COP 4710: Database Systems
Spring 2004

-Day 12 – February 16, 2004 –
Introduction to Normalization – Part 3

Instructor : Mark Llewellyn
markl@cs.ucf.edu
CC1 211, 823-2790
http://www.cs.ucf.edu/courses/cop4710/spr2004

School of Electrical Engineering and Computer Science
University of Central Florida

Practice Problem Solution

Let R = (C, S, Z)
F = {CS $\rightarrow$Z, Z$\rightarrow$C}
D = {(SZ), (CZ)}

G = F[SZ] $\cup$ F[CZ] $Z = Z \cup ((Z \cap R_i)^+ \cap R_i)$

Test for each fd in F.

Test for CS$\rightarrow$Z

Z = CS,
= {CS} $\cup$ ((CS $\cap$ SZ)$^+$ $\cap$ SZ)
= {CS} $\cup$ ((S)$^+$ $\cap$ SZ)
= {CS} $\cup$ (S)
= **{CS}**
= {CS} $\cup$ ((CS $\cap$ CZ)$^+$ $\cap$ CZ)
= {CS} $\cup$ ((C)$^+$ $\cap$ CZ)
= {CS} $\cup$ (C $\cap$ CZ)
= {CS} $\cup$ (C)
= **{CS}** thus, CS $\rightarrow$Z is not preserved.

Algorithm #1 for Producing a 3NF Decomposition

Algorithm 3NF.1
// input: a relation schema R= (A_1, A_2, …, A_n), a set of fds F, a set of candidate keys K.
// output: a 3NF decomposition of R, called D, which has the lossless join property and the
// functional dependencies are preserved.

3NF.1 (R, F, K)
    a = 0;
    for each fd X $\rightarrow$ Y in F do
        a = a +1;
        R_a = XY;
    endfor
    if [none of the schemes R_b ($1 \leq b \leq a$) contains a candidate key of R] then
        a = a + 1;
        R_a = any candidate key of R
    endif
    if [$\bigcup_{b=1}^{a} R_b \neq R$] then //there are missing attributes

        $R_{a+1} = R - \bigcup_{b=1}^{a} R_b$
    return D = {R_1, R_2, ..., R_{a+1}}
end.

Example – Using Algorithm 3NF.1

Let R = (A, B, C, D, E)
K = {AB, AC}
F = {AB→CDE, AC→BDE, B→C, C→B, C→D, B→E}

Step 1: D = {(ABCDE), (ACBDE), (BC), (CB), (CD), (BE)}

Reduce to: D = {(ABCDE), (BC), (CD), (BE)}

Step 2: Does D contain a candidate key for R?
Yes, in (ABCDE)

Step 3: Are all the attributes of R contained in D?
Yes.

Return D as: **{(ABCDE), (BC), (CD), (BE)}**

Algorithm #2 for Producing a 3NF Decomposition

Algorithm 3NF.2
// input: a relation schema R= (A_1, A_2, …, A_n), a set of fds F, a set of candidate keys K.
// output: a 3NF decomposition of R, called D, which is not guaranteed to have either the
// lossless join property or to preserve the functional dependencies in F.
// This algorithm is based on the removal of transitive dependencies.

3NF.2 (R, F, K)
do
if [$K \rightarrow Y \rightarrow A$ where A is non-prime and not an element of either K or Y] then
decompose R into: $R_1 = \{R - A\}$ with $K_1 = \{K\}$ and $R_2 = \{YA\}$ with $K_2 = \{Y\}$.
repeat until no transitive dependencies exist in any schema
D = union of all 3NF schemas produced above.
test for lossless join
test for preservation of the functional dependencies
end.

Example – Using Algorithm 3NF.2

Let R = (A, B, C, D, E)
K = {AB, AC}
F = {AB→CDE, AC→BDE, B→C, C→B, C→D, B→E}

Step 1: R not in 3NF since AB → C → D
Decompose to: R_1 = (A, B, C, E) with K_1 = K = {AB, AC}
R_2 = (C, D) with K_2 = {C}

Step 2: R_2 in 3NF. R_1 not in 3NF since AB → B → E
Decompose R_1 to: R_{11} = (A, B, C) with K_{11}= K_1 = K = {AB, AC}
R_{12} = (B, E) with K_{12} = {B}

Step 3: R_2, R_{11}, and R_{12} are all in 3NF

Step 4: Test for the lossless join property (see next page).

Step 4: Checking for a Lossless Join in the Decomposition

AB→CDE: (1st time: equates nothing)
AC→BDE: (1st time: equates nothing)
B→C: (1st time: equates a_3 & b_{33})
C→B: (1st time: equates a_2 & b_{12})
C→D: (1st time: equates b_{14}, b_{24}, b_{34}) – stop second row becomes all a's
B→E: (1st time: equates a_5, b_{15}, b_{25})

Decomposition has the lossless join property.

	A	B	C	D	E
(CD)	b_{11}	a_2	a_3	a_4	b_{15}
(ABC)	a_1	a_2	a_3	a_4	b_{15}
(BE)	b_{31}	a_2	a_3	a_4	a_5

Step 5: Testing the Preservation of the Functional Dependencies

Let R = (A, B, C, D, E)
F = {AB→CDE, AC→BDE, B→C, C→B, C→D, B→E}}
D = {(CD), (ABC), (BE)}

G = F[CD] ∪ F[ABC] ∪ F[BE]

$Z = Z \cup ((Z \cap R_i)^+ \cap R_i)$

Test for AB→CDE

Z = AB,
= {AB} ∪ ((AB ∩ CD)$^+$ ∩ CD)
= {AB} ∪ ((∅)$^+$ ∩ CD)
= {AB} ∪ (∅ ∩ CD)
= {AB} ∪ (∅)
= **{AB}**
= {AB} ∪ ((AB ∩ ABC)$^+$ ∩ ABC)
= {AB} ∪ ((AB)$^+$ ∩ ABC)
= {AB} ∪ (ABCDE ∩ ABC)
= {AB} ∪ (ABC)
= **{ABC}**
= {ABC} ∪ ((ABC ∩ BE)$^+$ ∩ BE)
= {ABC} ∪ ((B)$^+$ ∩ BE)
= {ABC} ∪ (BCDE ∩ BE)
= {ABC} ∪ (BE)
= **{ABCE}**

Step 5: Testing the Preservation of the Functional Dependencies (cont.)

Test for AB→CDE continues

$Z = \{ABCE\} \cup ((ABCE \cap CD)^+ \cap CD)$
$= \{ABCE\} \cup ((C)^+ \cap CD)$
$= \{ABCE\} \cup (CBDE \cap CD)$
$= \{ABCE\} \cup (CD)$
$=$ **{ABCDE} thus, AB ® CDE is preserved**

Test for AC→BDE

$Z = AC$
$= \{AC\} \cup ((AC \cap CD)^+ \cap CD)$
$= \{AC\} \cup ((C)^+ \cap CD)$
$= \{AC\} \cup (CBDE \cap CD)$
$= \{AC\} \cup (CD)$
$=$ **{ACD}**
$= \{ACD\} \cup ((ACD \cap ABC)^+ \cap ABC)$
$= \{ACD\} \cup ((AC)^+ \cap ABC)$
$= \{ACD\} \cup (ACBDE \cap ABC)$
$= \{ACD\} \cup (ABC)$
$=$ **{ABCD}**

Step 5: Testing the Preservation of the Functional Dependencies (cont.)

Test for AC→BDE continues

$Z = \{ABCD\} \cup ((ABCD \cap BE)^+ \cap BE)$
$= \{ABCD\} \cup ((B)^+ \cap BE)$
$= \{ABCD\} \cup (BCDE \cap BE)$
$= \{ABCD\} \cup (BE)$
$=$ **{ABCDE} thus, AC ® BDE is preserved**

Test for B→C

$Z = B$
$= \{B\} \cup ((B \cap CD)^+ \cap CD)$
$= \{B\} \cup ((C)^+ \cap CD)$
$= \{B\} \cup (CBDE \cap CD)$
$= \{B\} \cup (CD)$
$=$ **{BCD} thus B ® C is preserved**

Test for C→B

$Z = C$
$= \{C\} \cup ((C \cap CD)^+ \cap CD)$
$= \{C\} \cup ((C)^+ \cap CD)$
$= \{C\} \cup (CBDE \cap CD)$
$= \{C\} \cup (CD)$
$=$ **{CD}**

Step 5: Testing the Preservation of the Functional Dependencies (cont.)

Test for C→B continues

$Z = \{CD\} \cup ((CD \cap ABC)^+ \cap ABC)$
$= \{CD\} \cup ((C)^+ \cap ABC)$
$= \{CD\} \cup (CBDE \cap ABC)$
$= \{CD\} \cup (BC)$
$=$ **{BCD} thus, C ® B is preserved**

Test for C→D

$Z = C$
$= \{C\} \cup ((C \cap CD)^+ \cap CD)$
$= \{C\} \cup ((C)^+ \cap CD)$
$= \{C\} \cup (CBDE \cap CD)$
$= \{C\} \cup (CD)$
$=$ **{CD} thus C ® D is preserved**

Test for B→E

$Z = B$
$= \{B\} \cup ((B \cap CD)^+ \cap CD)$
$= \{B\} \cup ((\varnothing)^+ \cap CD)$
$= \{B\} \cup (\varnothing)$
$=$ **{B}**

Step 5: Testing the Preservation of the Functional Dependencies (cont.)

Test for B→E continues

$Z = \{B\} \cup ((B \cap ABC)^+ \cap ABC)$
$= \{B\} \cup ((B)^+ \cap ABC)$
$= \{B\} \cup (BCDE \cap ABC)$
$= \{BC\} \cup (BC)$
$= \mathbf{\{BC\}}$

$Z = \{BC\}$
$= \{BC\} \cup ((BC \cap ABC)^+ \cap ABC)$
$= \{BC\} \cup ((C)^+ \cap ABC)$
$= \{BC\} \cup (CBDE \cap ABC)$
$= \{BC\} \cup (BC)$
$= \mathbf{\{BC\}}$

$Z = \{BC\}$
$= \{BC\} \cup ((BC \cap BE)^+ \cap BE)$
$= \{BC\} \cup ((B)^+ \cap BE)$
$= \{BC\} \cup (BCDE \cap BE)$
$= \{BC\} \cup (BE)$
$= \mathbf{\{BCE\}}$ **thus, B ® E is preserved.**

Why Use 3NF.2 Rather Than 3NF.1

- Why would you use algorithm 3NF.2 rather than algorithm 3NF.1 when you know that algorithm 3NF.1 will guarantee that both the lossless join property and the preservation of the functional dependencies?
- The answer is that algorithm 3NF.2 will typically produce fewer relational schemas than will algorithm 3NF.1. Although both the lossless join and dependency preservation properties must be independently tested when using algorithm 3NF.2.

Algorithm #3 for Producing a 3NF Decomposition

Algorithm 3NF.3
// input: a relation schema R= (A_1, A_2, …, A_n), a set of fds F.
// output: a 3NF decomposition of R, called D, which is guaranteed to have both the
// lossless join property and to preserve the functional dependencies in F.
// This algorithm is based on the a minimal cover for F (see Day 9 notes page 45).

3NF.3 (R, F)
find a minimal cover for F, call this cover G (see Day 9 page 45 for algorithm)
for each determinant X that appears in G do
create a relation schema { $X \cup A_1 \cup A_2 \cup ... \cup A_m$} where A_i ($1 \leq i \leq m$) represents
all the consequents of fds in G with determinant X.
place all remaining attributes, if any, in a single schema.
if none of the schemas contains a key for R, create an additional schema which
contains any candidate key for R.
end.

Algorithm 3NF.3

- Algorithm 3NF.3 is very similar to algorithm 3NF.1, differing only in how the schemas of the decomposition scheme are created.
  - In algorithm 3NF.1, the schemas are created directly from F.
  - In algorithm 3NF.3, the schemas are created from a minimal cover for F.
- In general, algorithm 3NF.3 should generate fewer relation schemas than algorithm 3NF.1.

Another Technique for Testing the Preservation of Dependencies

- The algorithm given on page 14 of Day 11 notes for testing the preservation of a set of functional dependencies on a decomposition scheme is fairly efficient for computation, but somewhat tedious to do by hand.
- On the next page is an example solving the same problem that we did in the example on page 16 of Day 11, utilizing a different technique which is based on the concept of covers.
- Given D, R, and F, if D = $\{R_1, R_2, ..., R_n)$ then

  $G = F[R_1] \cup F[R_2] \cup F[R_3] \cup ... \cup F[R_n]$ and if every

  functional dependency in F is implied by G, then G covers F.
- The technique is to generate that portion of G^+ that allows us to know if G covers F.

A Hugmongously Big Example Using Different Technique

Let R = (A, B, C, D)

F = {A→B, B→C, C→D, D→A}

D = {(AB), (BC), (CD)}

$G = F[AB] \cup F[BC] \cup F[CD]$

Projection onto schema (AB)

$F[AB] = A^+ \cup B^+ \cup (AB)^+$

$= \{ABCD\} \cup \{ABCD\} \cup \{ABCD\}$

apply projection: = {AB} ∪ {AB} ∪ {AB} = **{AB}**, **A®B is covered**

Projection onto schema (BC)

$F[BC] = B^+ \cup C^+ \cup (BC)^+$

$= \{BCDA\} \cup \{CDAB\} \cup \{BCDA\}$

apply projection: = {BC} ∪ {BC} ∪ {BC} = **{BC}**, **C®C is covered**

A Hugmongously Big Example Using Different Technique (cont.)

Projection onto schema (CD)

$$F[CD] = C^+ \cup D^+ \cup (CD)^+$$
$$= \{CDAB\} \cup \{DABC\} \cup \{CDAB\}$$

apply projection: = $\{CD\} \cup \{CD\} \cup \{CD\} =$ **{CD}**, **C ® D is covered**

- Thus, the projections have covered every functional dependency in F except $D \rightarrow A$. So, now the question becomes does G logically imply $D \rightarrow A$?

- Generate D^+(with respect to G) and if A is in this closure the answer is yes.

$D_G^+ = \{D, C, B, A\}$ Therefore, G ? $D \rightarrow A$

Multi-valued Dependencies and Fourth Normal Form

- Functional dependencies are the most common and important type of constraint in relational database design theory.
- However, there are situations in which the constraints that hold on a relation cannot be expressed as a functional dependency.
- Multi-valued dependencies are related to 1NF. Recall that 1NF simply means that all attribute values in a relation are atomic, which means that a tuple cannot have a set of values for some particular attribute.
- If we have a situation in which two or more multi-valued independent attributes appear in the same relation schema, then we'll need to repeat every value of one of the attributes with every value of the other attribute to keep the relation instance consistent and to maintain the independence among the attributes involved.
- Basically, whenever two independent 1:M relationships A:B and A:C occur in the same relation, a multi-valued dependency may occur.

Multi-valued Dependencies (cont.)

- Consider the following situation of a N1NF relation.

name	classes	vehicles
Mark	COP 4710 COP 3502	Mercedes E320 Ford F350
Kristy	COP 3330 CDA 3103 COT 4810	Mercedes E500 Porsche Carrera

Multi-valued Dependencies (cont.)

- Converting the N1NF relation to a 1NF relation.

name	classes	vehicles
Mark	COP 4710	Mercedes E320
Mark	COP 4710	Ford F350
Mark	COP 3502	Mercedes E320
Mark	COP 3502	Ford F350
Kristy	COP 3330	Mercedes E500
Kristy	CDA 3103	Mercedes E500
Kristy	COT 4810	Mercedes E500
Kristy	COP 3330	Porsche Carrera
Kristy	CDA 3103	Porsche Carrera
Kristy	COT 4810	Porsche Carrera