An Introduction to Database Systems

2nd Edition

Bipin C. DESAI

Concordia University
Montreal

BytePress

Limit of Liability/Disclaimer of Warranty:

The authors and the publishers have taken care to prepare this book. However, there is no warranty of the accuracy, completeness or presentation of the latest version/generation of any system discussed in this book. The reader must be aware of the fact that software systems often have multiple bugs and are not well thought out, and are usually suitable for limited situations and/or data combinations. Hence the user must be responsible for the appropriate application of any technique and use of any software or code examples.

Furthermore, there is no assurance whatsoever of the possible usefulness or commercialization of any programs, scripts and examples given in this book.

Any references given are based on their existence at the time of writing and the authors and the publishers do not endorse them or imply any usefulness of the information found therein. The reader must be aware that any web site cited may change, disappear or change their terms of service.

Published by: Electronic Publishing BytePress.com Inc.
Hardcopy - ISBN: 978-1-988392-15-8
Electronic - ISBN: 978-1-988392-08-0

7 Synthesis Approach and Higher Order Normal Form

The first, second, third and the Boyce Codd normal forms and algorithms for converting a relation in the first normal form into higher order normal forms were discussed in the previous chapter. In this chapter, we continue our discussions of the issues involved in the design of a database application using the relational model. In section 7.1, we examine the problems in the decoposition approach and present the synthesis approach to database design in section 7.2. We then turn our attention to the higher order normal forms. The concept of multvalued dependency and axioms which involve both functional dependencies and multivalued dependencies are examined. The fourth normal form and a lossless decomposition algorithm for it is given. The concept of join dependency and a normal form for it is introduced. Finally, we introduce a scheme whereby all general constraints could be enforced via domain and key constraint and the associated normal form, called domain key normal form.

7.1 Problems in the decomposition approach

Any relation can be decomposed into a number of relations which are in the third normal form. Such a decomposition is both lossless and preserve the dependencies. Any relation can also be decomposed losslessly into relations which are in the Boyce Codd normal form (and hence in the third normal form). However, the decomposition into the Boyce Codd normal form may not be dependence preserving. A point in case was illustrated in Example 6.37, where among others, the FD $M \rightarrow S_d$, is no longer represented in any of the decomposed relation schemes. It is not always possible to find a BCNF decomposition that is both lossless and dependence preserving. In addition, the decomposition into the BCNF is not unique. Many different BCNF relation schemes exist as illustrated in Example 6.37.

The decomposition approach using the BCNF decomposition algorithm may produce "inter-relational join constraints". This happens when the attributes **XY** corresponding to one of the function dependencies $\mathbf{X} \rightarrow \mathbf{Y}$ do not appear in any of the decomposed relation schemes. In the decomposed relation schemes of Example 6.37, in order to determine if the FD $M \rightarrow S_d$ is satisfied, we have to join the relations (*SMA*), (*SSd*) for the left decomposition of Figure G in Chapter 6. In general, to find out if a functional dependency $\mathbf{X} \rightarrow \mathbf{Y}$ is maintained in the decomposed schemes requires several of the decomposed relations to be joined. Since join operations are computationally expensive, inter-relational join constraints are undesirable.

However, a lossless and dependence preserving decomposition of a relation scheme into third normal form does not always give the minimum number of relation schemes. Furthermore, many different possible decompositions with the lossless and dependence preserving properties may be possible.

The goal of the decomposition approach to the relational database design using FD's is to come up with a database scheme which is in the BCNF, and which is lossless and preserves the original set of FD's. If this goal is not possible then the alternate goal is to derive a database scheme which is in the 3NF and is lossless and dependency preserving.

7.2 Normalization through Synthesis

In the decomposition approach to relational database design, we start with a relation (a universal relation) with undesirable properties and decompose it into a number of smaller relations to avoid these anomalies. The decomposition into the third normal form using Algorithm 6.6 will be both lossless and dependence preserving. The decomposition into the BCNF using the Algorithm 6.7 will be lossless but may not be dependence preserving. Furthermore the decomposition into the BCNF form is not unique.

The synthesis approach is an alternate approach to relational database design. Here we start with an universal relation scheme **U** which is not in the third normal form and a set of functional dependencies **F** over **U**, and we create a database scheme $\mathbf{R} = \{\mathbf{R_1}, \mathbf{R_2}, .., \mathbf{R_k}\}$. The scheme **R** is dependence preserving, i.e., all the dependencies in **F** are preserved, and, in particular, if there is functional dependency $f_i \in \mathbf{F}$, then there is a relation $\mathbf{R_i} \in \mathbf{R}$ such that the determinant of the FD f_i is a key of $\mathbf{R_i}$. Every relation $\mathbf{R_i}$ is, in addition, in the third normal form and there are no extraneous relations in the relation scheme **R** and hence no data duplications. In addition **R** is a lossless relation scheme if we ensure that at least one of the relations in **R** contains a key of **U**.

7.2.1 Functional Dependencies and Semantics

The functional dependencies are representations of the semantics of the real world data in its model. As a consequence of this, we have to be careful that the semantics of the functional dependencies are preserved. We saw the importance of distinct names for attributes to indicate their semantic usage in the universal relation approach earlier.

Consider the attribute price of the entity set ***PART***. Each part could have two prices associated with it, the wholesale or cost price and the retail or sale price. These price attributes are defined on the same domain. However, the wholesale and retail price are not synonymous and are distinguished by using distinct names such as *Price_Wholesale* and *Price_Retail*.

Let us consider another example where different meanings are attached to an attribute defined on a given domain. The following example of functional dependencies involves the attribute *Department* defined on the domain consisting of all the departments of an university. The attribute *Department* appears a number of times: *Student* → *Department*, *Course* → *Department*, *Professor* → *Department*. However, the semantics of the use of this domain for the attribute *Department* is to indicate the department to which the student, course or professor belongs and this could be distinct. This distinction is carried into the model by giving distinct names, let us say, *S_Department*, *C_Department*, and *P_Department* to these distinct meanings assigned to the attribute and write the above mentioned FD's as follows:

Student → *S_Department*
Course → *C_Department*
Professor → *P_Department*

7.2.2 Semantics of Nonfunctional Relationships

A nonfunctional relationship among attributes exists in a relation when some attributes are grouped together without any apparent dependencies existing between them. However, there is a relationship between these attributes which may become obvious if additional attributes are introduced. The FD's may not be apparent because the values for one set of attributes do not define unique values for another set of

attributes. In addition there may be no real functional dependency between these attributes but the database designer wants these attributes together. For example, the attributes *Professor*, *Interest*, *Course* could be grouped together, in the absence of apparent functional dependencies. However, this may be done to reflect the reality that a given professor has expertise and interest in a given area and that he or she can teach a given course which requires a knowledge in that area. Such nonfunctional dependencies can be introduced by using the following scheme:

Professor, *Interest*, *Course* → **Φ**

Here **Φ** is a nonexistent attribute used only to show the nonfunctional relationship among the attributes of it's determinant. To indicate additional nonfunctional relationships we can introduce additional nonexistent attributes $\Phi_1, \Phi_2, .. , \Phi_n$.

These nonexistent attributes can be used to define the nonfunctional relationship during the database design process and once a satisfactory database scheme is obtained these attributes can be discarded.

7.2.3 Synthesis Approach

Since the FD's determine whether or not a relation scheme is in the third normal form or not, it would be easy to obtain a relation scheme in the 3NF if the FD's are used to design the scheme. The synthesis approach uses the assumption that there is at least one functional relationship between two sets of attributes. If no such relationship in fact exists, the synthesis design approach introduces appropriate nonfunctional relationships. In the synthesis approach, the starting point of the relational database design process is an universal relation and the set of functional (and nonfunctional) dependencies that have to be enforced among the attributes of this universal relation. The synthesis procedure then synthesizes a set of third normal form relation schemes which preserves the required dependencies.

If the set of FD's used in the synthesis design process is a nonredundant cover, then the number of relations synthesized will be minimum. In fact it has been shown that the synthesis approach will produce the same set of relations regardless of the minimal cover that is used. (Recall that for a given set of FD's, it is possible to derive a number of covers).

Example 7.1: Consider the universal relation **U**(*A, B, C, D, E, H*) and the set of FD's **F** = {$A \rightarrow BC$, $CD \rightarrow E$, $E \rightarrow C$, $D \rightarrow AEH$, $ABH \rightarrow BD$, $DH \rightarrow BC$}. If a relation is synthesized for each FD in **F**, it will result in the following design:

$\mathbf{R_1}(ABC)$	with key	A
$\mathbf{R_2}(CDE)$	with key	CD
$\mathbf{R_3}(EC)$	with key	E
$\mathbf{R_4}(DAEH)$	with key	D
$\mathbf{R_5}(ABHD)$	with key	ABH
$\mathbf{R_6}(DHBC)$	with key	BC.

However, **F** contains redundant FD's $CD \rightarrow E$ and $DH \rightarrow BC$. This means that the relations $\mathbf{R_2}$ and $\mathbf{R_6}$ are redundant and can be eliminated from the design.

If the FD's used in the synthesis approach are left reduced, i.e., there are no extraneous attributes on the left hand side of the FD's, then we will not introduce any partial dependencies in the relations

synthesized using such FD's.

Example 7.2: Consider **U**{*A*,*B*,*C*,*D*} with the set of FD's **F** = { $ABC \rightarrow D, A \rightarrow C$}. The approach of using each FD in **F** to synthesize a relation gives the following relations:

$\mathbf{R_1}(ABCD)$ with key ABC,
$\mathbf{R_2}(AC)$ with key A.

However, the relation $\mathbf{R_1}$ is not in 3NF since there is a partial dependency $AB \rightarrow D$. If the FD $A \rightarrow C$ was used to left reduce $ABC \rightarrow D$, we will replace the latter by $AB \rightarrow D$ and hence obtain a synthesized design which is in the 3NF.

If two, or more, FD's have determinants which are functionally dependent on each other they are said to be **equivalent**. For instance if we have set of attributes **X** and **Y** and if $\mathbf{X} \rightarrow \mathbf{Y}$ and $\mathbf{Y} \rightarrow \mathbf{X}$ then **X** and **Y** are equivalent and this is written as $\mathbf{X} \leftrightarrow \mathbf{Y}$. In this case, instead of building two or more relations, one for each such FD, we can build only a single relation for each such group of FD's. Such a strategy will produce an economic relational design.

Example 7.3: Let us return to the universal relation **U**{*A*, *B*, *C*, *D, E*, *H*} and the set of FD's **F** = {$A \rightarrow BC$, $CD \rightarrow E$, $E \rightarrow C$, $D \rightarrow AEH$, $ABH \rightarrow BD$, $DH \rightarrow BC$}. We saw that the FD's $CD \rightarrow E$ and $DH \rightarrow BC$ are redundant and we can eliminate these. In addition, the FD $ABH \rightarrow BD$ is not left-reduced, the attribute *B* being extraneous. This gives us, after reduction, the FD's $AH \rightarrow D$. Now, since $D \rightarrow AH$, we get the one-to-one dependency $AH \leftrightarrow D$. Thus, *AH* and *D* are equivalent. We can combine these equivalent keys into one relation to give us the following synthesized relational design:

$\mathbf{R_1}\{ABC\}$ with key A
$\mathbf{R_2}\{EC\}$ with key E
$\mathbf{R_3}\{ADEH\}$ with keys AH,D

Having determined the groups of FD's which are equivalent, we should eliminate any transitive dependencies that may exist. This will ensure that the relations produced will be in 3NF.

7.2.4 Synthesis Algorithm

The best known synthesis algorithm was proposed by Bernstein [Bern76] and is sometimes called the Bernstein Synthesis algorithm. The algorithm, starts with an universal relation and the functional dependencies to be enforced thereon, and it produces a third normal form relation scheme that is lossless and dependence preserving. The algorithm is called a synthesis algorithm since it constructs relation schemes from the FD's rather than decomposing a relation scheme into simpler relation schemes.

The synthesis algorithm uses a canonical cover of a set of left-reduced functional dependencies and groups the functional dependencies such that the determinant of the FD's in each group is the same. Recall that a FD is left-reduced if the left hand side does not contain any extraneous attributes. It then finds compound functional dependencies $(\mathbf{X_1},\mathbf{X_2}, .., \mathbf{X_k}) \rightarrow \mathbf{Y}$ by using the equivalent determinant $\mathbf{X_i} \leftrightarrow \mathbf{X_j}$

for $1 \le i \le k$ and $1 \le j \le k$. Note: the characteristic of the compound functional dependency $(\mathbf{X_1}, \mathbf{X_2}, .., \mathbf{X_k}) \rightarrow \mathbf{Y}$ is that $\mathbf{X_i} \rightarrow \mathbf{X_j}$ and $\mathbf{X_i} \rightarrow \mathbf{Y}$ for $1 \le i \le k$ and $1 \le j \le k$.

Title: Algorithm 7.1: Synthesis Algorithm for Third Normal Form

Input: An universal database scheme **U**, a key **X** of **U**, and a set of simple left-reduced FD's **F**.

Output: A third normal form database scheme {**R**,**F'**}

Body:
1. (* Find a canonical cover*)
Find a canonical cover **G** of **F**. (* Use Algorithm 6.3 to first compute a non redundant cover *)

2. (* Form groups with same determinant *)
Partition **G** into groups $\mathbf{H_1}$, $\mathbf{H_2}$, ... such that all functional dependencies in each group have the same determinant.

3. (* Find and merge equivalent determinants *)
J := ϕ;(* J will contain the FD's between equivalent keys *)
Examine each pair of group $\mathbf{H_i}$, $\mathbf{H_j}$ with the determinant $\mathbf{X_i}$ and $\mathbf{X_j}$. If $\mathbf{X_i} \leftrightarrow \mathbf{X_j}$ i.e., if $\mathbf{X_i} \rightarrow \mathbf{X_j}$ and $\mathbf{X_j} \rightarrow \mathbf{X_i}$ are in $\mathbf{G}^+$ then

$\mathbf{J} := \mathbf{J} \cup \{ \mathbf{X_i} \rightarrow \mathbf{X_j}, \mathbf{X_j} \rightarrow \mathbf{X_i}\}$;
$\mathbf{H_i} := \mathbf{H_i} - \{ \mathbf{X_i} \rightarrow A \mid A \in \mathbf{X_j} \}$;
$\mathbf{H_j} := \mathbf{H_j} - \{ \mathbf{X_j} \rightarrow B \mid B \in \mathbf{X_i} \}$;

merge $\mathbf{H_i}$ and $\mathbf{H_j}$ into a single group

(* Remove those FD's in $\mathbf{H_i}$, $\mathbf{H_j}$ which pertain to the FD $\mathbf{X_i} \rightarrow \mathbf{X_j}$ and $\mathbf{X_j} \rightarrow \mathbf{X_i}$ respectively; thus basically we are modifying **G** as follows:

$\mathbf{G} := \mathbf{G} - (\mathbf{X_i} \rightarrow \mathbf{X_j}) - (\mathbf{X_j} \rightarrow \mathbf{X_i})$;

(* i.e., remove from **G** the FD's $\mathbf{X_i} \rightarrow \mathbf{X_j}$ and/or $\mathbf{X_j} \rightarrow \mathbf{X_i}$ if they are in **G** *)

4. (* Eliminate transitive dependencies *)
Find a minimum set of FD's $\mathbf{G_1}$ of **G** such that
$(\mathbf{G_1} \cup \mathbf{J})^+ = (\mathbf{G} \cup \mathbf{J})^+$
Here $\mathbf{G_1} \subset \mathbf{G}$.
$\mathbf{G_2} := \mathbf{G_1} \cup \mathbf{J}$;

5. Partition $\mathbf{G_2}$ into groups $\mathbf{H_1'}$, $\mathbf{H_2'}$, .. where each group has the same or equivalent determinant (* here use **J** to find equivalent pairs $\mathbf{X_i} \leftrightarrow \mathbf{X_j}$ *).

6. For each group $\mathbf{H_i'}$ with attributes ($\mathbf{X_i}$, $\mathbf{X_j}$,.., **Y**) corresponding to the FD's $\mathbf{X_i} \rightarrow \mathbf{X_j} \rightarrow .. \rightarrow \mathbf{X_1} \rightarrow \mathbf{Y}$ form a relation $\{\mathbf{X_iX_j} \mathbf{..Y}\}$ with key ($\mathbf{X_i}$ or $\mathbf{X_j}$ or ..) and add it to the relation scheme **R**.

7. (* Ensure that the relation scheme is lossless *)
If $\mathbf{K} \notin \mathbf{X_i}$ i.e., if a candidate key of **U** is not in one of the keys of the relations constructed, add the relation {**X**} to the relation scheme **R**.

Let us illustrate the synthesis algorithm via Examples 7.4 and 7.5.

Example 7.4: Consider the universal relation **U**(A, B, C, D, E, F, G) with the functional dependencies:

$BC \rightarrow A$
$FG \rightarrow BC$
$B \rightarrow D$
$C \rightarrow E$
$F \rightarrow A$
$G \rightarrow A$
$ABE \rightarrow G$
$ACD \rightarrow F$

In step 1 we find that the canonical cover of **F** includes the above FD's.

In step 2 we find that the groups contain one FD each.

In step 3 we discover that $BC \rightarrow FG$ and $FG \rightarrow BC$ are in the cover hence we can combine these two groups into a single group (BC, FG) $\rightarrow A$
G now becomes ($BC \rightarrow A$, $B \rightarrow D$, ..., $ACD \rightarrow F$)
J is $BC \rightarrow FG$, $FG \rightarrow BC$

In step 4 we find that the minimum cover of **G** $\cup$ **J** does not contain $BC \rightarrow A$.

In step 5 we partition the FD's into the following groups:(BC, FG) $\rightarrow \phi$, $B \rightarrow D$, .., $ACD \rightarrow F$.

In step 6 the relations obtained are:

($BCFG$)	with keys	(BC,FG)
(BD)	with key	(B)
(CE)	with key	(C)
(FA)	with key	(F)
(GA)	with key	(G)
(ABE)	with key	(ABE)
(ACD)	with key	(ACD).

Since the keys of **U** are BC or FG, and since it is contained in one of the relations above, the synthesis algorithm gives the final set of relations.

We now synthesize a set of 3NF relations for the Student_info relation presented in Example 6.32**.**

Example 7.5: Consider the set of attributes in the relation scheme **STUDENT_INFO** {$SMS_dACC_dGPP_dRDT$} with the following functional dependencies ($S \rightarrow MA$, $M \rightarrow S_d$, $A \rightarrow S_d$, $C \rightarrow C_dP$, $P \rightarrow P_d$, $RTD \rightarrow C$, $TPD \rightarrow R$, $TSD \rightarrow R$, $SC \rightarrow G$). The key of this relation is TSD.

In step 1 we find that the given set of FD's is minimal i.e., **G** is the given set of FD's.

In step 2 the groups created are ($S \rightarrow M$, $S \rightarrow A$), ($C \rightarrow C_d$, $C \rightarrow P$), ($M \rightarrow S_d$), ($A \rightarrow S_d$), ($P \rightarrow P_d$), ($SC \rightarrow G$), ($RTD \rightarrow C$), ($TPD \rightarrow R$), ($TSD \rightarrow R$).

In step 3 we find that $\mathbf{G}^+$ $RTD \leftrightarrow TPD$ and hence we get: **J** as being $RTD \rightarrow TPD$, $TPD \rightarrow RTD$ and **G** reduces to ($S \rightarrow M$, $S \rightarrow A$), ($C \rightarrow C_d$, $C \rightarrow P$), ($M \rightarrow S_d$), ($A \rightarrow S_d$), ($P \rightarrow P_d$), ($SC \rightarrow G$), ($RTD \rightarrow C$), ($TPD \rightarrow \Phi$), ($TSD \rightarrow R$).

In step 4 we eliminate $TPD \rightarrow \Phi$ to obtain $\mathbf{G_1}$ as being $S \rightarrow M$, $S \rightarrow A$, $C \rightarrow C_d$, $C \rightarrow P$, $M \rightarrow S_d$, $A \rightarrow S_d$, $P \rightarrow P_d$, $SC \rightarrow G$, $RTD \rightarrow C$, $TSD \rightarrow R$.

In step 5 we regroup ($S \rightarrow M$, $S \rightarrow A$), ($C \rightarrow C_d$, $C \rightarrow P$), ($M \rightarrow S_d$), ($A \rightarrow S_d$), ($P \rightarrow P_d$), ($SC \rightarrow G$), ($RTD \rightarrow C$, $RTD \rightarrow TPD$, $TPD \rightarrow RTD$), ($TSD \rightarrow R$).

In step 6 we get the following relation schemes:

Relation	Key
SMA)	(S)
(CC_dP)	(C)
(MS_d)	(M)
(AS_d)	(A)
(PP_d)	(P)
(SCG)	(SC)
($RTDPC$)	(RTD,TPD)
($TSDR$)	(TSD)

Since the relation contains the key *TSD* the final relation scheme is as above.

If we compare the relation schemes obtained with this approach with the ones obtained using the third normal form decomposition algorithm (Example 6.32), we find that the synthesis approach gives one less scheme. Basically here we have combined the FD's $RTD \rightarrow C$ and $TPD \rightarrow R$ into one relation scheme(*RTDPC*). This particular relation scheme is not in BCNF since for the FD $C \rightarrow P$ in this relation, the determinant *C* of the FD is not a key of the relation. However, the relation (*RTDPC*) is in 3NF.

7.3 Multivalued Dependency

We discussed multivalued dependency (MVD) earlier with respect to the employee entity, and the dependents, positions and salary history of the employee. Figure 7.1 below is an unnormalized relation showing the relation EMPLOYEE {*Employee_Name*, *Dependent*(*Name*, *Relationship*), *Position*(*Title*, *Date*),*Home_City*, *Home_Phone#*} and contains the information about employees. Each employee can have a number of dependents and would have occupied various positions in the organization, over time.

The relation has non-atomic values and, hence, it is not in normal form. We can normalize this relation, a normalized version of it being given in Figure 7.2. We see in Figure 7.1 that for a given value for *Employee_Name*, there are multiple values for the attributes (*Dependent_Name*, *Dependent_Relationship*) and (*Position_Title*, *Position_Date*).

The set of values for the attributes of (*Dependent_Name*, *Dependent_Relationship*) is not connected in any way to the values of the attributes in {EMPLOYEE - *Employee_Name* - *Dependent*}.

Similarly, the set of values for the attributes of (*Position_Title*, *Position_Date*) is not connected in any way to the values of the attributes in {EMPLOYEE - *Employee_Name* - *Positions*}.

Employee_ Name	*Dependent*		*Positions*		*Home_ City*	*Home_ Phone*
	Name	*Relationship*	*Title*	*Date*		
Jill Jones	Bill Jones	spouse	J.Engineer	05/12/84	Laval	794-2356
			Engineer	10/06/86		
	Bob Jones	son	J.Engineer	05/12/84		
			Engineer	10/06/86		
Mark Smith	Ann Briggs	spouse	Programmer	09/15/83	Montreal	452-4729
			Analyst	06/06/86		
	Chloe Smith-Briggs	daughter	Programmer	09/15/83		
			Analyst	06/06/86		
	Mark Briggs-Smith	son	Programmer	09/15/83		
			Analyst	09/06/86		

Figure 7.1 Unnormalized EMPLOYEE relation

Employee_ Name	*Dependent Name*	*Dependent_ Relationship*	*Position_ Title*	*Position _Date*	*Home_ City*	*Home_ Phone#*
Jill Jones	Bill Jones	spouse	J.Engineer	05/12/84	Laval	794-2356
Jill Jones	Bill Jones	spouse	Engineer	10/06/86	Laval	794-2356
Jill Jones	Bob Jones	son	J.Engineer	05/12/84	Laval	794-2356
Jill Jones	Bob Jones	son	Engineer	10/06/86	Laval	794-2356
Mark Smith	Ann Briggs	spouse	Programmer	09/15/83	Montreal	452-4729
Mark Smith	Ann Briggs	spouse	Analyst	06/06/86	Montreal	452-4729
Mark Smith	Chloe S-B	daughter	Programmer	09/15/83	Montreal	452-4729
Mark Smith	Chloe S-B	daughter	Analyst	06/06/86	Montreal	452-4729
Mark Smith	Mark B-S	son	Programmer	09/15/83	Montreal	452-4729
Mark Smith	Mark B-S	son	Analyst	06/06/86	Montreal	452-4729

Figure 7.2 Normalized EMPLOYEE Relation

For a second example of MVD, look at the SCHEDULE relation described in Chapter 6 and shown below, with some slight modifications, in Figure 7.3 We notice that a course is offered not only on a single day of the week but it may meet a number of times, and on each such meeting the room in which it meets may be different (not a frequent occurrence but nonetheless possible). Thus, the dependency

between a course and a day is not simply functional but it is multivalued. Similarly, the dependency between a course and the room in which it meets is multivalued.

These multivalued dependencies can be indicated as follows:

Course →→ *Room*,*Day*,*Time*

Prof	*Course*	*Room*	*Max_Enrollment*	*Day*	*Time*
Smith	353	A532	40	mon	1145
Smith	353	A534	40	wed	1245
Clark	355	H942	300	tue	115
Clark	355	H940	300	thu	115
Turner	456	B278	45	mon	845
Turner	456	B279	45	wed	845
Jamieson	459	D111	45	tue	1015
Jamieson	459	D110	45	thu	1015

Figure 7.3 The SCHEDULE relation

However a given course meets on a given day and time in but one room, i.e., there is a functional dependency:

Course, *Day*,*Time* → *Room*.

Multivalued dependencies arise when a relation R, having a nonatomic attribute is converted to a normalized form. Thus, for each **X** value in such a relation, there will be a set of **Y** values associated with it. This association between the **X** and **Y** values does not depend on the values of the other attributes in the relation. Suppose we have two tuples t_1, t_2 in relation R defined on the relation scheme **R** with the same **X** value; we exchange the **Y** values of these tuples and call the tuples so obtained t_3 and t_4. Then the tuples t_3 and t_4 must also be in R.

In the SCHEDULE relation of Figure 7.3, there is a multivalued dependency between *Course* →→ *Room*,*Day*,*Time*. Thus, if we exchange the {*Room*, *Day*, *Time*} value in the tuples t_1 and t_2 with the same Course value (353) where

t_1 = | Smith | 353 | A532 | 40 | mon | 1145 |

t_2 = | Smith | 353 | A534 | 40 | wed | 1245 |

we get the tuples t_3 and t_4 as follows:

t_3 = | Smith | 353 | A534 | 40 | wed | 1245 |

t_4 = | Smith | 353 | A532 | 40 | mon | 1145 |

The tuples t_3 and t_4 are in the database. (In fact in this example the tuple t_3 is the original tuple t_2 and the tuple t_4 is the original tuple t_1!).

The multivalued dependency *Course* →→ {*Room*, *Day*, *Time*} does not mean that the multivalued dependencies *Course* →→ *Room*, *Course* →→ *Day* and *Course* →→ *Time* will hold. Thus,

corresponding to the tuples t_1 and t_2 above if we exchange just the Room values we get $t_{3'}$ and $t_{4'}$ which are **not** in the database.

$t_{3'}$ = | Smith | 353 | A534 | 40 | mon | 1145 |

$t_{4'}$ = | Smith | 353 | A532 | 40 | wed | 1245 |

Using Figure 7.2 we can verify that such an exchange of the **Y** values for a multivalued dependency **X** →→ **Y** in two tuples t_1 and t_2 with the same **X** value will always give tuples t_3 and t_4 which are in the database, even if the relation has multiple multivalued dependencies. However, the tuples t_3 and t_4 need not be the original tuples t_1 and t_2. Exchanging the values of the attributes {*Dependent_Name*, *Dependent_Relationship*} in the two tuples t_1 and t_2 of Figure 7.3, gives us the tuples t_3 and t_4 as shown below. The tuples t_3 and t_4 are in the database, however these tuples are not the original t_1 and t_2 tuples.

t_1= | J J | Bill J | spouse | J.Eng | 05/12/84 | Laval | 794-2356 |

t_2= | J J | Bob J | son | Eng | 10/06/86 | Laval | 794-2356 |

t_3= | J J | Bill J | spouse | Eng | 10/06/86 | Laval | 794-2356 |

t_4= | J J | Bob J | son | J.Eng | 05/12/84 | Laval | 794-2356 |

This property of multivalued dependency can be expressed formally by the definition given below.

Definition: Given a relation scheme **R**, let **X** and **Y** be subsets of attributes of **R** (**X** and **Y** need not be distinct). Then the ***multivalued dependency*** **X** →→ **Y** holds in a relation R defined on **R** if given two tuples t_1 and t_2 in R with $t_1(\mathbf{X}) = t_2(\mathbf{X})$; then R contains two tuples t_3 and t_4 with the following characteristics:

-t_1, t_2, t_3, t_4 have the same **X** value i.e.,

$$t_1(\mathbf{X}) = t_2(\mathbf{X}) = t_3(\mathbf{X}) = t_4(\mathbf{X})$$

-the **Y** value of t_1 and t_3 are the same and similarly the **Y** value of t_2 and t_4 are the same i.e.,

$t_1(\mathbf{Y}) = t_3(\mathbf{Y})$ and $t_2(\mathbf{Y}) = t_4(\mathbf{Y})$

- the **R - X - Y** values of t_1 and t_4 are the same and similarly the **R - X - Y** values of t_2 and t_3 are the same i.e.,

$$t_1(\mathbf{R} - \mathbf{X} - \mathbf{Y}) = t_4(\mathbf{R} - \mathbf{X} - \mathbf{Y}) \text{ and}$$

$$t_2(\mathbf{R} - \mathbf{X} - \mathbf{Y}) = t_3(\mathbf{R} - \mathbf{X} - \mathbf{Y})$$

Let us examine the problems that are created as a result of multivalued dependencies. Consider Figure 7.2 for the normalized EMPLOYEE relation. It has two multivalued dependencies

Employee_Name →→ (*Dependent_Name*, *Dependent_Relationship*)

Employee_name →→ (*Position_Title*, *Position_Date*)

Suppose the employee Jill Jones gets a promotion on 12/15/86 to the position of Manager. This will involve adding two tuples to the database, one for each of the two dependents of this employee to correctly register the employment history of the employee. In addition, a change in the value of a FD in a relation involving a MVD requires the change to be reflected in all tuples corresponding to that entity. In our EMPLOYEE relation of Figure 7.2 a change of the home address of an employee would have to be reflected in all tuples pertaining to that employee. Thus, if Jill Jones moves to Westmount and if her home phone number changes to 368-4384, then in order to reflect this change correctly in the database, a change is required in not one tuple but six tuples (after the addition of the two tuples for an additional position). Deletion will require that more than one tuple be deleted. For example, in the SCHEDULE relation, if course 355 is canceled, then it will entail deleting two tuples from the table shown in Figure 7.3.

Summarizing, we note that in multivalued dependencies, the requirement on the database is that if there is a certain tuple in a relation, then for consistency the relation must have additional tuple(s) with similar values. Updates to the database affect these sets of tuples or entail the insertion of more than one tuple. Failure to perform these multiple updates will lead to inconsistencies in the database. To avoid these multiple updates, it is preferable to replace a relation having undesirable MVD's with a number of more 'desirable' relation schemes. We illustrate such more desirable schemes below in Figure 7.4 for the EMPLOYEE relation of Figure 7.2.[1] We discuss this approach further in the next sections. Such a scheme avoids the necessity of the multiple storage of the same information.

Employee_ Name	*Dependant_ Name*	*Dependent_ Relationship*
Jill Jones	Bill Jones	spouse
Jill Jones	Bob Jones	son
Mark Smith	Ann Briggs	spouse
Mark Smith	Chloe S-B	daughter
Mark Smith	Mark B-S	son

Employee_ Name	*Position_ Title*	*Position _Date*
Jill Jones	J.Engineer	05/12/84
Jill Jones	Engineer	10/06/86
Mark Smith	Programmer	09/15/83
Mark Smith	Analyst	06/06/86

Employee_ Name	*Home_ City*	*Home_ Phone#*
Jill Jones	Laval	794-2356
Mark Smith	Montreal	452-4729

Figure 7.4 Replacing the EMPLOYEE Relation with Three Relations

7.3.1 MVD and Normalization

In the normalization approach of a relation scheme with deletion, insertion, and update anomalies we have considered only functional dependencies so far. When the relation scheme to be normalized exhibits multivalued dependencies, we have to ensure that the resulting relation schemes do not exhibit any of

1Recall our discussions of separating a repeating group from the representation of an entity set and replacing each such group by an identifying relationship and a weak entity. These were then represented by a relation which contains the key of the strong entity along with the attributes of the weak entity.

these undesirable deletion, insertion and update anomalies. A normal form called the fourth normal form has been defined for relation schemes that have FD's as well as MVD's. The fourth normal form imposes constraints on the type of multivalued dependencies that are allowed in the relation scheme and is more restrictive than the BCNF.

The normalization of a relation scheme with MVD requires, as in the case of normalization of relations with only FD's, that the decomposed relation schemes are both lossless and dependence preserving. The following property of MVD will be used in the normalization approach.

Property of MVD

The following theorem for multivalued dependency is due to Fagin[Fagi77]. We simply state it here. For the proof, please see the bibliographic notes at the end of the chapter for the reference.

Theorem 7.1: If there is a multivalued dependency $\mathbf{X} \rightarrow\rightarrow \mathbf{Y}$ in a relation **R**, then it also has a MVD $\mathbf{X} \rightarrow\rightarrow \mathbf{R} - \mathbf{XY}$ and **R** can be decomposed losslessly into two relations $\mathbf{R_1}(\mathbf{X},\mathbf{Y})$ and $\mathbf{R_2}(\mathbf{X},\mathbf{Z})$ where $\mathbf{Z} = \mathbf{R} - \mathbf{XY}$.

As a consequence of the above, a relation scheme with a MVD must be able to be decomposed losslessly. Consider a relation scheme **R**. Let **X**, **Y**, **Z** be subsets of **R**, not necessarily disjoint, such that $\mathbf{Z} = \mathbf{R} - \mathbf{XY}$. Let R be a relation on the relation scheme **R**. Relation R satisfies the MVD $\mathbf{X} \rightarrow\rightarrow \mathbf{Y}$ if, and only if

$$R = \prod_{R1(XY)}(R) \bowtie \prod_{R2(XZ)}(R)$$

In other words, **R** decomposes losslessly into the relation scheme $\mathbf{R_1}$ and $\mathbf{R_2}$.

Definition: A *trivial multivalued dependency* is one which is satisfied by all relations R on a relation scheme **R** with $\mathbf{XY} \subseteq \mathbf{R}$. Thus, a MVD $\mathbf{X} \rightarrow\rightarrow \mathbf{Y}$ is *trivial* if $\mathbf{Y} \subseteq \mathbf{X}$ or $\mathbf{XY} = \mathbf{R}$. Obviously if $\mathbf{Y} = \phi$, then the MVD $\mathbf{X} \rightarrow\rightarrow \mathbf{Y}$ is trivial.

Example 7.6:

(a) In the normalized EMPLOYEE relation of Figure 7.2, with the following dependencies:

Employee_Name → *Home_City, Home_Phone#*,

Employee_Name →→ *Dependent_Name, Dependent_Relationship*,

Employee_Name →→ *Position_Title, Position_Date*.

The following MVD's are also satisfied:

Employee_Name →→ *Home_City, Home_Phone#*, *Dependent_Name, Dependent_Relationship*,

Employee_Name →→ *Home_City, Home_Phone#*, *Position_Title, Position_Date*.

(b) In Figure 7.4 the following MVD's are trivial:

Employee_Name →→ *Dependent_Name*, *Dependent_Relationship*
Employee_Name →→ *Position_Title*, *Position_Date*

7.3.2 Axioms for Functional and Multivalued Dependencies

In order to design a relational database, given a relation scheme **R** with functional and multivalued dependencies, we need a set of rules or axioms which will allow us to determine all the dependencies implied by a given set of known dependencies. Furthermore, we need these axioms to verify whether a given relation scheme is legal (from the point of view of being lossless and dependence preserving) under a set of functional and multivalued dependencies. The first three of these axioms are the same as the ones we discussed for functional dependencies. As before, **W**, **X**, **Y**, **Z** are subsets of **R**.

F1: Reflexivity: X → X.

F2: Augmentation: (**X → Y** and **W ⊆ Z)** ⊨ (**XZ → Y and** .**XZ** →WY)

F4: Additivity: (**X → Y** and **X → Z)** ⊨ **X → YZ**.

M1: **Replication**: **X → Y** ⊨ **X →→ Y**.

The replication axiom leads to the following versions of axioms **F1** through **F3** for multivalued dependencies:

M2: **Reflexivity**: **X →→ X**.

M3: **Augmentation**: **X →→ Y** ⊨ **XZ →→ Y**. If **X →→ Y** and **V ⊆ W** then **WX →→ VY**.

M4: **Additivity** or **Union**: **X →→ Y** and **X →→ Z** ⊨ **X →→ YZ**.

M5: **Complementation**: **X →→ Y** ⊨ **X →→ (R - X - Y)**

M6: **Transitivity**: {**X →→ Y** and **Y →→ Z**} ⊨ **X →→ (Z-Y)**

Note that unlike the transitivity rule for functional dependency, if **X →→ Y** and **Y →→ Z**, then it does not always imply that **X →→ Z** (i.e., **X →→ Z** could be false).

M7: **Coalescene**: Given that **W ⊆ Y** and **Y ∩ Z** = ϕ then if **X →→ Y** and **Z → W**, then **X → W**.

In addition to the above axioms which have been shown to be sound and complete (refer to the bibliographic notes for reference to the formal proofs), the following rules are useful.

M8: **Decomposition** or **Projectivity** for MVD: If **X →→ Y** and **X →→ Z**, then **X →→ (Y ∩ Z)**, **X →→ (Y-Z)**, and **X →→ (Z-Y)**.

The decomposition rule for functional dependencies is much stronger than the corresponding one for MVD; in the former, if **X → Y**, then **X →** A_i for $A_i \in$ **Y**. However, if **X →→ Y**, then we can only say that **X →→** A, if we can find a **Z** such that **X →→ Z** and **Y - Z** = A or **Z - Y** = A or **Y ∩ Z** = A.

M9: **Mixed (Pseudo)Transitivity**: If **X →→ Y** and **XY →→ Z** then **X →→ (Z-Y)**.

7.3.3 Closure Under MVD's

Given **D**, a set of FD's and MVD's, we can find a set of all functional and multivalued dependencies that can be derived from **D**. This set is the closure of **D** and to be consistent with the nomenclature for indicating the closure of a set of FD's is indicated by $\mathbf{D}^+$. The problem of computing the closure of **D**, like the problem of computing the closure of a set of FD, is a time consuming operation. However, instead of computing $\mathbf{D}^+$, we can use the axioms **M1** through **M9** to ascertain if a given MVD is implied by a set of FD and MVD. With this goal in mind, we develop a method to determine if $\mathbf{D} \models \mathbf{X} \twoheadrightarrow \mathbf{Y}$.

The Dependency Basis

Let **T** be a collection of sets closed under union, difference and intersection. T is closed if, t_1 and t_2 are in **T**, then, $t_1 \theta t_2$ is also in **T**. Here θ is one of the union, difference or intersection operations for sets. Each member of **T** is made up of a subcollection **S** of nonempty, pairwise disjoint sets: the collection **S** is called the **basis** of **T**.

Given **U** a set of attributes, $\mathbf{X} \subseteq \mathbf{U}$ and a set of dependencies **D**, then we want to find all subsets of **U** - **X** that are dependent on **X** by some MVD in $\mathbf{D}^+$. The complementation rule (**M5**), the union rule (**M4**), and the decomposition rule (**M8**) for multivalued dependencies imply that if the left hand side of a set of MVD is the same, then the right hand side is closed under boolean operation (i.e., for MVD's of the form $\mathbf{X} \twoheadrightarrow \mathbf{Y_i}$, $1 \leq i \leq n$, the $\mathbf{Y_i}$s are closed under boolean operation).

Thus, given $\mathbf{X} \subseteq \mathbf{U}$ and a set **D** of dependencies, we can derive a set $\mathbf{Y_i}$, $1 \leq i \leq n$, such that

- $\mathbf{U} - \mathbf{X} = \mathbf{Y_1 Y_2} \ldots \mathbf{Y_n}$,
- $\mathbf{Y_1}, \mathbf{Y_2}, \ldots \mathbf{Y_n}$ are **pairwise disjoint**, i.e., $\mathbf{Y_i} \cap \mathbf{Y_j} = \phi$ for $i \neq j$, and
- for any MVD $\mathbf{X} \twoheadrightarrow \mathbf{Z}$ in $\mathbf{D}^+$, **Z** is the union of some of the $\mathbf{Y_i}$s.

Definition: The set $\{\mathbf{Y_1}, \mathbf{Y_2}, \ldots \mathbf{Y_n}\}$, with the properties given above is referred to as the **dependency basis** of **X** with respect to **D** and is indicated by the nomenclature DEP(**X**).

An MVD $\mathbf{X} \twoheadrightarrow \mathbf{Z}$ is in $\mathbf{D}^+$ if and only if **Z** is a union of some of the sets from DEP(**X**), the dependency basis of **X** relative to the set **D** of FD's and MVD's. It follows that for each set $\mathbf{Y_i} \in$ DEP(**X**), $\mathbf{X} \twoheadrightarrow \mathbf{Y_i}$ is in $\mathbf{D}^+$.

The MVD $\mathbf{X} \twoheadrightarrow \mathbf{Y_i}$ where $\mathbf{Y_i} \in$ DEP(**X**) is called a **simple MVD**.

We see that DEP(**X**), the dependency basis of **X**, serves a similar function in determining if any MVD $\mathbf{X} \twoheadrightarrow \mathbf{Y}$ is implied by a set **D** of FD's and MVD's, as $\mathbf{X}^+$, the closure of a set of attribute under a set of FD, was used to determine if any FD $\mathbf{X} \rightarrow \mathbf{Y}$ was implied by a set of FD's **F**.

Algorithm 7.2 gives the method to compute the dependency basis of **X**. It simply converts each FD into an MVD, and then applies the rules of MVD to decompose the MVD's into simpler MVD's. A careful implementation of the algorithm can be shown to take time proportional to n^3m to complete,

where n is the number of attributes in **U** and m is the number of dependencies in **D**.

Title: Algorithm 7.2: Computing the dependency basis of X
Input: **U** a set of attributes, $\mathbf{X} \subset \mathbf{U}$ and **D** a set of FD's and MVD's.

Output: The dependency basis $\{\mathbf{Y_1}, \mathbf{Y_2}, .. , \mathbf{Y_n}\}$ of **X** under **D**.

Body:
Step 1:
Convert each FD $\mathbf{W} \rightarrow A$ to an MVD $\mathbf{W} \twoheadrightarrow A$ using rule **M1**.

Step 2:(* initialize the set **S** to the null set *)
$\mathbf{S} = \phi$;
Step 3: (* Apply rules **M3** and **M5** *)
For each MVD $\mathbf{W} \twoheadrightarrow \mathbf{Z}$ in **D** such that $\mathbf{W} \subseteq \mathbf{X}$ add $\mathbf{Z} - \mathbf{X}$ and $\mathbf{U} - \mathbf{Z} - \mathbf{X}$ to the set **S** as per rule **M3** and **M5**.

Step 4: (* Now apply the decomposition rule **M8** to each pair of set of attributes in set **S** such that they are not disjoint*)

For each pair of sets of attributes $\mathbf{Y_1}$ and $\mathbf{Y_2}$ in **S** such that $\mathbf{Y_1} \cap \mathbf{Y_2} \neq \phi$: replace $\mathbf{Y_1}$ and $\mathbf{Y_2}$ by the nonempty sets $\mathbf{Y_1} \cap \mathbf{Y_2}$, $\mathbf{Y_1} - \mathbf{Y_2}$, and $\mathbf{Y_2} - \mathbf{Y_1}$ (* i.e., discard the sets $\mathbf{Y_1} - \mathbf{Y_2}$ and $\mathbf{Y_2} - \mathbf{Y_1}$ if they are empty *).

Step 5: (* Now look for MVD $\mathbf{W} \twoheadrightarrow \mathbf{Z}$ in **D** and **Y** in **S** such that $\mathbf{Y} \cap \mathbf{W} = \phi$ but $\mathbf{Y} \cap \mathbf{Z} \neq \phi$ and $\mathbf{Y} \cap \mathbf{Z} \neq \phi$ and for such a MVD replace **Y** by $\mathbf{Y} - \mathbf{Z}$ and $\mathbf{Y} \cap \mathbf{Z}$.*)

For each MVD $\mathbf{W} \twoheadrightarrow \mathbf{Z} \in \mathbf{D}$ **and** ($\mathbf{Y} \in \mathbf{S}$)
and ($\mathbf{Y} \cap \mathbf{W} = \phi$)
and ($\mathbf{Y} \cap \mathbf{Z} \neq \phi$)
and ($\mathbf{Y} \cap \mathbf{Z} \neq \phi$)
replace **Y** in **S** by $\mathbf{Y} \cap \mathbf{Z}$ and $\mathbf{Y} - \mathbf{Z}$;

Step 6: (***S** now contains the dependency basis of **X** *)
Output **S**$\{\mathbf{Y_1}, \mathbf{Y_2}, .. , \mathbf{Y_n}\}$, the dependency basis of **X** under **D**.

Example 7.7 illustrates the use of Algorithm 7.2

Example 7.7: Consider a database to store student information which contains the following attributes: student's name(S) their majors(M), the department they are registered in (S_d), their adviser's name(A), the courses they are taking (C), the department which is responsible for the course(C_d), the final grade of the student in a course (G), the teacher of the course (P), the department of the teacher of the course (P_d), and the room, day and time (RDT) where the course is taught. Let us assume that the student's name and the adviser's names are unique. The database must satisfy the following set **H** of functional and multivalued dependencies:

$S \rightarrow MA$,
$M \rightarrow S_d$,
$A \rightarrow S_d$,
$C \rightarrow C_d P$,

$P \rightarrow P_d$,
$RTD \rightarrow C$,
$TPD \rightarrow R$,
$TSD \rightarrow R$,
$SC \rightarrow G$
$C \rightarrow\rightarrow RTD$
$C \rightarrow\rightarrow SMG$

We want to compute DEP(C) using the Algorithm 7.2. The first step will convert all FD's into MVD's.

Step 3 will give us the set **S** with the following sets:
$\{C_dP\}$, $\{RTD\}$, $\{SMG\}$, $\{SMAS_dP_dRTDG\}$, $\{SMAS_dP_dG\}$, $\{AS_dC_dPP_dRTD\}$.
Step 4 will split the sets in **S** to give the following sets in **S**
$\{C_dP\}$, $\{RTD\}$, $\{SMG\}$, $\{AS_dP_d\}$.
Step 5 will complete the intersections and splitting to give **S** with the following sets which is DEP(C), the dependency basis of C under the above set of FD's and MVD's:
$\{C_dP\}$, $\{RTD\}$, $\{SMG\}$, $\{S_d\}$, $\{A\}$, $\{P_d\}$.

The dependency basis allows us to conclude that the MVD's $C \rightarrow\rightarrow SS_dAMG$, $C \rightarrow\rightarrow PP_dC_d$ etc., are in $\mathbf{H}^+$, since the right hand side of each of the MVD's is an union of sets from DEP(C).

7.3.4 Fourth Normal Form

A generalization of the Boyce Codd normal form to relations schemes which includes the multivalued dependencies is called fourth normal form and is defined as follows:

Definition: Given a relation scheme **R** such that the set **D** of FD's and MVD's are satisfied. Consider a set of attributes **X** and **Y** where $\mathbf{X} \subseteq \mathbf{R}$, $\mathbf{Y} \subseteq \mathbf{R}$, $\mathbf{Y} \not\subset \mathbf{X}$, $\mathbf{Y} \neq \phi$, and $\mathbf{XY} \subseteq \mathbf{R}$. The relation scheme **R** is in the ***fourth normal form*** (4NF) if for all multivalued dependencies of the form $\mathbf{X} \rightarrow\rightarrow \mathbf{Y} \in \mathbf{D}^+$, either $\mathbf{X} \rightarrow\rightarrow \mathbf{Y}$ is a trivial MVD or **X** is a superkey of **R**. A database scheme is in the 4NF if all relation schemes included in the database scheme are in the 4NF.

If a relation scheme **R** with the set **D** of FD's and MVD's is in the fourth normal form, then, it is also in the BCNF. If this were not so, then **R** would satisfy a functional dependency not involving the superkey as a determinant of the form $\mathbf{X} \rightarrow \mathbf{Y}$. However, by the rule **M1** $\mathbf{X} \rightarrow \mathbf{Y} \models \mathbf{X} \rightarrow\rightarrow \mathbf{Y}$ and again **X** here is not a superkey; but this contradicts the assertion that **R** is in the fourth normal form.

7.3.5 Lossless Join Decomposition into Fourth Normal Form

Given a relation scheme that is not in the fourth normal form, we would like to decompose it into a set of relations that are in the fourth normal form and at the same time we want to preserve all the dependencies. Furthermore, we want the decomposition to be lossless. The latter requirement in the decomposition can be obtained using the property of MVD given in section 7.3.1 and restated in a

different form in the next paragraph. However, the first requirement, namely that of dependence preservation, is not as simple to satisfy (as in the case of having only FD's) when we have both functional and multivalued dependencies.

The following property of MVD can be used to perform a lossless decomposition of a relation **R** with both functional and multivalued dependencies. We are given a relation scheme, **R**, where **D** is a set of FD's and MVD's on the attributes of **R**. If **R** is decomposed into $\mathbf{R_1}$ and $\mathbf{R_2}$, then the decomposition is a lossless-join decomposition if and only if $\mathbf{D}^+$ contains one of the following MVD:

$(\mathbf{R_1} \cap \mathbf{R_2}) \rightarrow\!\!\!\rightarrow \mathbf{R_1}$ or $(\mathbf{R_1} \cap \mathbf{R_2}) \rightarrow\!\!\!\rightarrow \mathbf{R_2}$.

Recall that the requirement of a lossless-join decomposition, when only FD's are involved, was $(\mathbf{R_1} \cap \mathbf{R_2}) \rightarrow \mathbf{R_1}$ or $(\mathbf{R_1} \cap \mathbf{R_2}) \rightarrow \mathbf{R_2}$.

The similarity between the Boyce Codd normal form and the fourth normal form extends to the decomposition algorithm of a relation scheme not in the fourth normal form into a set of relations, which are in the fourth normal form. The adaptation of the decomposition algorithm for relation schemes with MVD is given in Algorithm 7.3.

Title: Algorithm 7.3: Lossless join decomposition into fourth normal form.

Input: A relation scheme **R** not in 4NF and a set of FD's and MVD's **D**.

Output: Decomposition of **R** into a set **S** of relation schemes $\mathbf{R_i}$, $\mathbf{R_i} \subset \mathbf{S}$ for $1 \leq i \leq n$ such that each $\mathbf{R_i}$ is in 4NF and the decomposition is lossless.

Body:

```
i := 0;
S := R0 (* initialize S to R0 ≡ R *);
for each nontrivial MVD ( X →→ Y ) that hold on some relation Rj in S, such that X is not a
superkey of Rj (* i.e., X → Rj is not in D+, we can further assume that X ⋂ Y = ϕ *) do
    begin
     i := i+1;
     Rj := Rj -Y;
            (* remove the attributes Y from Rj *)
     S := S ⋃ Ri{X,Y};
            (* form relation Ri{X, Y} and add it to S *)
     end;
end;
```

Let us return to the normalized EMPLOYEE relation of Figure 7.2. It has the following set of FD's and MVD's {*Employee_Name* →→ *Dependent_Name*, *Dependent_Relationship*, *Employee_Name* →→ *Position_Title*, *Position_Date*, *Employee_Name* → *Home_City*, *Home_Phone* }. Is this relation in the fourth normal form? It will be if the attribute *Employee_Name* is a superkey of the EMPLOYEE relation. We have used relations where the name, for convenience, was taken as an unique identifier for a person, the relation about student and faculty members being other such examples. If *Employee_Name* were the key of the EMPLOYEE relation, then according to the definition of the fourth normal form, the EMPLOYEE relation is in the fourth normal form. However, let us recall the definitions for key and superkey. A *superkey* of a relation R defined on a relation scheme **R** was defined as being a set of attributes $\mathbf{X} \subseteq \mathbf{R}$ such that, for two tuples t_1 and t_2 in R, $t_1(\mathbf{X}) \neq t_2(\mathbf{X})$. Thus, the values of the set of

attributes in **X** uniquely identify a tuple in R. A *key* is a set **K** such that no proper subset **K'** of **K** can uniquely identify a tuple of R, i.e., t_1(**K'**) may or may not be equal to t_2(**K'**).

With the above definition of superkey and key we see that the attribute *Employee_Name* is not a superkey of the relation EMPLOYEE and hence the relation is not in the fourth normal form. As a matter of fact the key of the EMPLOYEE relation is the entire relation! We note that even though *Employee_Name* is not a key of the relation, it still uniquely identifies an instance of the entity EMPLOYEE. All characteristics of an instance of the entity, let us say Jill Jones, are found by locating all tuples with this value for the *Employee_Name* attribute.

We noted the disadvantage in the form of anomalies in insertions, deletions, and updates for the EMPLOYEE relation as given in Figure 7.2. We can use the Algorithm 7.3 to decompose the EMPLOYEE relation losslessly into a set of fourth normal form relations. The resulting relations are given in Figure 7.5. (Note that these relations are the same as the ones shown in Figure 7.4.) The relation of Figure 7.5(a) and (b) have the trivial multivalued dependency **X** →→ **Y** with **R** = **XY**. In addition, both of these are all key relations. We note in passing that a non trivial MVD can be said to exist only if the relation has at least one attribute in addition to the two sets of attributes involved in the MVD.

Employee	*Dependant_ Name*	*Dependent_ Relationship*
Jill Jones	Bill Jones	spouse
Jill Jones	Bob Jones	son
Mark Smith	Ann Briggs	spouse
Mark Smith	Chloe S-B	daughter
Mark Smith	Mark B-S	son

(a)

Employee	*Position_ Title*	*Position_ Date*
Jill Jones	J.Engineer	05/12/84
Jill Jones	Engineer	10/06/86
Mark Smith	Programmer	09/15/86
Mark Smith	Analyst	06/06/86

(b)

Employee	*Home _City*	*Home_ Phone#*
Jill Jones	Laval	794-2356
Mark Smith	Revere,MA	452-4729

(c)

Figure 7.5 Decomposition of the EMPLOYEE Relation

7.3.6 Enforceability of Dependencies in the Fourth Normal Form

The fourth normal form decomposition algorithm produces a relation scheme which is lossless; however, it may not preserve all the dependencies in the original non-4NF relation scheme. In the following example, we use one MVD at a time to decompose a non-4NF relation scheme into two relation schemes. Then we determine if each of these schemes is in the 4NF. The following properties are used to find the dependencies that apply to the decomposed schemes.

Given **R** and the set of FD's and MVD's **D**, and let $\mathbf{R_1}$ be a projection of **R** i.e., $\mathbf{R_1} \subseteq \mathbf{R}$. The projection of **D** on $\mathbf{R_1}$ is derived as follows:

For each FD $\mathbf{X} \rightarrow \mathbf{Y}$ such that $\mathbf{D} \models \mathbf{X} \rightarrow \mathbf{Y}$, and if $\mathbf{X} \subseteq \mathbf{R_1}$, then $\mathbf{X} \rightarrow (\mathbf{Y} \cap \mathbf{R_1})$ hold in $\mathbf{R_1}$.

For each MVD $\mathbf{X} \twoheadrightarrow \mathbf{Y}$ such that $\mathbf{D} \models \mathbf{X} \twoheadrightarrow \mathbf{Y}$, and if $\mathbf{X} \subseteq \mathbf{R_1}$, then $\mathbf{X} \twoheadrightarrow (\mathbf{Y} \cap \mathbf{R_1})$ hold in $\mathbf{R_1}$.

The following example illustrates this method.

Example 7.8: Consider **R**(*A, B, C, D, E, F, G*) with the set **H** of FD's and MVD's given by H{ $A \twoheadrightarrow B$, $B \twoheadrightarrow G$, $B \twoheadrightarrow EF$, $CD \rightarrow E$}.

R is not in the 4NF since for the non-trivial MVD $A \twoheadrightarrow B$, *A* is not a superkey of **R**. We can take this MVD and decompose R into $\mathbf{R_1}(A,B)$, and **R**(*A,C,D,E,F,G*). $\mathbf{R_1}$ is in 4NF however, the reduced relation **R** is not in the 4NF.

Now the MVD's $A \twoheadrightarrow B$ and $B \twoheadrightarrow G$ gives by axiom **M6** $A \twoheadrightarrow G - B$ which is equivalent to $A \twoheadrightarrow G$. Using this MVD, we decompose **R** into $\mathbf{R_2}(A,G)$ and **R***(A,C,D,E,F)*. $\mathbf{R_2}$ is in 4NF however, the reduced relation **R** is still not in the 4NF.

We now take the MVD $CD \twoheadrightarrow E$ (after converting the FD into a MVD) and decompose **R** into $\mathbf{R_3}(C, D, E)$ and **R**(*A,C,D,F*).

The MVD's $A \twoheadrightarrow B$, $B \twoheadrightarrow EF$ by axiom **M6** gives $A \twoheadrightarrow EF - B$, which reduces to $A \twoheadrightarrow EF$ and when restricted to the current relation **R** gives $A \twoheadrightarrow F$. Decomposing **R** now gives $\mathbf{R_4}(A,F)$ and **R**(*A,C,D*).

R(*A,C,D*) is in the 4NF since $A \twoheadrightarrow B \models A \twoheadrightarrow CDEFG$ and its restriction to current relation **R** gives $A \twoheadrightarrow CD$.

However we notice that the dependency $B \twoheadrightarrow G$ is not preserved.

Example 7.8 illustrates that the 4NF decomposition is not dependence preserving. Thus if lossless as well as dependence preserving decomposition is required, we may have to settle for simple 3NF relation schemes, unless the BCNF decomposition is lossless as well as dependence preserving. An approach that could be used to derive a dependence preserving decomposition is to eliminate each redundant dependency in D.[2] This process can be repeated until only nonredundant dependencies remain in D. However, the order in which the dependencies are checked for redundancy determines the resulting nonredundant cover of D. It has been found that in this process, the MVD's should be eliminated before trying to eliminate FD's. The intuitive reason for this is that the FD's convey more semantics about the data than the MVD's.

2Elimination of redundant dependencies doesn't guarantee dependence preserving decomposition, in general. However, with conflict-free MVD's, the lossless decomposition is also dependence preserving. Conflict-free MVD sets are equivalent to an acyclic join dependency [Lien85,Scio81].

Dependence preserving decomposition, when **D**, a set of FD's and MVD's are involved, requires the derivation of the so called 4NF cover of **D**. No efficient algorithms exist to date to compute such a cover. The algorithm to decompose a relation into a lossless and dependency preserving 4NF relation is beyond the scope of this text. Interested readers are referred to the references in the bibliographic notes. Attempts have been made to find a synthesis algorithm to construct a relation scheme from a set of FD's and MVD's. Here again, no satisfactory algorithm has emerged.

7.4 Normalization Using Join Dependency - Fifth Normal Form

A criterion of good database design is to reduce the data redundancy as much as possible and one way of doing this in a relational database design is to decompose one relation into multiple relations. However, the decomposition should be lossless and preferably maintain the dependencies of the original scheme. A relational database design is, as such, a compromise between the universal relation and a set of relations with desirable properties. The relational database design thus tries to find relations satisfying as high a normal form as possible. For instance, the 3NF is preferable to the 2NF, the BCNF is preferable to the 3NF, and so on.

However, recent research in the relational database design theory has discovered higher and higher, and, hence, more desirable, normal forms. The **Fifth normal form(5NF)** is a case in point. It is related to what is called ***join dependency***, which is the term used to indicate the property of a relation scheme that cannot be decomposed losslessly into two simpler relation schemes, but can be decomposed losslessly into three or more simpler relation schemes.

To understand join dependency, let us use the following dependencies from the database for an enterprise involved in developing computing products. It employs a number of employees and has a variety of projects.

Project →→ *Expertise*
(i.e., expertise needed for a given project)
Employee →→ *Expertise*
(i.e., expertise of the employee)
Employee →→ *Project*
(i.e., preferences of the employees to match their expertise)

These dependencies are the translation of the enterprise's need, that the employees involved in a given project must have certain expertise. Due to the expertise of employees and their natural tendency, they want to be involved in a given set of projects whose requirements match their interests etc. Let us look at the relation scheme **PROJECT_ASSIGNMENT***(Employee*, *Project*, *Expertise*). A relation defined on this scheme is given in Figure 7.6

Employee	*Project*	*Expertise*
Smith	Query System	Data Base Systems
Smith	File System	Operating Systems
Lalonde	Database Machine	Computer Architecture
Lalonde	Database Machine	VLSI Technology
Evan	Database Machine	VLSI Technology
Evan	Database Machine	Computer Architecture

Drew	SQL++	Relational Calculus
Drew	QUEL++	Relational Calculus
Shah	SQL++	Relational Calculus
Shah	QUEL++	Relational Calculus

Figure 7.6 PROJECT_ASSIGNMENT Relation

The relation scheme stores the employee's assignments based on the needs of the project, as well as the qualification and preferences of the employee who can contribute to the project. A project may demand more than one type of expertise, and an employee may be an expert in more than one area. A project that involves more than one type of expertise requires an employee with these capabilities. Thus, we notice that the project Query Systems needs only the expertise of Database Systems, while a project Database Machine needs the expertise of VLSI Technology, as well as Computer Architecture. Further expertise of an employee, not needed for any project to which he or she is assigned, is not shown in this relation.

Further expertise of an employee, not needed for any project to which he or she is assigned, is not shown in this relation. Figure 7.6 illustrates the sample contents of a database defined on this relation scheme. Here the employees Lalonde and Evan are assigned to the project Database Machine, while employees Drew and Shah are assigned to projects SQL++ and QUEL++. The relation, as shown, exhibits the following non-trivial multivalued dependencies: *Project* →→ *Expertise* and *Project* →→ *Employee*. Note further that the MVD *Employee* →→ *Project* and, hence, *Employee* →→ *Expertise* are not exhibited in this relation. This can be verified by exchanging the *Project* value for Smith, whereby we find that the resulting tuples are not in the database.

The relation PROJECT_ASSIGNMENT{*Employee*, *Project*, *Expertise*} having the MVD *Project* →→ *Expertise* (and by axiom **M5** *Project* →→ *Employee*) can be decomposed, losslessly, into relations PROJECT_REQUIREMENT{*Project*, *Expertise*} and PROJECT_PREFERENCE{*Employee*, *Project*}. Figure 7.7 shows the decomposition of the relation of Figure 7.6. The join of PROJECT_REQUIREMENT and PROJECT_PREFERENCE gives the same data as that in Figure 7.6.

Notice from Figure 7.7(b) that the relation PROJECT_PREFERENCE exhibits the (trivial) multivalued dependency *Employee* →→ *Project*. Such a multivalued dependency that is not exhibited in a relation but becomes evident in a projection of the relation is called ***embedded multivalued dependency***. Unlike multivalued dependencies, functional dependencies are never embedded. A functional dependency **X** → **Y**, that is evident in a projection of relation R is also evident in the relation R.

Project	Expertise
Query System	Data base Systems
File System	Operating Systems
Database Machine	Computer Architecture
Database Machine	VLSI Technology
SQL++	Relational Calculus
QUEL++	Relational Calculus

(a) PROJECT_REQUIREMENT

Employee	Project
Smith	Query System
Smith	File System
Evan	Database Machine
Lalonde	Database Machine
Drew	SQL++
Shah	QUEL++
Drew	SQL++
Shah	QUEL++

(b) PROJECT_PREFERENCE

Figure 7.7 Lossless Decomposition of Relation of Figure 7.6

Consider a relation scheme **R** and let **X**, **Y**, and **Z** be sets of attributes of **R**. Here **X**, **Y**, **Z** need not be disjoint. A relation R over the relation scheme **R** satisfies the embedded multivalued dependency **X** →→

Y | Z (i.e., **R** satisfies **X →→ Y** and hence, by axiom **M5**, **X →→ Z**), if the projection of the relation R over **X**, **Y**, **Z** (i.e., $\prod_{XUYUZ}(\mathbf{R})$) satisfies the MVD's **X →→ Y and X →→ Z**.

Now consider the relation scheme **NEW_PROJECT_ASSIGNEMNT**. Perhaps after some modifications in the enterprise involved, there has been a turnover in employees and the expertise of new employees requires some changes in the assignment of projects. Figure 7.8 gives a sample table for a relation defined on the scheme **NEW_PROJECT_ASSIGNEMNT**. As the figure indicates, we are assigning more than one employee to a given project. Each employee is assigned a specific role in this project, requiring knowledge that lies within her or his field of expertise. Thus, project Work Station, which requires expertise of User Interface, Artificial Intelligence, VLSI Technology, and Operating Systems, can be carried out by Brent, Mann and Smith combined. Brent is assigned the User Interface and Artificial Intelligence related role, Mann is assigned the VLSI Technology related role, while Smith is assigned the Operating Systems parts. This flexibility was not exhibited in the data of Figure 7.6.

The relation of Figure 7.8 does not show any functional or multivalued dependencies; as a matter fact it is an all key relation, and therefore in the fourth normal form. Unlike the relation **PROJECT_ASSIGNEMNT**, the relation **NEW_PROJECT_ASSIGNEMNT** cannot be decomposed losslessly into two relations. However, it can be decomposed losslessly into three relations. This decomposition is shown in Figure 7.9. Two of these relations, when joined, creates a relation that contains extraneous tuples and, hence, the corresponding decomposition is not lossless. These superfluous tuples are removed when the resulting relation is joined with the third relation. Note, that the MVD's, similar to those exhibited in Figure 7.6, are embedded in this example.

Employee	*Project*	*Expertise*
Brent	Work Station	User Interface
Brent	Work Station	Artificial Intelligence
Mann	Work Station	VLSI Technology
Smith	Work Station	Operating Systems
King	SQL 2	Relational Calculus
Ito	SQL 2	Relational Algebra
Ito	QBE++	Relational Calculus
Smith	Query System	Database Systems
Smith	File System	Operating Systems

Figure 7.8 NEW_PROJECT_ASSIGNEMNT Relation

Project	*Expertise*
Work Station	User Interface
Work Station	Artificial Intelligence
Work Station	VLSI Technology
Work Station	Operating Systems
SQL 2	Relational Calculus
SQL 2	Relational Algebra
QBE++	Relational Calculus
Query System	Database Systems
File System	Operating Systems

(a)

Employee	*Expertise*
Brent	User Interface
Brent	Artificial Intelligence

Employee	*Project*
Brent	Work Station
Mann	Work Station

Mann	VLSI Technology
King	Relational Calculus
Ito	Relational Algebra
Ito	Relational Calculus
Smith	Database Systems
Smith	Operating Systems

(b)

King	SQL 2
Ito	SQL 2
Ito	QBE++
Smith	File System
Smith	Query System
Smith	Work Station

(c)

Figure 7.9 Decomposition of Relation of Figure 7.8

7.4.1 Join Dependencies

So far we have focused on the decomposition of a relation scheme with undesirable properties into two relation schemes (at each step of a multistep process), such that the decomposition is lossless. A join of these decomposed relation schemes will give the original scheme and, hence, the data. However, as we saw in the previous section, it is not possible to find a lossless decomposition of a relation scheme into two relation schemes, but the same relation scheme can be decomposed losslessly into three relation schemes. This property is referred to as the join dependency.

Definition: Given a relation scheme **R** and let us consider the following set of its projections: {$\mathbf{R_1}$, $\mathbf{R_2}$, .. $\mathbf{R_n}$}. A relation R(**R**) satisfies the **join dependency** *[$\mathbf{R_1}$,$\mathbf{R_2}$, ... $\mathbf{R_n}$], if and only if, the join of the projection of R on $\mathbf{R_i}$ is equal to R.

$R = \prod_{R1}(R) \bowtie \prod_{R2}(R) \bowtie ... \bowtie \prod_{Rn}(R)$

In other words, join dependency is the assertion that the decomposition of R onto $\mathbf{R_1}$,...,$\mathbf{R_n}$ is a lossless decomposition. A join dependency is ***trivial*** if one of the projections of **R** is **R** itself.

A necessary condition for a relation scheme **R** to satisfy a join dependency *[$\mathbf{R_1}$, $\mathbf{R_2}$, .. $\mathbf{R_n}$] is that **R** = $\mathbf{R_1} \cup \mathbf{R_2} \cup \quad \cup \mathbf{R_n}$.

The relation scheme **PROJECT_ASSIGNMENT** satisfies the join dependency *[**PROJECT_REQUIREMENT**, **PROJECT_PREFERENCE**], since the join of PROJECT_REQUIREMENT, and PROJECT_PREFERENCE gives the relation PROJECT_ASSIGNMENT losslessly. However, the relation NEW_PROJECT_ASSIGNMENT does not satisfy any of the following join dependencies:

*[(*Project*,*Expertise*),(*Employee*,*Expertise*)],
*[(*Project*,*Expertise*),(*Employee*, *Project*)],
*[(*Employee*, *Expertise*),(*Employee*, *Project*)]

Relation NEW_PROJECT_ASSIGNMENT, however, satisfies the join dependency:

*[(*Project*,*Expertise*), (*Employee*,*Expertise*), (*Employee*, *Project*)].

Since the relation scheme **NEW_PROJECT_ASSIGNMENT** does not satisfy any non-trivial MVD, then by the theorem of Fagin (Theorem 7.1), it cannot be decomposed losslessly into two relations

It is worthwhile pointing out that every MVD is equivalent to a join dependency, however, the

converse is not true, i.e., there are join dependencies that are not equivalent to any non-trivial MVD's. The first part of this statement can be confirmed as follows: the relation R(**R**) satisfies the MVD $\mathbf{X} \rightarrow\rightarrow \mathbf{Y}$ if and only if the decomposition of R into **XY** and **R - Y** is lossless. This is equivalent to saying that R(**R**) satisfies the JD *[**XY**, **R - Y**]. Conversely, **R** satisfies the JD *[$\mathbf{R_1}$, $\mathbf{R_2}$] if $\mathbf{R_1} \cap \mathbf{R_2} \rightarrow\rightarrow \mathbf{R_1}$, or $\mathbf{R_1} \cap \mathbf{R_2} \rightarrow\rightarrow \mathbf{R_2}$. However, not all JD's are equivalent to MVD, as seen in the example of Figures 7.7 and 7.9.

A join dependency on the relation scheme **R**, in addition to those for MVD's, could also be a result of key dependencies. This can occur when the decomposition of a relation involves a superkey and the relation can be reconstructed by joins, every join involving a superkey. Thus, if **R**($\mathbf{X_1}$, $\mathbf{X_2}$, ..., $\mathbf{X_m}$) and if $\mathbf{X_i}$s are the superkeys of **R**, then the join dependency ***[$\mathbf{X_1}$, $\mathbf{X_2}$, ..., $\mathbf{X_m}$]**, is due to the keys of R.

Join dependency expresses the fact that a set of relationships is independent, just as MVD indicates that a pair of relationships is independent. These independent relationships can be separated in different relations and their join will be lossless. The join dependency in a relation scheme gives rise to another normal form, called ***project-join normal form***, discussed in the following section.

7.4.2 Project-join Normal Form

Consider a relation scheme **R**(**U**) and a set of FD's {$\mathbf{S_1} \rightarrow \mathbf{U}$, $\mathbf{S_2} \rightarrow \mathbf{U}$, ... $\mathbf{S_p} \rightarrow \mathbf{U}$}. We name these FD's **Key Dependencies** or KDs since the determinant, $\mathbf{S_i}$ in each FD, is a superkey. Let us present the JD membership algorithm to determine if a JD is implied by a set of KDs. The algorithm terminates successfully if and only if the KDs $\models$ JD.

Example 7.9 determines the JDs implied by a given set of KDs.

Example 7.9: Let **R**(ABCDE) with the FD's F = { A → BCDE, C → ABDE and D → ABCE}. Let **R** satisfy the join dependencies *[ABE, CD, ABCD]. The FD's are KDs and we see that for the superkey(key) A, A ⊆ ABE ∩ ABCD. Hence, we replace the set {ABE, CD, ABCD} with the set {ABCDE, CD}. Again we find that for the superkey(key) C, C ⊆ ABCDE ∩ CD. Hence, we replace the set {ABCDE, CD} with the set {ABCDE}. Since this is the set of attributes in R we have shown that KD $\models$ JD . It can similarly shown that the KD implies the following JD *[ABC,BCD, CDE].

We can now define the project join normal form.

Definition: Consider a relation scheme **R** and a set **D** of dependencies (functional, multivalued and join). The relation R is in the ***project-join normal form*** with respect to **D** if for every join dependency *[$\mathbf{R_1}$, $\mathbf{R_2}$, ..., $\mathbf{R_n}$] that is applicable to R and is implied by **D**, either of the following two holds:

- the join dependency is trivial, or

- every $\mathbf{R_i}$ is a superkey of **R**.

A database is in the project join normal form if all relations schemes are in the project join normal form.

Project-join normal form is also referred to as the fifth normal form (5NF) or as PJ/NF in the database literature. Every fifth normal form relation scheme is also in the fourth normal form and, hence, in the

BCNF and consequently in the 3NF.

Title: **Algorithm 7.4: JD Membership Algorithm**
Input: JD[$\mathbf{X}_1$, $\mathbf{X}_2$, $\mathbf{X}_3$, . .. $\mathbf{X}_q$] and KD{$\mathbf{S}_1 \rightarrow \mathbf{U}$, $\mathbf{S}_2 \rightarrow \mathbf{U}$, ... $\mathbf{S}_p \rightarrow \mathbf{U}$}
Output: Success or Failure. Success indicates KD $\models$ JD.
Body
H = {$\mathbf{X}_1$, $\mathbf{X}_2$, $\mathbf{X}_3$, . .. $\mathbf{X}_q$} (* initialize set **H** to be the JD
to be checked *)
change := true
while (change or number of members q in **H** > 1) do
begin
if $\mathbf{S}_i \subseteq \mathbf{X}_j \cap \mathbf{X}_k$ for $1 \leq i \leq p$ and $\mathbf{X}_j$ and $\mathbf{X}_k \in \mathbf{H}$ and $j \neq k$
then begin
delete $\mathbf{X}_j$ and $\mathbf{X}_k$ from **H**
insert $\mathbf{X}_j \cup \mathbf{X}_k$ into **H**
decrease the q by 1
else change := false
end
if $\mathbf{U} \in \mathbf{H}$
then KD $\models$ JD is proven successfully
else KD $\models$ JD is not proven

If a relation is in the project join normal form, then every functional dependency is determined by a key. Every multivalued dependency is also determined by a key. Furthermore, every JD is determined by one or more candidate keys. As a result, since all FD's, MVD's, and JD's are implied by keys, all that is required to be specified is the relation scheme and the set of keys. A database having all relations in the PJ/NF and supporting the concept of key need no other consistency support mechanism, if there are no inter-relational dependencies. However, when we convert a relation that is not in the PJ/NF into a set of relations in the PJ/NF, we could introduce inter-relational dependencies.

Our example relation schemes **PROJECT_ASSIGNMENT** and **NEW_PROJECT_ASSIGN MENT** were not in the fifth normal form, since each of them had non-trivial join dependencies. Their decompositions, (respectively into {**PROJECT_REQUIREMENT**, **PROJECT_PREFERENCE**}, and {*(Project*, *Expertise*), *(Employee*, *Expertise*), *(Employee*, *Project*)}) are in the fifth normal form, nonetheless.

Let us return to the **NEW_PROJECT_ASSIGNMENT** relation scheme. Here, we have three independent relationships:

Project $\rightarrow\!\!\rightarrow$ *Expertise*
Employee $\rightarrow\!\!\rightarrow$ *Expertise*
Employee $\rightarrow\!\!\rightarrow$ *Project*

There are other MVD relationships, for instance *Project* $\rightarrow\!\!\rightarrow$ *Employee* that can be derived from the MVD *Employee* $\rightarrow\!\!\rightarrow$ *Project*.

It is not possible to insert, into NEW_PROJECT_ASSIGNEMNT relation without null values, a project and the expertise needed for it unless we know the employees that could be assigned to the project. Similarly, it is not possible to record all types of expertise of an employee unless each of it is called for in a project where that employee is required to use such expertise. The decomposition of the relation into {*(Project*, *Expertise*), *(Employee*, *Expertise*), *(Employee*, *Project*)} allows these independent

relationships to be separated. It is then possible to independently maintain each separate relation. On the other hand, in the relation NEW_PROJECT_ASSIGNMENT, it is necessary to insert additional tuples when a tuple is inserted and the deletion of a tuple requires the deletion of other tuples.

Consider the relation STUDENT_INFO(*Name*, *Address*, *Department*, *Phone#*) with the FD's {*Name* → *Address*, *Name* → *Department*, *Name* → *Phone#*). The decomposition of STUDENT_INFO into the following relation is lossless and dependence preserving. (*Name*, *Address*), (*Name*, *Department*), (*Name*, *Phone#*). The reader will have noted that the original relation is in PJ/NF. However, since the only key of the original relation is *Name*, and if the remaining attributes could have null values assigned to them, then there is no advantage to decomposing the relation.

7.6 Domain/Key Normal Form

Before discussing the Domain Key/Normal Form let us define two additional type of dependencies, called domain constraints (DC) and key constraints (KC).

Definition - Domain Constraint: Each attribute A_i of a relation scheme **R**(A_1, A_2, A_i, ...), is assigned a domain constraint of the form IN(A_i,S_{Ai}). This simply means that the attribute A_i of relation R, that is defined on the relation scheme **R**, must have a value from the set S_{Ai}.

We have, implicitly, used domain constraint as part of integrity constraints.

Definition - Key Constraint: For the relation scheme **R**(A_1, A_2, A_i, ...), the key constraint, KEY(**K**), where **K** is a subset of **R**, is the restriction that no two tuples of relation R, defined on the relation scheme **R**, have the same values for the attributes in **K**.

We also define the concept of general constraints (GC).

Definition - General Constraints: A general constraint is expressed as a simple statement or predicate and specifies some special requirement. Each tuple of a relation must satisfy this predicate in order for it to be a valid tuple.

The Domain Key Normal Form (DK/NF) requires, just as did the previously discussed normal forms, that relations do not exhibit insertion and deletion anomalies. However, unlike the other normal forms, DK/NF is not defined in terms of FD's, MVD's or JD's. The central requirement of the DK/NF is the basic concepts of domains, keys and general constraints. We elaborate on each of these requirements in the following discussions. A relation scheme is in the DK/NF if every general constraint can be inferred from the knowledge of the attributes involved in the scheme, their underlying domains and the sets of attributes which form the keys. An insertion anomaly in the case of DK/NF occurs when a tuple is inserted in a relation and the resulting relation violates one or more general constraints. Similarly, a deletion anomaly occurs when a tuple from a relation is deleted and the remaining relation violates one or more general constraints. We illustrate these dependencies and general constraints in Example 7.10.

Example 7.10: Consider the relation scheme **TRANSCRIPT** (*Student#*, *Course*, *Grade*).

Student#	*Course*	*Grade*

23714539	353	A
42717390	329	A
23714539	928	P
38815183	456	F
37116259	293	B
82317293	491	C
82317293	953	F
23714539	491	C
11011978	353	A
83910827	979	P

Figure A The TRANSCRIPT relation

Suppose the attributes *Student#* and *Course* are numeric, and are 8 and 3 digits long, respectively. The attribute *Grade* is a letter grade and could be A, B, C, D, P, F. The general constraint is that for *Course*s numbered 900 through 999, the *Grade* assigned is only P or F. For *Course*s 000 through 899, the *Grade* can only be A, B, C, D, F. The domain constraints for this relation are the following: *Student#* is required to be 8 digits long, *Course* is 3 digits long, and *Grade* has to be from the set {A, B, C, D, P, F}. The key constraint for the relation is that no two tuples could exist with the same values for the key attributes which are *Student#*, and *Course*. Obviously, *Student#*, *Course* → *Grade*. Finally, the general constraint can be expressed by the following:

if Course ≥ 900

then Grade ∈ {P, F}

else Grade ∈ {A, B, C, D, F}

The problem with this relation is that a tuple such as (12345678, 991, A), which satisfies all the DCs and KCs can be inserted in the relation TRANSCRIPT of Figure A. However, since the tuple does not satisfy the general constraint, the relation TRANSCRIPT will become illegal after the insertion.

We now give the formal definition of DK/NF.

Definition: A normalized relation scheme **R** {**S**, Γ, **σ**}, where **S** is the set of attributes, Γ is the set of DCs and KCs, and **σ** is the set of general constraints, is in the **Domain Key Normal Form(DK/NF)** if Γ ⊨ **σ** for every constraint in **σ**.

A normalized relation is in the DK/NF if the DCs and KCs imply the general constraints. The DK/NF is considered to be the highest form of normalization, since all insertions and deletion anomalies are eliminated and all general constraints can be verified by using only the DCs and KCs. For the TRANSCRIPT relation of Example 7.10, we can use the following decomposition to get two relations which are in DK/NF.

Example 7.11: The TRANSCRIPT relation of the Example 7.10 can be decomposed into the following relations:

TRANSCRIPTS_REGULAR(*Student#*, *Course*, *Grade*) with the domain constraints (*Student#* being 8 digit, *Course* being 3 digit in the range 000 through 899 and Grade in the set {A, B, C, D, P, F}) -the key as before is *Student#*, *Course*. and

TRANSCRIPTS_SPECIAL(*Student#*, *Course*, *Grade*) with the domain constraints (*Student#* being 8 digit, *Course* being 3 digit in the range 900 through 999 and Grade in the set {P, F}) the key as before is *Student#*, *Course*.

A MVD can be expressed as a general constraint. To examine the insertion and deletion anomalies in such a situation, let us look at the following example using a software company.

Example 7.12: The work of the company is organized as projects and the employees are grouped as teams. A number of projects are assigned to each group and it is assumed that all employees in the group are involved with each and every project assigned to it. This is the general constraint for the relation TEAMWORK(*Group*, *Employee*, *Project*) as shown in Figure B(i. Assume that the domain of the attributes are a character string of, let us say, length 20. The only key of the relation is the entire relation.

The insertion of a legal tuple, let us say (B, Su, FILE_MANAGER), causes the relation TEAMWORK to become invalid. This is because the general constraint is no longer satisfied and requires the insertion of additional tuples.

Similarly, the deletion of the tuple (A, Lalonde, FILE_MANAGER) makes the relation TEAMWORK violate the general constraint and requires the deletion of additional tuples.

In order to convert the relation into DK/NF, we can decompose it into the two relations, TEAM(*Group*, *Employee*) and WORK(*Group*, *Project*).

Group	*Employee*	*Project*
A	Jones	HEAP_SORT
A	Smith	HEAP_SORT
A	Lalonde	HEAP_SORT
A	Jones	BINARY_SEARCH
A	Smith	BINARY_SEARCH
A	Lalonde	BINARY_SEARCH
B	Evan	B++_TREE
B	Lalonde	B++_TREE
B	Smith	B++_TREE
B	Evan	FILE_MANAGER
B	Lalonde	FILE_MANAGER
B	Smith	FILE_MANAGER

(i)

Group	*Employee*
A	Jones
A	Smith
A	Lalonde
B	Evan
B	Lalonde
B	Smith

Group	*Project*
A	HEAP_SORT
A	BINARY_SEARCH
B	B++_TREE
B	FILE_MANAGER

(ii)

Figure B *The TEAMWORK Relation and Its DK/NF Decompositions*

It has been shown that a relation which is in the DK/NF is also in the PJ/NF and, therefore, in the 4NF and BCNF. The proof found in [Fagi81], is beyond the scope of this text.

The advantage of DK/NF relations is that all constraints could be satisfied by ensuring that tuples of the relations satisfy the corresponding domain and key constraints. Since this is easy to implement in a database system, relations in DK/NF are preferable. However, no simple algorithms exist to help in the design of DK/NF. Moreover, it appears unlikely that relation schemes with complex constraints could be converted to DK/NF.

The theory for join dependency is well developed unfortunately, the results are negative, i.e., it has been concluded that JD's don't have a finite axiom system. Consequently, we have to be content with relations in the 3NF or BCNF. Since we cannot always guarantee that the BCNF relations will be dependence preserving when both lossless and dependence preserving relations are required, we have to settle for the third normal form.[3]

7.6 Summary

The decomposition approach we examined in Chapter 6 starts with a relation and the associated set of constraints in the form of functional dependencies.. The relation has certain number of undesirable properties(in the form of insertion, deletion, or update anamolies0 and it is replaced by its projections. A number of desirable forms of projections have been identified, In Chapter 6 we discusses the followinf normal forms: 1NF, 2NF, 3NF and BCNF, Any relation having constraints in the form of Fds only can be decomposses into relations in third normal form: such decomposition is lossless and preserves the dependencies. Any relation having FDs can also be decomposed losslessly into relations in Boyce Codd normal form(and hence into third normal form).

In this chapter we examined the synthesis approach to designing a 3NF database and the higher normal forms namely 4NF, 5NF or PJ/NF, and DK/NF.

In the synthesis approach, the starting point of the relational database design process is an universal relation and the set of functional (and nonfunctional) dependencies that have to be enforced between the attributes of this universal relation; the synthesis procedure then synthesizes a set of third normal form relation schemes which preserves the required dependencies.

Multivalued dependencies arise when **R**, having a nonatomic attribute is converted to a normalized form. Thus, for each **X** value in such a relation, there will be a set of **Y** values associated with it. This association between the **X** and **Y** values does not depend on the values of the other attributes in the relation. A normal form called the fourth normal form has been defined for relations that have FD's as well as MVD's. We discussed an algorithm for decomposing a relation into the 4NF; however, like the BCNF decomposition algorithm, this algorithm does not always produce relation schemes which are dependence preserving. Hence if dependence preserving scheme is essential, in general, we will have to settle for the 3NF.

The 5NF is related to what is called ***join dependency***. This is the term used to indicate the property of a relation that can be decomposed losslessly into n simpler relations, but cannot be decomposed losslessly into fewer relations. A relation that is in the PJ/NF is also in the 4NF.

In a DK/NF relation scheme, it is possible to enforce all general constraints from knowledge of the

3When MVD's are conflict-free, a unique 4NF decomposition can be obtained. It has been observed that conflict-free MVD's are natural enough to cover the "real world" situation

domains of the attributes and the key constraints. This is the highest and most desirable normal form, albeit, it is not always possible to generate relation schemes in this form. Consequently, the database designer settles for a lower normal form which better meets the needs of the user community.

Key Terms

closure under MVD
dependency basis
domain key normal form
due to MVD
due to keys
embedded multivalued dependency
equivalent functional dependencies
fifth normal form
fourth normal form
general constraints
inference rules for FD and MVD
join dependency
multivalued dependency (MVD)
nonfunctional dependencies
normalization through synthesis
project join normal form
synthesis algorithm
trivial MVD
trivial join dependency

Exercises:

7.1. Given **U**{*ABCDE*} and F={ $A \rightarrow B$, $BC \rightarrow D$, $D \rightarrow BC$, $DE \rightarrow \phi$ }. Synthesize a set of 3NF relation schemes.

7.2. Given: **U**{*ABCDEFGH*} with the FD's given by{ $A \rightarrow BCDEFGH$, $BCD \rightarrow AEFGH$, $BCE \rightarrow ADEFGH$, $CE \rightarrow H$, $CD \rightarrow H$}. Synthesize a set of lossless join relation schemes.

7.3. Given the relation **R** {*ABCDE*} with the FD's {$A \rightarrow BCDE$, $B \rightarrow ACDE$, $C \rightarrow ABDE$}. What are the join dependencies of **R**? Give the lossless decomposition of **R**.

7.4. Given the relation **R** {*ABCDEF*} with the set H ={ $A \rightarrow CE$, $B \rightarrow D$, $C \rightarrow ADE$, $BD \rightarrow\!\!\rightarrow F$ }. Find the dependency basis of *BCD*.

7.5. Design a 3NF relation scheme for the database of Exercise 6.16 using the synthesis algorithm. Is the resulting database in the BCNF?

7.6. Is it possible to decompose the relation STUDENT_ADVISOR(*Name*, *Department*, *Advisor*) with the functional dependencies **F**{*Name* → *Department*, *Name* → *Advisor*, *Advisor* → *Department*} illustrated in Figure E of Example 6.19 iinto PJ/NF relation schemes? If so give the projected relation schemes.

7.7. What are the difficulties in generating a relational design wherein all relations are in the DK/NF?

7.8. Why is the 4NF preferable to the BCNF?

7.9. Show that axiom M7 is sound.

Bibliographic Notes

The universal relation concept and the associated problems were first discussed in [Kent81]. The algorithm for synthesizing relation schemes from a given set of attributes and FD's was proposed and studied in [Bern76].

MVD's were introduced by Fagin[Fagi77] and independently by Delobel[Delo78] and Zaniolo[Zani81]. Embedded MVD's were noted in [Fagi77] and [Delo78]. Join dependencies were introduced formally by Rissanen[Riss79] and examined further in [Aho78]. The project join normal [Fagin79], and the domain key normal[Fagi81] forms were conceived by Fagin. The axioms for JD were proposed by Beeri and Vardi[Beer79a] and also in [Scio82]. The algorithm for the dependency basis and its correctness and complexity issues were presented in [Beer80]. The DK/NF was proposed by Fagin in [Fagi81], wherein he proves the theorem which states that a DK/NF is also in the PJ/NF, 4NF, and BCNF. Axiom systems for generalized and template constraints can be found in [Beer84] and [Sadr81].

Textbook discussions of the relational database design are included in [Date85], [Lien85], [Kort86], and [Ullm82]. [Maie83] gives a very detailed theoretical discussion of the relational database theory including relational database design.

Bibliography

[Aho79] Aho, A.V., Beeri, C., Ullman, J.D., "The Theory of Joins in Relational Databases", ACM TODS, Vol. 4-3, September, 1979, pp.297-314.

[Arms74] Armstrong, W. W.,"Dependency Structures of Database Relationships", Proc. of the IFIP (1974), pp.580-583.

[Beer77] Beeri, C., Fagin R., Howard, J.H., "A Complete Axiomatization for Functional and Multivalued Dependencies", Proc of ACM SIGMOD International Symposium on Management of Data, 1977, pp.47-61.

[Beer79] Beeri, C., Bernstein, P.A., "Computational Problems Related to the Design of Normal Form Relational Schemes", ACM TODS, Vol. 4-1, March 1979, pp.113-124.

[Beer79a] Beeri, C., Vardi, M.Y., "On the properties of join dependencies" in "Advances in Database Theory", ed. H. Gallaire et al, Vol. 1, Plenum Press, New York.

[Beer80] Beeri, C., "On the Membership Problem for Functional and Multivalued Dependencies in Relational Databases", ACM TODS, Vol. 5-3, September 1980, pp.241-259.

[Beer84] Beeri, C., Vardi, M.Y., 'Formal Systems for Tuple and Equality Generating Dependencies', SIAM Journal of Computing, Vol. 13-1, pp. 76-98.

[Bern76] Bernstein, P.A., "Synthesizing third normal form relations from functional dependencies", ACM TODS, Vol. 1-4, March 1976, pp.277-298.

[Bisk79] Biskup, J., Dayal, U., Bernstein, P.A., "Synthesizing Independent Database Schemas" Proc. ACM SIGMOD International Symposium on Management of Data, 1979, pp.143-152.

[Codd70] Codd, E.F., "A Relational Model for Large Shared Data Banks", Communications of the ACM, Vol. 13-6, June 1970, pp.377-387.

[Codd72] Codd, E.F., "Further Normalization of the Data Base Relational Model" in 'Data Base Systems' ed. R. Rustin, Prentice-Hall, Englewood Cliffs, 1972, pp.33-64.

[Date85] Date, C.J., An Introduction to Database Systems, Vol. 1., Fourth edition, Addison Wesley, Reading, MA, 1985.

[Delo78] Delobel, C., "Normalization and Hierarchical Dependencies in the Relational Data Model" ACM TODS, Vol. 3-3, September 1978, pp.201-22.

[Fagi77] Fagin, R.,"Multivalued Dependencies and a New Normal Form for Relational Databases", ACM TODS, Vol. 2-3, September 1977, pp.262-278.

[Fagi79] Fagin, R., "Normal Forms and Relational Database Operators", ACM SIGMOD International Symposium on Management of Data, 1979, pp.153-160.

[Fagi81] Fagin, R., "A Normal Form for Relational Databases that is based on Domains and Keys", ACM TODS, Vol. 6-3, September 1982, pp.387-415.

[Kent81] Kent, W., "Consequences of Assuming a Universal Relation", ACM TODS, Vol6-4, December

1981, pp.539-556.
[Kort86] Korth, H.F., Silberschatz, A., Database Systems Concepts, McGraw Hill, New York, 1986.
[Lien81] Lien, Y. E.,"Hierarchical Schemata for Relational Databases", ACM TODS, Vol. 6-1, March 1981, pp.48-69.
[Lien85] Lien, Y. E.,"Relational Database Design", in Principles of Database Design, ed. S. Bing Yao. Prentice-Hall, Englewood Cliffs, 1985.
[Maie80] Maier, D., "Minimum Covers in the Relational Database Model", Journal of the ACM, Vol. 27-4, October 1980, pp.664-674.
[Maie83] Maier, D., The Theory of Relational Databases, Computer Science Press, Rockville, MD, 1983.
[Riss79] Rissanen, J., "Theory of Joins for Relational Databases - A Tutorial Survey", Proc. Seventh Symposium on Mathematical Foundations of Computer Science, Lecture notes in Computer Science 64, Springer-Verlag, pp. 537-551.
[Sadr81] Sadri, F., Ullman, Jeffrey D., 'Template Dependencies: A Large Class of Dependencies in Relational Databases and THeir Complete Axiomatization', Journal of the ACM, Vol. 29-2, April 1981, pp. 363-372.
[Scio81] Sciore, E., 'Real World MVD's', Proc. of the ACM SIGMOD Conf., 1981, pp. 121-132.
[Scio82] Sciore, E., 'A Complete Axiomatization of Full Join Dependencies', Journal of the ACM, Vol. 29-2, April 1982, pp. 373-393.
[Ullm82] Ullman, Jeffrey D., Principle of Database Systems, 2nd edition, Computer Science Press, Rockville, MD.,1982.
[Zani81] Zaniolo, C., Melkanoff, M.A.,"On the Design of Relational Database Schemata", ACM TODS, Vol. 6-1, March 1981, pp.1-47.