IIT Madras
BSc Degree

Module 23

Partha Pratim Das

Database Management Systems

Module 23: Relational Database Design/3

Partha Pratim Das

Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

ppd@cse.iitkgp.ac.in

Module Recap

- Introduced the notion of Functional Dependencies

Module Objectives

- To develop the theory of functional dependencies
- To understand how a schema can be decomposed for a 'good' design using functional dependencies

Module Outline

- Functional Dependency Theory
- Decomposition Using Functional Dependencies

Functional Dependency Theory

Functional Dependencies: Armstrong's Axioms

- Given a set of Functional Dependencies F, we can infer new dependencies by the **Armstrong's Axioms**:
  - **Reflexivity**: if $\beta \subseteq \alpha$, then $\alpha \rightarrow \beta$
  - **Augmentation**: if $\alpha \rightarrow \beta$, then $\gamma\alpha \rightarrow \gamma\beta$
  - **Transitivity**: if $\alpha \rightarrow \beta$ and $\beta \rightarrow \gamma$, then $\alpha \rightarrow \gamma$
- These axioms can be repeatedly applied to generate new FDs and added to F
- A new FD obtained by applying the axioms is said to the **logically implied** by F
- The process of generations of FDs terminate after finite number of steps and we call it the **Closure Set** F^+ for FDs F. This is the set of **all** FDs logically implied by F
- Clearly, $F \subseteq F^+$
- These axioms are
  - **Sound** (generate only functional dependencies that actually hold), and
  - **Complete** (eventually generate all functional dependencies that hold)
- Prove the axioms from definitions of FDs
- Prove the soundness and completeness of the axioms

Functional Dependencies (2): Closure of a Set FDs

- $F = \{A \rightarrow B, B \rightarrow C\}$
- $F^+ = \{A \rightarrow B, B \rightarrow C, A \rightarrow C\}$

Functional Dependencies (3): Closure of a Set FDs

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

  $F = \{A \rightarrow B$

  $A \rightarrow C$

  $CG \rightarrow H$

  $CG \rightarrow I$

  $B \rightarrow H\}$

- 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 $CG \rightarrow I$ with CG to infer $CG \rightarrow CGI$, and augmenting $CG \rightarrow H$ with I to infer $CGI \rightarrow HI$, and then transitivity

Functional Dependencies (4): Closure of a Set FDs: Computing F^+

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

  $F^+ \leftarrow F$
  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

Functional Dependencies (5): Armstrong's Axioms: Derived Rules

- Additional Derived Rules:
  - **Union**: if $\alpha \rightarrow \beta$ holds and $\alpha \rightarrow \gamma$ holds, then $\alpha \rightarrow \beta\gamma$ holds
  - **Decomposition**: if $\alpha \rightarrow \beta\gamma$ holds, then $\alpha \rightarrow \beta$ holds and $\alpha \rightarrow \gamma$ holds
  - **Pseudotransitivity**: if $\alpha \rightarrow \beta$ holds and $\gamma\beta \rightarrow \delta$ holds, then $\alpha\gamma \rightarrow \delta$ holds
- The above rules can be inferred from basic Armstrong's axioms (and hence are not included in the basic set). They can be proven independently too
  - **Reflexivity**: if $\beta \subseteq \alpha$, then $\alpha \rightarrow \beta$
  - **Augmentation**: if $\alpha \rightarrow \beta$, then $\gamma\alpha \rightarrow \gamma\beta$
  - **Transitivity**: if $\alpha \rightarrow \beta$ and $\beta \rightarrow \gamma$, then $\alpha \rightarrow \gamma$
- Prove the Rules from:
  - Basic Axioms
  - The definitions of FDs

Functional Dependencies (6): Closure of Attribute Sets

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

  result $\leftarrow \alpha$
  while (changes to *result*) **do**
      for each $\beta \rightarrow \gamma$ **in** F **do**
          begin
              if $\beta \subseteq$ result **then** *result* $\leftarrow$ *result* $\cup\ \gamma$
          end

Functional Dependencies (7): Closure of Attribute Sets: Example

- $R = (A, B, C, G, H, I)$
- $F = \{A \rightarrow B,\ A \rightarrow C,\ CG \rightarrow H,\ CG \rightarrow I,\ B \rightarrow H\}$
- $(AG)^+$
  a) result = AG
  b) result = ABCG ($A \rightarrow C$ and $A \rightarrow B$)
  c) result = ABCGH ($CG \rightarrow H$ and $CG \subseteq AGBC$)
  d) result = ABCGHI ($CG \rightarrow I$ and $CG \subseteq AGBCH$)
- Is AG a candidate key?
  a) Is AG a super key?
    i) Does $AG \rightarrow R$? == Is $(AG)^+ \supseteq R$
  b) Is any subset of AG a superkey?
    i) Does $A \rightarrow R$? == Is $(A)^+ \supseteq R$
    ii) Does $G \rightarrow R$? == Is $(G)^+ \supseteq R$

Functional Dependencies (7): Closure of Attribute Sets: Use

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 $\gamma \rightarrow S$.

Decomposition using Functional Dependencies

BCNF: Boyce-Codd Normal Form

- A relation schema R is in BCNF with respect to a set F of FDs if for all FDs in F^+ of the form

  $\alpha \rightarrow \beta$, where $\alpha \subseteq R$ and $\beta \subseteq R$ at least one of the following holds:
  - $\alpha \rightarrow \beta$ is trivial (that is, $\beta \subseteq \alpha$)
  - α is a superkey for R
- Example schema *not in* BCNF:

  instr_dept (<u>ID</u>, name, salary, <u>dept_name</u>, building, budget)
- because the non-trivial dependency *dept_name* $\rightarrow$ *building, budget* holds on *instr_dept*, but *dept_name* is not a superkey

BCNF (2): Decomposition

- If in schema R and a non-trivial dependency $\alpha \rightarrow \beta$ causes a violation of BCNF, we decompose R into:
  - $\alpha \cup \beta$
  - $(R - (\beta - \alpha))$
- In our example,
  - $\alpha = dept\_name$
  - $\beta = building, budget$
  - $dept\_name \rightarrow building,\ budget$

  inst_dept is replaced by
  - $(\alpha \cup \beta) = (dept\_name,\ building,\ budget)$
    - $dept\_name \rightarrow building,\ budget$
  - $(R - (\beta - \alpha)) = (ID,\ name,\ salary,\ dept\_name)$
    - $ID \rightarrow name,\ salary,\ dept\_name$

BCNF (3): Lossless Join

- If we decompose a relation R into relations R_1 and R_2:
  - Decomposition is lossy if $R_1 \bowtie R_2 \supset R$
  - Decomposition is lossless if $R_1 \bowtie R_2 = R$
- To check for lossless join decomposition using FD set, following must hold:
  - Union of Attributes of R_1 and R_2 must be equal to attribute of R

$$R_1 \cup R_2 = R$$

  - Intersection of Attributes of R_1 and R_2 must not be NULL

$$R_1 \cap R_2 \neq \Phi$$

  - Common attribute must be a key for at least one relation (R_1 or R_2)

$$R_1 \cap R_2 \rightarrow R_1 \textit{ or } R_1 \cap R_2 \rightarrow R_2$$

- Prove that BCNF ensures Lossless Join

BCNF (4): Dependency Preservation

- Constraints, including FDs, are costly to check in practice unless they pertain to only one relation
- If it is sufficient to test only those dependencies on each individual relation of a decomposition in order to ensure that *all* functional dependencies hold, then that decomposition is *dependency preserving*.
- It is not always possible to achieve both BCNF and dependency preservation. Consider:
  - $R = CSZ$, $F = \{CS \rightarrow Z, Z \rightarrow C\}$
  - Key $= CS$
  - $CS \rightarrow Z$ satisfies BCNF, but $Z \rightarrow C$ violates
  - Decompose as: $R_1 = ZC, R_2 = CSZ - (C - Z) = SZ$
  - $R_1 \cup R_2 = CSZ = R$, $R_1 \cap R_2 = Z \neq \Phi$, and $R_1 \cap R_2 = Z \rightarrow ZC = R_1$. So it has *lossless join*
  - However, we cannot check $CS \rightarrow Z$ without doing a join. Hence it is not *dependency preserving*
- We consider a weaker normal form, known as **Third Normal Form (3NF)**

3NF: Third Normal Form

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

  $$\alpha \rightarrow \beta \in F^+$$

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

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, 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-join decomposition
    - Preferably, the decomposition should be dependency preserving

Problems with Decomposition

There are three potential problems to consider:

- May be impossible to reconstruct the original relation! (Lossiness)
- Dependency checking may require joins
- Some queries become more expensive
  - What is the building for an instructor?

Tradeoff: Must consider these issues vs. redundancy

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

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

inst_info

How good is BCNF? (2)

- There are no non-trivial functional dependencies and therefore the relation is in BCNF
- Insertion anomalies – that is, 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)

How good is BCNF? (3)

- 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 the Fourth Normal Form (4NF)

Module Summary

- Introduced the theory of functional dependencies
- Discussed issues in "good" design in the context of functional dependencies

Slides used in this presentation are borrowed from http://db-book.com/ with kind permission of the authors.
Edited and new slides are marked with "PPD".