^{+} contains all attributes of *R*
- Testing functional dependencies
  - To check if a functional dependency $\alpha \rightarrow \beta$ holds (or, in other words, is in F^{+}), just check if $\beta \subseteq \alpha^{+}$
  - That is, we compute α^{+} by using attribute closure, and then check if it contains β
  - Is a simple and cheap test, and very useful
- Computing closure of F
  - For each $\gamma \subseteq R$, we find the closure γ^{+}, and for each $S \subseteq \gamma^{+}$, we output a functional dependency $\gamma \rightarrow S$

Next Lecture

Normalization

Thank you for your attention...

Any question?

Contact:
Department of Information Technology, NITK Surathkal, India
6th Floor, Room: 13
Phone: +91-9477678768
E-mail: shrutilipi@nitk.edu.in