7.4 Functional-Dependency Theory

- 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

7.4.1 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.
  - e.g.

    If $A \rightarrow B$ and $B \rightarrow C$, then we can infer that $A \rightarrow C$

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

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

北京邮电大学

$\boldsymbol{F} = \{A \to B\ ,\ B \to C\ \};$

f: $A \to C$ is logically implied by $\boldsymbol{F}$

	A	B	C
t1	1	4	2
t2	3	5	6
t3	4	4	2
t4	7	3	8
t5	9	1	0

Fig. 8.0.5

- **Def.** Given a schema ***R***, a functional dependency ***f*** on ***R*** is ***logically implied*** by a set of ***FD*** ***F*** on ***R*** , if

  every instance $r(R)$ that satisfies ***F*** also satisfies ***f***

  - e.g. Fig. 8.0.5

- **Def.** Given a set ***F*** of functional dependencies, the ***closure* of *F*,** denoted as F^+

  - $F^+ = \{ f \mid f \text{ is logically implied by } F \}$
  - e.g. in Fig. 8.0.5, $\{A \rightarrow B , B \rightarrow C \}^+$

    $= \{A \rightarrow B , B \rightarrow C, A \rightarrow C, \ldots \}$

Closure of a Set of Functional Dependencies

- We can find F^+, the closure of F, by repeatedly applying **Armstrong's Axioms:**
  - if $\beta \subseteq \alpha$, then $\alpha \rightarrow \beta$ **(reflexivity自反律)**
  - if $\alpha \rightarrow \beta$, then $\gamma\,\alpha \rightarrow \gamma\,\beta$ **(augmentation增广律)**
  - if $\alpha \rightarrow \beta$, and $\beta \rightarrow \gamma$, then $\alpha \rightarrow \gamma$ **(transitivity传递律)**
- These rules are
  - **Sound（正确有效的 ）** (generate only functional dependencies that actually hold), and
  - **Complete（完备的）**(generate all functional dependencies that hold).

Closure of Functional Dependencies (Cont.)

- Additional rules:
  - If $\alpha \rightarrow \beta$ holds and $\alpha \rightarrow \gamma$ holds, then $\alpha \rightarrow \beta\gamma$ holds (**union合并律**)
  - If $\alpha \rightarrow \beta\gamma$ holds, then $\alpha \rightarrow \beta$ holds and $\alpha \rightarrow \gamma$ holds (**decomposition分解律**)
  - If $\alpha \rightarrow \beta$ holds and $\gamma\beta \rightarrow \delta$ holds, then $\alpha\gamma \rightarrow \delta$ holds (**pseudotransitivity伪传递律**)

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

An Example of Functional Dependency

- Question

  Which rule about functional dependencies shown below is right

  - A. if $\alpha \rightarrow \beta$ then $\beta \rightarrow \alpha$
  - B if $A \rightarrow C$, $BC \rightarrow D$ then $AB \rightarrow D$
  - C. if $AB \rightarrow C$ then $B \rightarrow C$
  - D if $\alpha \subseteq \beta$, then $\alpha \rightarrow \beta$

- Answer: B

Example

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

  $F = \{ A \rightarrow B$

  $A \rightarrow C$

  $CG \rightarrow H$

  $CG \rightarrow I$

  $B \rightarrow H\}$
- 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 $CG \rightarrow I$

Example (cont.)

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

  $F = \{\ A \rightarrow B$

  $A \rightarrow C$

  $CG \rightarrow H$

  $CG \rightarrow I$

  $B \rightarrow H\}$

- some members of F^+
  - $CG \rightarrow HI$
    - by augmenting $CG \rightarrow I$ to infer $CG \rightarrow \mathrm{CG}I$,

      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

7.4.2 Closure of Attribute Sets

- **Def**. An attribute B is functionally determined by α if $\alpha \rightarrow B$
- **Def.** Given a set of attributes α , the ***closure of α under F*** , denoted by α^+, is

  {β | β is *functionally determined by* α **under *F***}

-

Closure of Attributes (cont.)

Input: α , F

Output: α^{+}

result := α ;

while (changes to *result*) **do**

  for each $\beta \rightarrow \gamma$ **in** F **do**

   begin

    if $\beta \subseteq$ *result* **/* *result* =(β, …)**

    then *result* := result $\cup\ \gamma$

   end

Fig.7.9 An *efficient* algorithm to compute α^{+} under F

An Example

- R = (A, B, C, G, H, I)
  - F = {$A \to B$,
    $A \to C$,
    $CG \to H$, $CG \to I$
    $B \to H$ }
  - Computing $(AG)^+$

1. *result* = AG /* or denoted as {A, G }
2. *result* = ABCG /* $A \to C$, $A \to B$
3. *result* = ABCGH /* $CG \to H$
4. *result* = ABCGHI /* $CG \to I$

/* or { A, B, C, G, H, I }

An Example (cont.)

- $(AG)^+ = R$, AG is a *superkey* of R

- Is *AG* a candidate key?
  - step1. is AG a super key?
    - does $AG \rightarrow R$? == Is $(AG)^+ = R$
    - *yes*
  - step2. is any subset of AG a superkey?
    - does $A \rightarrow R$? == is $(A)^+ = R$?, *no*
    - does $G \rightarrow R$? == is $(G)^+ = R$?, *no*
  - so, *AG* is a candidate key

Uses of Attribute Closure

- **Usage-I**. Testing for superkey

  To test whether α is a superkey of R ***under F*** , i.e. whether $\alpha \rightarrow R$, we check if

  $$R = \alpha^+$$

- **Usage-II.** Testing functional dependencies

  To determine whether or not $\alpha \rightarrow \beta$ holds on R under F, we 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

- **Usage-III.** Computing closure $\boldsymbol{F}^+$

  for each $\gamma \subseteq R$, compute $\gamma^+=\{S\}$ under $\boldsymbol{F}$;

  for each $S \subseteq \gamma^+$, output $\gamma \rightarrow S$ as a functional dependency in F^+

Closure of Attributes (cont.)

- **Def.** For functional dependencies $\boldsymbol{F}$ and $\boldsymbol{G}$, if

  $$\boldsymbol{F}^{+} = \boldsymbol{G}^{+}$$

  then $\boldsymbol{F}$ and $\boldsymbol{G}$ are equivalent

- E.g. F={A $\rightarrow$ B , B $\rightarrow$ C } is equivalent to

  G={A $\rightarrow$ B , B $\rightarrow$ C, A $\rightarrow$ C }

7.4.3 Canonical Cover

- checking *all* $\alpha \to \beta$ in ***F*** is time-consuming
  - ***F*** may have *redundant* dependencies that can be inferred from the others
- Sets of functional dependencies may have redundant dependencies that can be inferred from the others
  - E.g. $A \to C$ is redundant in: $\{A \to B,\ \ B \to C, A \to C\}$
  - Parts of a functional dependency may be redundant
    - E.g. on RHS: $\{A \to B,\ \ B \to C,\ \ A \to CD\}$ can be simplified to

      $$\{A \to B,\ \ B \to C,\ \ A \to D\}$$

    - E.g. on LHS: $\{\mathrm{A} \to B,\ \ B \to C,\ \ AC \to D\}$ can be simplified to

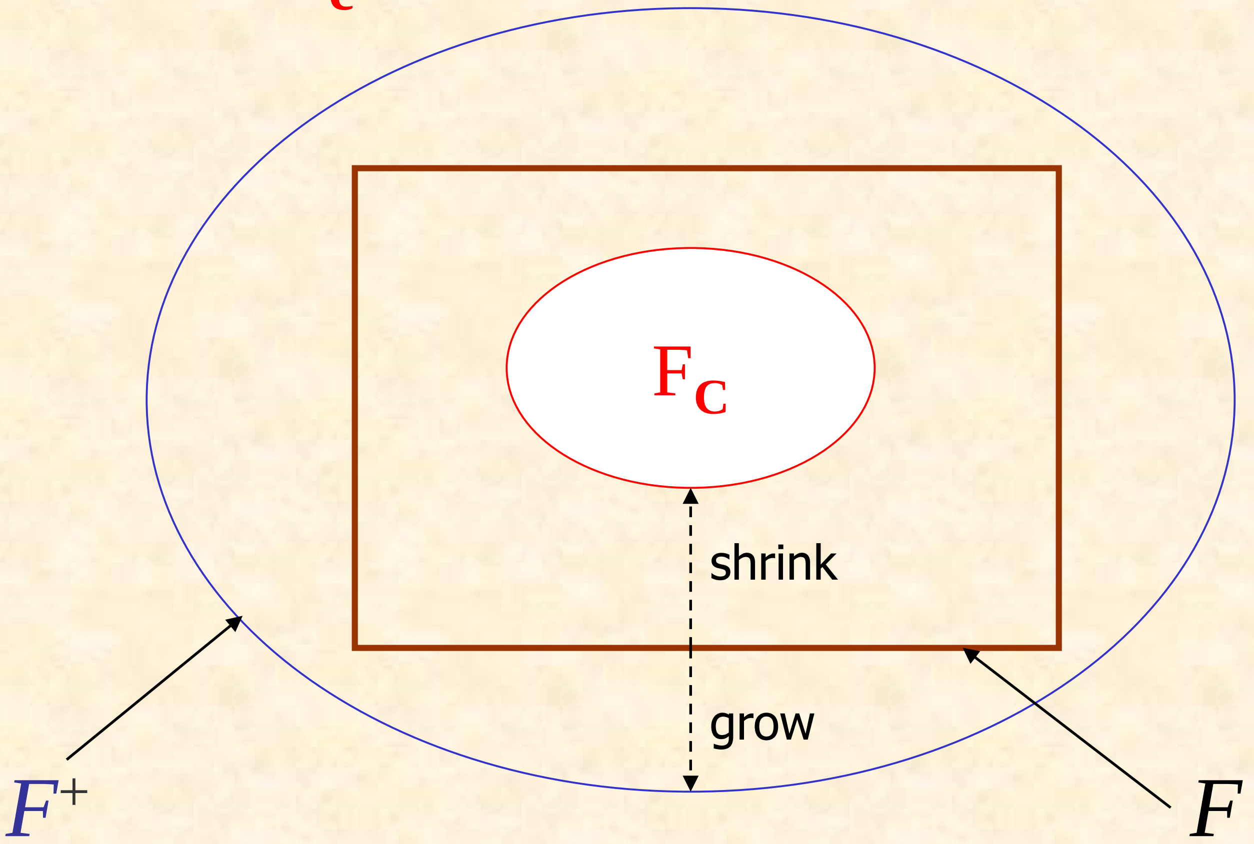
      $$\{\mathrm{A} \to B,\ \ B \to C,\ \ A \to D\}$$

Canonical Cover

- It is desirable to test a “***minimal***” set of functional dependencies ***equivalent*** to F
- Intuitively, a ***canonical cover*** (正则覆盖) of ***F***, denoted as F_c, is
  - a “***minimal***” set of functional dependencies ***equivalent*** to ***F***
    - without redundant functional dependencies or attributes
    - $F^+ = F_c^+$

F, F^+, and F_c

$$F^+ = F_c^{\,+}$$



Extraneous Attributes

- **Def.** Consider a set $\boldsymbol{F}$ of functional dependency and $\alpha \to \beta$ in $\boldsymbol{F}$,
  - attribute A is **extraneous（无关的） in α,** if
    - $A \in \alpha$, and
    - F implies/***is equivalent to*** $(F - \{\alpha \to \beta\}) \cup \{(\alpha - A) \to \beta\}$

      that is, $F^+ \supseteq (F - \{\alpha \to \beta\}) \cup \{(\alpha - A) \to \beta\}$
  - attribute A is **extraneous in β,** if
    - $A \in \beta$, and
    - the set of functional dependencies

      $(F - \{\alpha \to \beta\}) \cup \{\alpha \to (\beta - A)\}$ implies F

      that is, $((F - \{\alpha \to \beta\}) \cup \{\alpha \to (\beta - A)\})^+ \supseteq F$

Extraneous Attributes

- Example: Given $F = \{A \rightarrow C, AB \rightarrow C\}$
  - B is extraneous in $AB \rightarrow C$ because $\{A \rightarrow C, AB \rightarrow C\}$ logically implies $A \rightarrow C$ (I.e. the result of dropping B from $AB \rightarrow C$).
- Example: Given $F = \{A \rightarrow C, AB \rightarrow CD\}$
  - C is extraneous in $AB \rightarrow CD$ since $\mathrm{AB} \rightarrow C$ can be inferred even after deleting C

Testing if an Attribute is Extraneous

- Consider a set F of functional dependencies and the functional dependency $\alpha \rightarrow \beta$ in F ,
  - to test if attribute $A \in \alpha$ is extraneous

    $(\alpha - A) \rightarrow \beta$ under F?

    1. compute $(\alpha\text{-}A)^+$ using the dependencies in F
    2. check $\beta \in (\alpha - A)^+$?

    if it does, then $(\alpha - A) \rightarrow \beta$ holds, A is extraneous

Testing if an Attribute is Extraneous

- to test if attribute $A \in \beta$ is extraneous in β

  $\alpha \rightarrow A$ under F'

1. compute α^+ using only the dependencies in

   $F' = (F - \{\alpha \rightarrow \beta\}) \cup \{\alpha \rightarrow (\beta - A)\}$,

2. check $A \in \alpha^+$? if it does, $\alpha \rightarrow A$ holds, A is extraneous

Canonical Cover

- A **canonical cover** for F is a set of dependencies F_c such that
  - F logically implies all dependencies in F_c, and
  - F_c logically implies all dependencies in F, and
  - No functional dependency in F_c contains an extraneous attribute, and
  - Each left side of functional dependency in F_c is unique, that is, there are no two dependencies $\alpha_1 \rightarrow \beta_1$, $\alpha_2 \rightarrow \beta_2$ in F_c, such that $\alpha_1 = \alpha_2$

Canonical Cover

- To compute a canonical cover for *F*:

repeat

1. Use the union rule to replace any dependencies in *F*

$\alpha_1 \to \beta_1$ and $\alpha_1 \to \beta_2$ with $\alpha_1 \to \beta_1 \beta_2$

2. Find a functional dependency $\alpha \to \beta$ with an extraneous attribute either in α or in β

if an extraneous attribute is found, delete it from $\alpha \to \beta$

until *Fc* does not change

Canonical Cover (cont.)

- Note: ***Union*** rule may become applicable after some extraneous attributes have been deleted, so it has to be re-applied

- Note:

  for a set *F* of functional dependencies, there may be ***several*** $F_{c,}$

Example of Canonical Cover

- Computing a Canonical Cover for

  R = (A, B, C)

  $F1 = \{ A \rightarrow BC,$

  $B \rightarrow C,$

  $A \rightarrow B,$

  $AB \rightarrow C\}$

  - applying the **Union** rule to combine $A \rightarrow BC$ and $A \rightarrow B$ into $A \rightarrow BC$, F_c becomes

    $$F = \{A \rightarrow BC, B \rightarrow C, AB \rightarrow C\}$$

  - A is extraneous in $AB \rightarrow C$, because
    - F logically implies $\{A \rightarrow BC, B \rightarrow C\} \cup \{B \rightarrow C\}$; or

Example of Canonical Cover (cont.)

- $\{AB\text{-}A\}^+ = \{B\}^+$ under F is $\{BC\}$, and contains C,
- F_c is now $\{A \rightarrow BC, B \rightarrow C\}$
- C is extraneous in $\boldsymbol{A} \rightarrow BC$, because
  - $\{A \rightarrow B, B \rightarrow$ C$\}$ logically implies $\{A \rightarrow BC, B \rightarrow C\}$, or
  - $(\boldsymbol{A})^+$ under $\{A \rightarrow B, B \rightarrow C\}$ is {BC}, and contains C
- F_c is: $$A \rightarrow B$$ $$B \rightarrow C$$

7.4.4 Lossless-join Decomposition

- For the case of $R = (R_1, R_2)$, we require that for all possible relations r on schema R

$$r = \Pi_{R1}(r) \bowtie \Pi_{R2}(r)$$

- A decomposition of R into R_1 and R_2 is lossless join if at least one of the following dependencies is in F^+:
  - $R_1 \cap R_2 \rightarrow R_1$
  - $R_1 \cap R_2 \rightarrow R_2$
- The above functional dependencies are a sufficient condition for lossless join decomposition; the dependencies are a necessary condition only if all constraints are functional dependencies

$r(R) = r_1 \bowtie r_2 \bowtie r_3$?

/*无损连接分解中，R所对应的r被分成若干垂直片段$\Pi_{Ri}(r)$，各垂直片段通过自然连接可恢复r中的数据，保证了数据的完整性/分解可恢复性

- $R(A_1, A_2, \ldots, A_i, \ldots, A_n)$, $R = R_1 \cup R_2 \cup R_3$

$r(R)$

A_1	A_2		A_i		A_{n-1}	A_n
*	*	*	*	*	*	*
*	*	*	*	*	*	*
*	*	*	*	*	*	*
*	*	*	*	*	*	*

$r_1 = \Pi_{R1}(r)$ $r_2 = \Pi_{R2}(r)$ … $r_3 = \Pi_{R3}(r)$

Fig. Decomposition of R and $r(R)$

Lossless Decomposition (cont.)

- Def. Lossy decomposition /*有损连接分解*/

  $$r \neq \Pi_{R1}(r) \bowtie \Pi_{R2}(r) \bowtie \ldots \bowtie \Pi_{Rn}(r)$$

  - also known as *lossy-join* (有损连接分解) decomposition

- Lossy decompositions may result in *information loss*

Example

- $R = (A, B, C)$
  $F = \{A \rightarrow B, B \rightarrow C)$
  - Can be decomposed in two different ways
- $R_1 = (A, B), \quad R_2 = (B, C)$
  - Lossless-join decomposition:
    $$R_1 \cap R_2 = \{B\} \text{ and } B \rightarrow BC$$
  - Dependency preserving
- $R_1 = (A, B), \quad R_2 = (A, C)$
  - Lossless-join decomposition:
    $$R_1 \cap R_2 = \{A\} \text{ and } A \rightarrow AB$$
  - Not dependency preserving
    (cannot check $B \rightarrow C$ without computing $R_1 \bowtie R_2$)

7.4.5 Dependency Preservation

- Let F_i be the set of dependencies F^+ that include only attributes in R_i.
  - A decomposition is **dependency preserving**, if

    $$(F_1 \cup F_2 \cup \ldots \cup F_n)^+ = F^+$$
  - If it is not, then checking updates for violation of functional dependencies may require computing joins, which is expensive.

Dependency Preserving (cont.)

- **Def.** For a schema $\boldsymbol{R}$, $\boldsymbol{F}$ holds on $\boldsymbol{R}$, and the decomposition $\{R_1, R_2, .., R_n\}$ of $\boldsymbol{R}$,

  the restriction of $\boldsymbol{F}$ to R_i, denoted as $\boldsymbol{F}_i$ is defined as

  $$\boldsymbol{F}_i = \{\alpha \rightarrow \beta \mid \alpha \rightarrow \beta \in \boldsymbol{F}^+ \text{ AND } \alpha\beta \subseteq R_i\}$$

  - i.e. the set of dependencies in F^+ that include only attributes in R_i /*限制、投影
  - e.g. **example** in the *next* slide

Dependency Preserving

- R(A, B, C, D), F ={A →B, B →C, A →D, B→D} on R
  - F^+ = F ∪ {A →C}
  - lossless decomposition:

    R_1(A, B, C), F$_1$ ={A →B, B→C, A→C} on R_1,

    R_2(B, D), F$_2$= {B→D} on R_2

  - note: A →D is lost on R_1 and R_2

Testing for Dependency Preservation

- for **F**= {α→β}, apply the following procedure for each α→β

  result = α
  while (changes to *result*) do
  for each R_i in the decomposition
  $t = (result \cap R_i)^+ \cap R_i$ (with respect to F)
  $result = result \cup t$

/*利用只包含在各个子模式R_i中的中间结果***result***去推导

  - if ***result*** contains all attributes in *β*, then the functional dependency α → β is preserved

- The decomposition is preserved, if and only if

  all α→β in F are preserved.

Testing for Dependency Preservation

- We apply the test on all dependencies in F to check if a decomposition is dependency preserving
- This procedure takes polynomial time, instead of the exponential time required to compute F^+ and $(F_1 \cup F_2 \cup \ldots \cup F_n)^+$

Example

- $R = (A, B, C)$
  $F = \{A \to B$
  $\quad B \to C\}$
  Key = $\{A\}$
- R is not in BCNF
- Decomposition $R_1 = (A, B)$, $R_2 = (B, C)$
  - R_1 and R_2 in BCNF
  - Lossless-join decomposition
  - Dependency preserving

Example

- Student(*sno*, *dept*, *head*),
  $F = \{sno \rightarrow dept, dept \rightarrow head\}$, $F^+ = F \cup \{sno \rightarrow head\} \cup \{\ldots\}$

- Decomposition 1
  $R_1(sno, dept)$, $F_1 = \{sno \rightarrow dept\}$
  $R_2(sno, head)$, $F_2 = \{sno \rightarrow head\}$
  - *lossless*, because
    - $R_1 \cap R_1 = \{sno\}$, and is the key of R_1 and R_2
  - non-dependency preservation, because
    - $(F_1 \cup F_2)^+ \neq F^+$, $dept \rightarrow head$ is lost,

Example of Dependency Preserving (cont.)

- for *dept* $\rightarrow$*head* in **F**,

(1) with respect to R_1,

$\textbf{\textit{result}}=(dept \cap R_1)^+ \cap R_1 = \{dept\}^+ \cap \{sno, dept\}$

$= \{dept, head\} \cap \{sno, dept\}$

$= \{dept\}$;

(2) with respect to R_2,

$\textbf{\textit{result}}=(dept \cap R_2)^+ \cap R_2$

$= \phi^+ \cap \{sno, head\} = \phi$

dept$\rightarrow$*head* is not preserved

Example of Dependency Preserving (cont.)

- Decomposition 2

  $R_1(sno, dept)$, $F_1=\{sno \rightarrow dept\}$

  $R_2(dept, head)$, $F_2=\{dept \rightarrow head\}$

  - *lossless-join*, because
    - $R_1 \cap R_2=\{dept\}$, and is the key of R_2
  - dependency preservation, because
    - $(F_1 \cup F_2)^+ = F^+$