COP 4710: Database Systems
Spring 2006

Chapter 19 – Normalization – Part 3

Instructor : Mark Llewellyn
markl@cs.ucf.edu
CSB 242, 823-2790
http://www.cs.ucf.edu/courses/cop4710/spr2006

School of Electrical Engineering and Computer Science
University of Central Florida

Practice Problem Solution

Let $R = (C, S, Z)$

$F = \{CS \rightarrow Z, Z \rightarrow C\}$

$D = \{(SZ), (CZ)\}$

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

Test for each fd in F.

Test for $CS \rightarrow Z$

$$
\begin{aligned}
Z &= CS, \\
&= \{CS\} \cup ((CS \cap SZ)^+ \cap SZ) \\
&= \{CS\} \cup ((S)^+ \cap SZ) \\
&= \{CS\} \cup (S) \\
&= \mathbf{\{CS\}} \\
&= \{CS\} \cup ((CS \cap CZ)^+ \cap CZ) \\
&= \{CS\} \cup ((C)^+ \cap CZ) \\
&= \{CS\} \cup (C \cap CZ) \\
&= \{CS\} \cup (C) \\
&= \mathbf{\{CS\}} \quad \text{thus, } CS \rightarrow Z \text{ is not preserved.}
\end{aligned}
$$

Algorithm #1 for Producing a 3NF Decomposition

```
Algorithm 3NF.1
// input: a relation schema R= (A1, A2, …, An),   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 → Y in F do
           a = a +1;
           Ra = XY;
   endfor
   if [none of the schemes Rb (1 ≤ b ≤ a) contains a candidate key of R] then
           a = a + 1;
           Ra = any candidate key of R
   endif
   if [ ⋃_{b=1}^{a} Rb ≠ R ] then   //there are missing attributes

           Ra+1 = R − ⋃_{b=1}^{a} Rb
   return D = {R1, R2, ..., Ra+1}
end.
```

Example – Using Algorithm 3NF.1

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

K = {AB, AC}

F = {AB$\rightarrow$CDE, AC$\rightarrow$BDE, B$\rightarrow$C, C$\rightarrow$B, C$\rightarrow$D, B$\rightarrow$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 [$K \to Y \to A$ 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 $R = (A, B, C, D, E)$

$K = \{AB, AC\}$

$F = \{AB \rightarrow CDE, AC \rightarrow BDE, B \rightarrow C, C \rightarrow B, C \rightarrow D, B \rightarrow E\}$

Step 1: R not in 3NF since $AB \rightarrow C \rightarrow D$

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 $AB \rightarrow B \rightarrow E$

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

$AB \rightarrow CDE$: (1st time: equates nothing)
$AC \rightarrow BDE$: (1st time: equates nothing)
$B \rightarrow C$: (1st time: equates a_3 & b_{33})
$C \rightarrow B$: (1st time: equates a_2 & b_{12})
$C \rightarrow D$: (1st time: equates b_{14}, b_{24}, b_{34}) – stop second row becomes all a's
$B \rightarrow E$: (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 $R = (A, B, C, D, E)$

$F = \{AB \rightarrow CDE, AC \rightarrow BDE, B \rightarrow C, C \rightarrow B, C \rightarrow D, B \rightarrow E\}$

$D = \{(CD), (ABC), (BE)\}$

$G = F[CD] \cup F[ABC] \cup F[BE]$

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

Test for $AB \rightarrow CDE$

$$
\begin{aligned}
Z &= AB, \\
&= \{AB\} \cup ((AB \cap CD)^+ \cap CD) \\
&= \{AB\} \cup ((\varnothing)^+ \cap CD) \\
&= \{AB\} \cup (\varnothing \cap CD) \\
&= \{AB\} \cup (\varnothing) \\
&= \mathbf{\{AB\}} \\
&= \{AB\} \cup ((AB \cap ABC)^+ \cap ABC) \\
&= \{AB\} \cup ((AB)^+ \cap ABC) \\
&= \{AB\} \cup (ABCDE \cap ABC) \\
&= \{AB\} \cup (ABC) \\
&= \mathbf{\{ABC\}} \\
&= \{ABC\} \cup ((ABC \cap BE)^+ \cap BE) \\
&= \{ABC\} \cup ((B)^+ \cap BE) \\
&= \{ABC\} \cup (BCDE \cap BE) \\
&= \{ABC\} \cup (BE) \\
&= \mathbf{\{ABCE\}}
\end{aligned}
$$

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

Test for $AB \rightarrow CDE$ continues

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

Test for $AC \rightarrow BDE$

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

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 $C \rightarrow B$ 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 $C \rightarrow D$

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

Test for $B \rightarrow E$

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

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

Test for $B \rightarrow E$ continues

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

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

$Z = \{BC\}$
$= \{BC\} \cup ((BC \cap BE)^+ \cap BE)$
$= \{BC\} \cup ((B)^+ \cap BE)$
$= \{BC\} \cup (BCDE \cap BE)$
$= \{BC\} \cup (BE)$
$=$ **$\{BCE\}$ thus, $B \rightarrow E$ 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 $R = (A_1, A_2, \ldots, A_n)$, 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 Chapter 19 – Part 1, page 45).

3NF.3 (R, F)
find a minimal cover for F, call this cover G (see Chapter 19 - Part 1 page 45 for algorithm)
for each determinant X that appears in G do
create a relation schema $\{ X \cup A_1 \cup A_2 \cup \ldots \cup A_m \}$ 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 31 of Chapter 19 – Part 2 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 33 of Chapter 19 – Part 2, 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)

$F = \{A \rightarrow B, B \rightarrow C, C \rightarrow D, D \rightarrow A\}$

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

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

Projection onto schema (AB)

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

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

apply projection: $= \{AB\} \cup \{AB\} \cup \{AB\} =$ **{AB}**, **A→B is covered**

Projection onto schema (BC)

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

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

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

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

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

$$D^+_G = \{D, C, B, A\} \text{ Therefore, } G \vDash D \to A$$

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 4610	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 4610	Mercedes E320
Mark	COP 4610	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

Multi-valued Dependencies (cont.)

- Basically, a multi-valued dependency is an assertion that two attributes or sets of attributes are independent of one another.
- This is a generalization of the notion of a functional dependency, in the sense that every fd implies a corresponding multi-valued dependency.
- However, there are certain situations involving independence of attributes that cannot be explained as functional dependencies.
- There are situations in which a relational schema may be in BCNF, yet the relation exhibits a kind of redundancy that is not related to functional dependencies.

Multi-valued Dependencies (cont.)

- The most common source of redundancy in BCNF schemas is an attempt to put two or more M:M relationships in a single relation.

name	city	classes	vehicles
Mark	Orlando	COP 4710	Mercedes E320
Mark	Orlando	COP 4710	Ford F350
Mark	Orlando	COP 4610	Mercedes E320
Mark	Orlando	COP 4610	Ford F350
Kristy	Milan	COP 3502	Mercedes E500
Kristy	Milan	CDA 3103	Mercedes E500
Kristy	Milan	COT 4810	Mercedes E500
Kristy	Milan	COP 3502	Ford F350
Kristy	Milan	CDA 3103	Ford F350
Kristy	Milan	COT 4810	Ford F350

Multi-valued Dependencies (cont.)

- Focusing on the relation on the previous page, notice that there is no reason to associate a given class with a given vehicle and not another vehicle.

- To express the fact that classes and vehicles are independent properties of a person, we have each class appear with each class.

- Clearly, there is redundancy in this relation, but this relation does not violate BCNF. In fact there are no non-trivial functional dependencies at all in this schema.

- We know from our earlier discussions of normal forms based on functional dependencies that redundancies were removed, yet here is a schema in BCNF that clearly contains redundant information.

Multi-valued Dependencies (cont.)

- For example, in this relation, attribute city is not functionally determined by any of the other three attributes.
- Thus the fd: name class vehicle $\rightarrow$ city does not hold for this schema because we could have two persons with the same name, enrolled in the same class, and drive the same type of vehicle.
- You should verify that none of the four attributes in functionally determined by the other three. Which means that there are no non-trivial functional dependencies that hold on this relation schema.
- Thus, all four attributes form the only key and this means that the relation is in BCNF, yet clearly is redundant.

Multi-valued Dependencies (cont.)

- A multi-valued dependency (mvd) is a statement about some relation R that when you fix the values for one set of attributes, then the values in certain other attributes are independent of the values of all the other attributes in the relation.

- More precisely, we have the mvd

  $$A_1A_2...A_n \twoheadrightarrow B_1B_2...B_m$$

  holds for a relation R if when we restrict ourselves to the tuples of R that have particular values for each of the attributes among the A's, then the set of values we find among the B's is independent of the set of values we find among the attributes of R that are **not** among the A's or B's.

Multi-valued Dependencies (cont.)

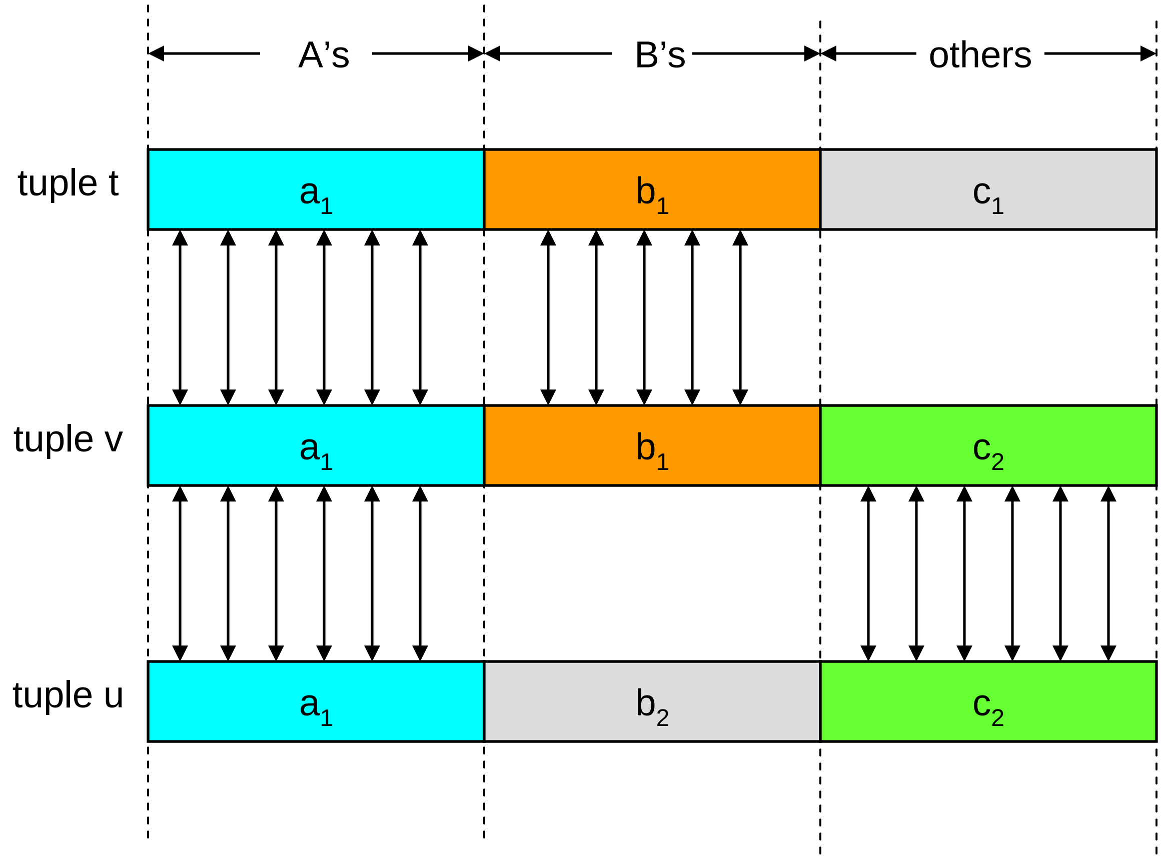
- Even more precisely, a mvd holds if:

  For each pair of tuples t and u of relation R that agree on all the A's, we can find in R some tuple v that agrees:

  1. With both t and u on the A's
  2. With t on the B's
  3. With u on all attributes of R that are not among the A's or B's.

  – Note that we can use this rule with t and u interchanged, to infer the existence of a fourth tuple w that agrees with u on the B's and with t on the other attributes. As a consequence, for any fixed values of the A's, the associated values of the B's and the other attributes appear in all possible combinations in different tuples.

Relationship of Tuple *v* to Tuple *t* When mvd Exists



A multi-valued dependency guarantees that tuple *V* exists

Multi-valued Dependencies (cont.)

- In general, we can assume that the A's and B's (left side and right side) of a mvd are disjoint.
- As with functional dependencies, it is permissible to add some of the A's to the right side.
- Unlike, functional dependencies where a set of attributes on the right side was a short-hand notation for a set of fds with single attribute right sides, with mvds, we must deal only with sets of attributes on the right side as it is not always possible to break the right side of mvds into single attributes.

Example: Multi-valued Dependencies

- Consider the following relation instance.

name	street	city	title	year
C. Fisher	123 Maple Street	Hollywood	Star Wars	1977
C. Fisher	5 Locust Lane	Malibu	Star Wars	1977
C. Fisher	123 Maple Street	Hollywood	Empire Strikes Back	1980
C. Fisher	5 Locust Lane	Malibu	Empire Strikes Back	1980
C. Fisher	123 Maple Street	Hollywood	Return of the Jedi	1983
C. Fisher	5 Locust Lane	Malibu	Return of the Jedi	1983

- The mvd name $\twoheadrightarrow$ street city holds on this relation.
  - That is, for each star's name, the set of addresses appears in conjunction with each of the star's movies.

Example: Multi-valued Dependencies (cont.)

- For an example of how the formal definition of this mvd applies, consider the first and fourth tuples from the previous relation instance.

name	street	city	title	year
C. Fisher	123 Maple Street	Hollywood	Star Wars	1977
C. Fisher	5 Locust Lane	Malibu	Empire Strikes Back	1980

- If we let the first tuple be t and the second tuple be u, then the mvd asserts that we must also find in R the tuple that has name C. Fisher, a street and city that agree with the first tuple, and other attributes (title and year) that agree with the second tuple. There is indeed such a tuple (the third tuple in the original instance).

name	street	city	title	year
C. Fisher	123 Maple Street	Hollywood	Empire Strikes Back	1980

Example: Multi-valued Dependencies (cont.)

- Similarly, we could let t be the second tuple below and u be the first tuple below (reversed from the previous page). Then the mvd tells us that there is a tuple of R that agrees with the second tuple in attributes name, street, and city with the first tuple in attributes name, title, and year.

name	street	city	title	year
C. Fisher	123 Maple Street	Hollywood	Star Wars	1977
C. Fisher	5 Locust Lane	Malibu	Empire Strikes Back	1980

- There is indeed such a tuple (the second tuple in the original instance).

name	street	city	title	year
C. Fisher	5 Locust Lane	Malibu	Star Wars	1977

Reasoning about Multi-valued Dependencies

- There are a number of inference rules that deal with mvds that are similar to the inference rules for functional dependencies.

1. Trivial multi-valued dependencies:

   If $A_1A_2...A_n \twoheadrightarrow B_1B_2...B_m$ holds for some relation, then so does $A_1A_2...A_n \twoheadrightarrow C_1C_2...C_k$ where the C's are the B's plus one or more of the A's.

   Conversely, we can also remove attributes from the B's if they are among the A's and infer the mvd $A_1A_2...A_n \twoheadrightarrow D_1D_2...D_r$ if the D's are those B's that are not among the A's.

Reasoning about Multi-valued Dependencies

2. Transitive rule for multi-valued dependencies:

   If $A_1A_2...A_n \twoheadrightarrow B_1B_2...B_m$ and $B_1B_2...B_m \twoheadrightarrow C_1C_2...C_k$ both hold for some relation, then so does $A_1A_2...A_n \twoheadrightarrow C_1C_2...C_k$. However, any C's that are also B's must be deleted from the right side.

- mvds do not obey the additivity/projectivity rules as do functional dependencies.

Reasoning about Multi-valued Dependencies

- Consider the same relation schema as before, where the mvd name $\twoheadrightarrow$ street city held. If the projectivity (splitting) rule held we would expect that

  name $\twoheadrightarrow$ street would also be true. This mvd states that each star's street addresses are independent of the other attributes (including city). However, that statement is false. The first two tuples in the relation instance indicate that this is not true.

name	street	city	title	year
C. Fisher	123 Maple Street	Hollywood	Star Wars	1977
C. Fisher	5 Locust Lane	Malibu	Star Wars	1977

Reasoning about Multi-valued Dependencies

- This hypothetical mvd name $\twoheadrightarrow$ street, if it held would allow us to infer that the tuples with the streets interchanged would be in the relation instance. However, these tuples are not there because the home at 5 Locust Lane is in Malibu and not Hollywood.

name	street	city	title	year
C. Fisher	5 Locust Lane	Hollywood	Star Wars	1977
C. Fisher	123 Maple Street	Malibu	Star Wars	1977

invalid tuples that cannot exist

Reasoning about Multi-valued Dependencies

- There are however, several new inference rules that apply only to multi-valued dependencies.
- First, every fd is a mvd. That is, if $A_1A_2...A_n \rightarrow B_1B_2...B_m$ holds for some relation, then so does $A_1A_2...A_n \twoheadrightarrow B_1B_2...B_m$ hold.
- Second, complementation has no fd counterpart. The complementation rule states: if $A_1A_2...A_n \twoheadrightarrow B_1B_2...B_m$ is a mvd that holds on some relation R, then R also satisfies $A_1A_2...A_n \twoheadrightarrow C_1C_2...C_k$, where the C's are all attributes of R that are not included in the A's or B's.
  - Thus, if name $\twoheadrightarrow$ street city holds, the complementation rule states that name $\twoheadrightarrow$ title year also holds, because street and city are not mentioned in the first mvd. The inferred mvd intuitively means that each star has a set of movies that they appeared in, which are independent of their address.

Fourth Normal Form

- The redundancy that we've seen in the relation instances in this section of the notes are caused by the existence of multi-valued dependencies.

- As we did with functional dependencies, we can use multi-valued dependencies and a different decomposition algorithm to produce a stronger normal form which is based not on functional dependencies but the multi-valued dependencies.

- Fourth Normal Form (4NF) eliminates all non-trivial multi-valued dependencies (as are all fds that violate BCNF). The resulting decomposition scheme has neither the redundancy from fds nor redundancy from mvds.

Fourth Normal Form (cont.)

- A mvd $A_1A_2...A_n \twoheadrightarrow B_1B_2...B_m$ for a relation scheme R is non-trivial if:

  1. None of the B's is among the A's.
  2. Not all of the attributes of R are among the A's and B's.

- 4NF is essentially the BCNF condition, but applied to mvds instead of fds.

- Formally, a relation scheme R is in 4NF if whenever $A_1A_2...A_n \twoheadrightarrow B_1B_2...B_m$ is a non-trivial mvd, $\{A_1A_2...A_n\}$ is a superkey of R.

Fourth Normal Form (cont.)

- The example relation scheme that we have been dealing with is not in 4NF because name ↠ street city is a non-trivial mvd, yet name by itself is not a superkey. In fact, for this relation the only key is all the attributes.

- 4NF is truly a generalization of BCNF. Since every fd is a mvd, every BCNF violation is also a 4NF violation. In other words, every relation scheme that is in 4NF is therefore in BCNF.

- However, there are some relation that are in BCNF but not in 4NF. The relation instance we have been using in this section of notes is a case in point. It is clearly in BCNF, yet as we just illustrated, it is not in 4NF.

Decomposition into Fourth Normal Form

- The 4NF decomposition algorithm is analogous to the 3NF and BCNF decomposition algorithm:
- Find a 4NF violation, say $A_1A_2...A_n \twoheadrightarrow B_1B_2...B_m$ where $\{A_1A_2...A_n\}$ is not a superkey. Note that this mvd could be a true mvd or it could be derived from the corresponding fd $A_1A_2...A_n \rightarrow B_1B_2...B_m$, since every fd is an mvd. Then break the schema for R into two schemas where: (1) the first schema contains all the A's and B's and the second schema contains the A's and all the attributes of R that are not among the A's or B's.

Decomposition into Fourth Normal Form (cont.)

- Using our previous example relation that we now know is not in 4NF, let's decompose into a relation schema that is in 4NF.

- We know that name $\twoheadrightarrow$ street city is a 4NF violation. The original schema R (5 attributes) will be replaced by one schema that contains only the three attributes from the mvd above, and a second schema that consists of the left side of the above mvd plus the attributes that do not appear in this mvd, which are the attributes title, and year.

  R1 = {name, street, city}

  R2 = {name, title, year}

Decomposition into Fourth Normal Form (cont.)

R1 = {name, street, city} R2 = {name, title, year}

- In each of these schema there are no non-trivial mvds or fds, so they are both in 4NF. Notice that in the relation scheme R1, the mvd name $\twoheadrightarrow$ street city is now trivial since it involves every attribute. Likewise, in R2, the mvd name $\twoheadrightarrow$ title year is also trivial.

Summary of Normal Forms

Property	3NF	BCNF	4NF
Eliminates redundancy due to functional dependencies	most	yes	yes
Eliminates redundancy due to multi-valued dependencies	no	no	yes
Preserves functional dependencies	yes	maybe	maybe
Preserves multi-valued dependencies	maybe	maybe	maybe
Has the lossless join property	yes	yes	yes