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 $X \twoheadrightarrow Y$ holds in a relation R defined on R if given two tuples t_1 and t_2 in R with $t_1(X) = t_2(X)$; 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.,

$$t_1(X) = t_2(X) = t_3(X) = t_4(X)$$

· the Y values of t_1 and t_3 are the same and the Y values of t_2 and t_4 are the same, i.e.,

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

· the R − X − Y values of t_1 and t_4 are the same and the R − X − Y values of t_2 and t_3 are the same, i.e.,

$$t_1(R - X - Y) = t_4(R - X - Y)$$
$$t_2(R - X - Y) = t_3(R - X - Y)$$

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

Employee_Name $\twoheadrightarrow$ *Dependent_NameDependent_Relationship*
Employee_Name $\twoheadrightarrow$ *Position_TitlePosition_Date*

Suppose employee Jill Jones gets a promotion on 12/15/86 to the position of manager. This involves adding two tuples to the database, one for each of her two dependents, to correctly register her employment history. A change in the value of an FD in a relation involving an MVD requires the change to be reflected in all tuples corresponding to that entity. In the 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 Boston and her home phone number changes to 368-4384, a change is required in not one tuple but six tuples (after the addition of the two tuples for an additional position). Deletion requires that more than one tuple be deleted. For example, in the SCHEDULE relation, if course 355 is canceled, two tuples must be deleted from the table shown in Figure 7.3.

Summarizing, note that in multivalued dependencies the requirement 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 leads to inconsistencies in the database. To avoid these multiple updates, it is preferable to replace a relation having undesirable MVDs with a number of more "desirable" relation schemes. We illustrate more desirable schemes in Figure 7.4

Figure 7.4 Replacing the EMPLOYEE relation with three relations.

Employee_Name	*Dependent_Name*	*Dependent_Relationship*
Jill Jones	Bill Jones	spouse
Jill Jones	Bob Jones	son
Mark Smith	Ann Briggs	spouse
Mark Smith	Chloe Smith-Briggs	daughter
Mark Smith	Mark Briggs-Smith	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	Lynn, MA	794-2356
Mark Smith	Revere, MA	452-4729

for the EMPLOYEE relation of Figure 7.2.[1] Such a scheme avoids the necessity of multiple storage of the same information.

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

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

[1]Recall our discussions on 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 containing the key of the strong entity along with the attributes of the weak entity (See Chapter 2).

Property of MVD

The following theorem for multivalued dependency is from Fagin (Fagi 77). We simply state it here. For the proof, see the bibliographic notes at the end of the chapter for the reference.

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

As a consequence of the above, a relation scheme with an 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 $\mathbf{X} \twoheadrightarrow \mathbf{Y}$ if and only if

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

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

Definition: **A trivial multivalued dependency is one that is [illegible] all relations R on a relation scheme R with $XY \subseteq R$. [illegible] if $Y \subseteq X$ or $XY = R$. Obviously if $Y = \phi$, [illegible]**

Example 7.6

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

Employee_Name → *Home_CityHome_Phone#*,
Employee_Name ↠ *Dependent_NameDependent_Relationship*,
Employee_Name ↠ *Position_TitlePosition_Date*.

the following MVDs are also satisfied:

Employee_Name ↠ *Home_CityHome_Phone#Dependent_Name Dependent_Relationship*,
Employee_Name ↠ *Home_CityHome_Phone#Position_Title Position_Date*.

(b) In Figure 7.4 the following MVDs are trivial:

Employee_Name ↠ *Dependent_NameDependent_Relationship*
Employee_Name ↠ *Position_TitlePosition_Date* ■

7.3.2 Axioms for Functional and Multivalued Dependencies

To design a relational database, given a relation scheme **R** with functional and multivalued dependencies, we need a set of rules or axioms that will allow us to deter-

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

- $\mathbf{U} - \mathbf{X} = \mathbf{Y}_1\mathbf{Y}_2 \ldots \mathbf{Y}_n$,
- $\mathbf{Y}_1, \mathbf{Y}_2, \ldots \mathbf{Y}_n$ are **pairwise disjoint**, i.e., $\mathbf{Y}_i \cap \mathbf{Y}_j = \phi$ for $i \neq j$, and
- For any MVD $\mathbf{X} \rightarrow\rightarrow \mathbf{Z}$ 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 $\mathbf{X} \rightarrow\rightarrow \mathbf{Z}$ 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 FDs and MVDs. It follows that for each set $\mathbf{Y}_i \in$ **DEP(X)**, $\mathbf{X} \rightarrow\rightarrow \mathbf{Y}_i$ is in $\mathbf{D}^+$.

The MVD $\mathbf{X} \rightarrow\rightarrow \mathbf{Y}_i$ 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 $\mathbf{X} \rightarrow\rightarrow \mathbf{Y}$ is implied by a set **D** of FDs and MVDs, as $\mathbf{X}^+$ was used to determine if any FD $\mathbf{X} \rightarrow \mathbf{Y}$ was implied by a set of FDs **F.**

Algorithm 7.2 computes the dependency basis of **X.** It simply converts each FD into an MVD and then applies the rules of the MVD to decompose the MVDs into simpler MVDs. 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.**

The following example illustrates the use of Algorithm 7.2

Example 7.7

Consider a database to store student information that contains the following attributes: students' names (S), their majors (M), the department they are registered in (S_d), their advisers' name (A), the courses they are taking (C), the departments responsible for the course (C_d), the final grades of the students 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. Assume that the students' names and the advisers' names are unique. The database must satisfy the following set H of functional and multivalued dependencies:

$$
\begin{array}{lll}
S & \rightarrow & MA \\
M & \rightarrow & S_d \\
A & \rightarrow & S_d \\
C & \rightarrow & C_dP \\
P & \rightarrow & P_d \\
RTD & \rightarrow & C \\
TPD & \rightarrow & R \\
TSD & \rightarrow & R \\
SC & \rightarrow & G \\
C & \rightarrow\rightarrow & RTD \\
C & \rightarrow\rightarrow & SMG
\end{array}
$$

We want to compute DEP(*C*) using Algorithm 7.2. The first step will convert all FDs into MVDs.

Step 3 will give us the set **S** with the following sets of attributes:

$\{C_dP\}$, $\{RTD\}$, $\{SMG\}$, $\{SMAS_dP_dRTDG\}$, $\{SMAS_dP_dG\}$, $\{AS_dC_dPP_dRTD\}$.

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, **DEP**(*C*), the dependency basis of *C* under the above set of FDs and MVDs:

$\{C_dP\}$, $\{RTD\}$, $\{SMG\}$, $\{S_d\}$, $\{A\}$, $\{P_d\}$

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

7.3.4 Fourth Normal Form

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

[illegible] [illegible] a relation scheme **R** such that the set **D** of FDs and MVDs are satisfied, [illegible] attributes **X** and **Y** where $\mathbf{X} \subseteq \mathbf{R}$, $\mathbf{Y} \subseteq \mathbf{R}$. The relation scheme [illegible] form (4NF) if for all multivalued dependencies of the [illegible] either $\mathbf{X} \to\to \mathbf{Y}$ is a trivial MVD or **X** is a superkey of **R**. [illegible] if all relation schemes included in the database [illegible]

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

7.3.5 Lossless Join Decomposition into Fourth Normal Form

Given a relation scheme that is not in fourth normal form, we would like to decompose it into a set of relations that are in 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 tions are the same as the ones shown in Figure 7.4.) The relations of Figure 7.5a and b have the trivial multivalued dependency $\mathbf{X} \twoheadrightarrow \mathbf{Y}$ with $\mathbf{R} = \mathbf{XY}$. In addition, they are all key relations. A nontrivial 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.

7.3.6 Enforceability of Dependencies in the Fourth Normal Form

The fourth normal form decomposition algorithm produces a lossless relation scheme; however, it may not preserve all the dependencies in the original non-4NF relation scheme. In Example 7.8, 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 4NF. The following properties are used to find the dependencies that apply to the decomposed schemes.

Given **R** and the set of FDs and MVDs **D**, 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 $\mathbf{X} \rightarrow (\mathbf{Y} \cap \mathbf{R_1})$ holds 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 $\mathbf{X} \twoheadrightarrow (\mathbf{Y} \cap \mathbf{R_1})$ holds in $\mathbf{R_1}$.

Example 7.8 illustrates this method.

Example 7.8

Consider $\mathbf{R}(A, B, C, D, E, F, G)$ with the set **H** of FDs and MVDs given by $\mathbf{H}\{A \twoheadrightarrow B, B \twoheadrightarrow G, B \twoheadrightarrow EF, CD \rightarrow E\}$.

R is not in 4NF since for the nontrivial **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 $\mathbf{R}(A, C, D, E, F, G)$. $\mathbf{R_1}$ is in 4NF; however, the reduced relation **R** is not in 4NF.

Now the MVDs $A \twoheadrightarrow B$ and $B \twoheadrightarrow G$ give 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 $\mathbf{R}(A, C, D, E, F)$. $\mathbf{R_2}$ is in 4NF; however, the reduced relation **R** is still not in 4NF.

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

The MVDs $A \twoheadrightarrow B$, $B \twoheadrightarrow EF$ by axiom **M6** give $A \twoheadrightarrow EF - B$, 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 $\mathbf{R}(A, C, D)$.

$\mathbf{R}(A, C, D)$ is in 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 dependency preserving. Thus if lossless as well as dependency preserving decomposition is required, we may have to settle for simple 3NF relation schemes, unless the BCNF decomposition is lossless as well as dependency preserving. An approach that could be used to derive a dependency 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**. In this process, the MVDs should be eliminated before trying to eliminate FDs. The intuitive reason for this is that the FDs convey more semantics about the data than the MVDs.

Dependency preserving decomposition involving **D**, a set of FDs and MVDs, 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 should consult the references in the bibliographic notes. Attempts have been made to find a synthesis algorithm to construct a relation scheme from a set of FDs and MVDs. 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. 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 should 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, 3NF is preferable to 2NF, BCNF is preferable to 3NF, and so on.

However, recent research in relational database design theory has discovered higher and higher, hence more desirable normal forms. **Fifth normal form (5NF)** is a case in point. It is related to **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 workers and has a variety of projects.

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

[2]Elimination of redundant dependencies doesn't guarantee dependency-preserving decomposition, in general. However, with conflict-free MVDs, the lossless decomposition is also dependency preserving. Conflict-free MVD sets are equivalent to acyclic join dependencies (Lien 85, Scio 81).

Figure 7.9 Decomposition of relation of Figure 7.8.

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 Systems	Database Systems
File Systems	Operating Systems

(a)

Employee	*Expertise*
Brent	User Interface
Brent	Artificial Intelligence
Mann	VLSI Technology
King	Relational Calculus
Ito	Relational Algebra
Ito	Relational Calculus
Smith	Database Systems
Smith	Operating Systems

(b)

Employee	*Project*
Brent	Work Station
Mann	Work Station
King	SQL 2
Ito	SQL 2
Ito	QBE ++
Smith	File Systems
Smith	Query Systems
Smith	Work Station

(c)

Mann, and Smith combined. Brent is assigned the User Interface and Artificial Intelligence related role, Mann is assigned the VLSI Technology related role, and Smith is assigned the Operating Systems role. 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; it is an all-key relation and therefore in fourth normal form. Unlike the relation **PROJECT_ASSIGNMENT**, the relation **NEW_PROJECT_ASSIGNMENT** 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, create a relation that contains extraneous tuples; thus the corresponding decomposition is not lossless. These superfluous tuples are removed when the resulting relation is joined with the third relation. Note that the MVDs, similar to those exhibited in Figure 7.6, are embedded in this example.

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, although it may not be possible to find a lossless decomposition of a relation scheme into two relation schemes, the same relation scheme can be decomposed losslessly into three relation schemes. This property is referred to as the join dependency (JD).

Definition: Given a relation scheme **R**, consider the following set of its projections: $\{R_1, R_2, \ldots, R_n\}$. A relation $R(\mathbf{R})$ satisfies the join dependency $*[R_1, R_2, \ldots, R_n]$, if and only if the join of the projection of R on R_i, $1 \le i \le n$, is equal to R.

$$R = \pi_{R_1}(R) \bowtie \pi_{R_2}(R) \bowtie \ldots \bowtie \pi_{R_n}(R)$$

In other words, join dependency is the assertion that the decomposition of R onto $R_1, \ldots, R_n$ 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 $*[\mathbf{R_1}, \mathbf{R_2}, \ldots \mathbf{R_n}]$ is that $\mathbf{R} = \mathbf{R_1} \cup \mathbf{R_2} \cup \ldots \cup \mathbf{R_n}$.

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 nontrivial MVD, then by Fagin's theorem (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 nontrivial MVDs. The first part of this statement can be confirmed as follows: The relation $R(\mathbf{R})$ satisfies the MVD $\mathbf{X} \twoheadrightarrow \mathbf{Y}$ if and only if the decomposition of R into **XY** and $\mathbf{R} - \mathbf{Y}$ is lossless. This is equivalent to saying that $R(\mathbf{R})$ satisfies the JD $*[\mathbf{XY}, \mathbf{R} - \mathbf{Y}]$. Conversely, **R** satisfies the JD $*[\mathbf{R_1}, \mathbf{R2}]$ if $\mathbf{R_1} \cap \mathbf{R_2} \twoheadrightarrow \mathbf{R_1}$, or $\mathbf{R_1} \cap \mathbf{R_2} \twoheadrightarrow \mathbf{R_2}$. However, not all JDs are equivalent to MVD, as seen in Figures 7.8 and 7.9.

A join dependency on the relation scheme **R**, in addition to those for MVDs, 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 $R(\mathbf{X_1}, \mathbf{X_2}, \ldots, \mathbf{X_m})$ and if $\mathbf{X_i}$s are the superkeys of **R**, then the join dependency $*[\mathbf{X_1}, \mathbf{X_2}, \ldots, \mathbf{X_m}]$, is due to the keys of R.

constraint, the relation TRANSCRIPT becomes 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 domain key normal form (DK/NF) if Γ ⊨ σ for every constraint in σ.**

A normalized relation is in 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 insertion 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 in DK/NF.

Example 7.11

The TRANSCRIPT relation of 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, F}). The key as before is *Student#Course*.

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*. ■

An MVD can be expressed as a general constraint. To examine the insertion and deletion anomalies in such a situation, let us look at Example 7.12 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 project assigned to it. This is the general constraint for the relation TEAMWORK(*Group, Employee, Project*) as shown in Figure Bi. Assume that the domain of the attributes are a character string of length 20. The only key of the relation is the entire relation.

The insertion of a legal tuple, (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.

Figure B The TEAMWORK relation and its DK/NF decompositions.

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)

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*). This is shown in Figure Bii. ■

It has been shown that a relation in DK/NF is also in PJ/NF and, therefore, in 4NF and BCNF. The proof, found in (Fagi 81), 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. It has been concluded that JDs don't have a finite axiom system. Consequently, we have to be content with relations in 3NF or BCNF. Since we cannot theorem that 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 (Beer 84) and (Sadr 81).

Textbook discussions of the relational database design are included in (Date 85), (Lien 85), (Kort 86), and (Ullm 82). (Maie 83) gives a very detailed theoretical discussion of the relational database theory including relational database design.

Bibliography

(Aho 79) A. V. Aho, C. Beeri, & J. D. Ullman, "The Theory of Joins in Relational Databases," *ACM TODS* 4(3), September 1979, pp. 297–314.

(Arms 74) W. W. Armstrong, "Dependency Structures of Database Relationships." Proc. of the IFIP, 1974, pp. 580–583.

(Beer 77) C. Beeri, R. Fagin, & J. H. Howard, "A Complete Axiomatization for Functional and Multivalued Dependencies," Proc. of ACM SIGMOD International Symposium on Management of Data, 1977, pp. 47–61.

(Beer 79a) C. Beeri, & P. A. Bernstein, "Computational Problems Related to the Design of Normal Form Relational Schemes," *ACM TODS* 4(1), March 1979, pp. 113–124.

(Beer 79b) C. Beeri, & M. Y. Vardi, "On the Properties of Join Dependencies," in H. Gallaire et al., ed., *Advances in Database Theory,* vol. 1. New York: Plenum Press, 1979.

(Beer 80) C. Beeri, "On the Membership Problem for Functional and Multivalued Dependencies in Relational Databases," *ACM TODS* 5(3), September 1980, pp. 241–259.

(Beer 84) C. Beeri, & M. Y. Vardi, "Formal Systems for Tuple and Equality Generating Dependencies," *SIAM Journal of Computing* 13(1), pp. 76–98.

(Bern 76) P. A. Bernstein, "Synthesizing Third Normal Form Relations from Functional Dependencies," *ACM TODS* 1(4), March 1976, pp. 277–298.

(Bisk 79) J. Biskup, U. Dayal, & P. A. Bernstein, "Synthesizing Independent Database Schemas," Proc. ACM SIGMOD International Symposium on Management of Data, 1979, pp. 143–152.

(Codd 70) E. F., Codd, "A Relational Model for Large Shared Data Banks," *Communications of the ACM* 13(6), June 1970, pp. 377–387.

(Codd 72) E. F. Codd, E.F "Further Normalization of the Data Base Relational Model," in R. Rustin, ed., *Data Base Systems.* Englewood Cliffs, NJ: Prentice-Hall, 1972, pp. 33–64.

(Date 85) C. J. Date, *An Introduction to Database Systems,* vol. 1, 4th ed. Reading, MA: Addison-Wesley, 1985.

(Delo 78) C., Delobel, "Normalization and Hierarchical Dependencies in the Relational Data Model," *ACM TODS* 3(3), September 1978, pp. 201–22.

(Fagi 77) R. Fagin, "Multivalued Dependencies and a New Normal Form for Relational Databases," *ACM TODS* 2(3), September 1977, pp. 262–278.

(Fagi 79) R. Fagin, "Normal Forms and Relational Database Operators," ACM SIGMOD International Symposium on Management of Data, 1979, pp. 153–160.

(Fagi 81) R. Fagin, "A Normal Form for Relational Databases that is Based on Domains and Keys," *ACM TODS* 6(3), September 1982, pp. 387–415.

(Kent 81) W. Kent, "Consequences of Assuming a Universal Relation," *ACM TODS,* 6(4), December 1981, pp. 539–556.

(Kort 86) H. F. Korth, & A. Silberschatz, *Database Systems Concepts.* New York: McGraw-Hill, 1986.

(Lien 81) Y. E. Lien, "Hierarchical Schemata for Relational Databases," *ACM TODS,* 6(1), March 1981, pp. 48–69.

(Lien 85) Y. E. Lien, "Relational Database Design," in S. Bing Yao, ed., *Principles of Database Design.* Englewood Cliffs, NJ: Prentice-Hall, 1985.

(Maie 80) D. Maier, "Minimum Covers in the Relational Database Model," *Journal of the ACM*. 27(4), October 1980, pp. 664–674.

(Maie 83) D. Maier, *The Theory of Relational Databases*. Rockville, MD: Computer Science Press, 1983.

(Riss 79) J. Rissanen, "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, New York pp. 537–551.

(Sadr 81) F. Sadri, & J. D. Ullman, "Template Dependencies: A Large Class of Dependencies in Relational Databases and Their Complete Axiomatization," *Journal of the ACM*. 29(2), April 1981, pp. 363–372.

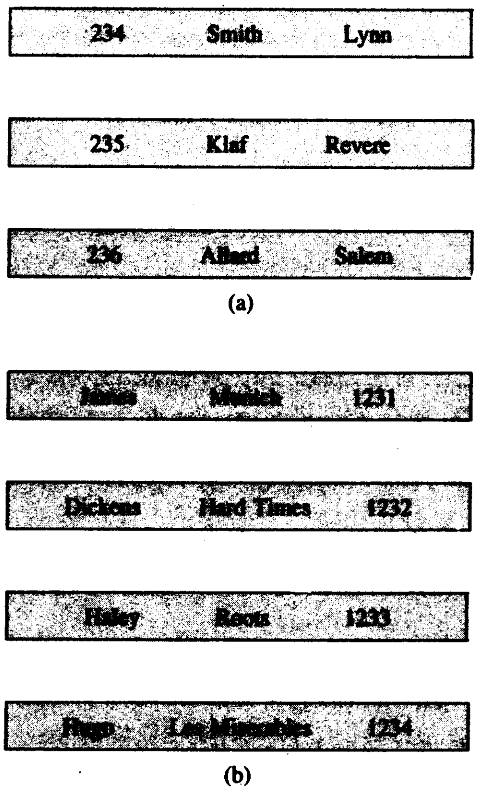
(Scio 81) E. Sciore, "Real World MVDs," Proc. of the ACM SIGMOD Conf., 1981, pp. 121–132.

(Scio 82) E. Sciore, "A Complete Axiomatization of Full Join Dependencies," *Journal of the ACM*. 29(2), April 1982, pp. 373–393.

(Ullm 82) J. D. Ullman, *Principles of Database Systems*, 2nd ed. Rockville, MD: Computer Science Press, 1982.

(Zani 81) C. Zaniolo, & M. A. Melkanoff, "On the Design of Relational Database Schemata," *ACM TODS*. 6(1), March 1981, pp. 1–47.

Figure 8.1 Occurrences of CLIENT and BOOK record types.

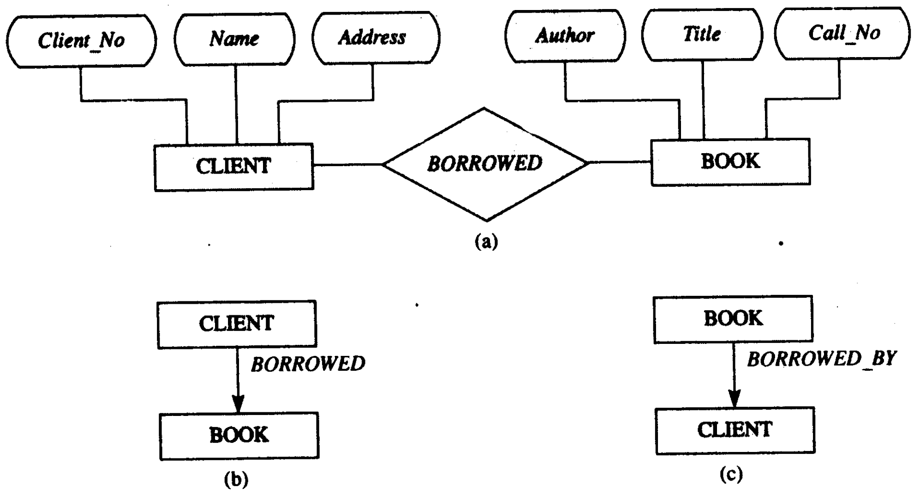


8.1.1 Expressing Relationship: The DBTG Set

The relationship of a client borrowing a book from the library may be represented by the entity relationship diagram of Figure 8.2a. The corresponding data structure diagram is shown in Figure 8.2b. In part a, we have the entity set CLIENT, which is related to the entity set BOOK in a one-to-many relationship; a client may have borrowed several books. Later we look at the possibility of a many-to-many relationship, where we show that a client has borrowed several books, as shown in part b, and also that a book (or a copy of the book) may have been borrowed by many clients, as shown in part c.

To express the relationship between the client and the borrowed book, the network model uses the set construct. The word *set* used here does not imply the mathematical meaning but indicates that there is a relationship between two record types. A set type represents a one-to-many relationship from the E–R model. An instance of the relationship is expressed by an instance or occurrence of the set type. A set consists of an owner record type and one or more member record type(s). The DBTG proposal of 1971 did not allow a record type to be both an owner and a member within the same set type. However, in the 1978 version of the proposal this restric-

Figure 8.2 Relationship between CLIENT and BOOK.



tion was eliminated. In the revised version, the records participating in a set type may be of the same type or of different types (We examine this aspect of the set construct in Section 8.4.) An occurrence of a set type consists of one occurrence of the owner record type and zero or more occurrences of the member record type(s).

The data structure diagram of Figure 8.2b represents the set *BORROWED;* the owner record type is CLIENT and the member record type is BOOK. The relationship between them is represented by the directed arc labeled with the name of the set; it is a functional link. The direction of the arc is from the owner to the member record type. The direction of the functionality is opposite to the direction of the arc. Each occurrence of the set *BORROWED* represents a relationship between a client and the books he or she borrows. If we want to represent the fact that a given book could have been borrowed by many clients, we must have, in addition to the set of Figure 8.2b, another set *BORROWED_BY*, as shown by the data structure diagram of Figure 8.2c. In the set *BORROWED_BY*, BOOK is the owner record type and CLIENT is the member record type.

Even though we can show a many-to-many relationship between two entities by data structure diagrams as in Figure 8.2b and c, its direct implementation is not allowed in the NDM. (We examine the reasons for this in Section 8.3 and show how a many-to-many relationship is implemented in the NDM.)

The set *BORROWED* can be defined as follows:

set is ***BORROWED***
    owner is CLIENT
    member is BOOK
    end

Figure 8.3a gives some occurrences of the set type *BORROWED*. As we can see there is a one-to-many relationship expressed in this set; a CLIENT could borrow more than one book. If we allow the possibility that there could be more than one copy of the same book, then the relationship between CLIENT and BOOK becomes many-to-many; this is shown in Figure 8.3b.

the record occurrences corresponding to DEPT_SECTIONs of that BRANCH. On the next level we find the set type *WORKS_IN;* here the owner is the record type DEPT SECTION and the member is the record type EMPLOYEE.

A simple database corresponding to the diagram of Figure 8.4 is shown in Figure 8.5. Here an occurrence of the record type LIBRARY, the MUC Public Library System, is the owner of the set *HAS*. The members of this set occurrence are the two occurrences of the record type BRANCH, Lynn and Revere. The record occurrence Lynn of the record type BRANCH is the owner of one of the occurrences of the set type *CONTAIN* and this set has as its members the record occurrences Adult_Sec (adult section), Childrn_Sec (children's section), Acqstn_Dept (acquisition department), Crcln_Dept (circulation department), and Ref_Dept (reference department) of the record type DEPT_SECTION. The record occurrence Adult_Sec, in its turn, is the owner in the set type *WORKS_IN* occurrence and has the record occurrence of the record type EMPLOYEE, for instance Barry, as its member.

8.1.3 Complex Multilevel Set Construct

Figure 8.6 is a portion of the library database example of Figure 8.4. However, here we have split the original record type DEPT_SECTION into two separate record types DEPT and SECTION.

We illustrate in this example that the DBTG proposal allows a set to have more than one record type as its member record type. For instance, the set *CONTAINS* has two record types as its members. This is not the same as replacing the set *CONTAINS* with two sets, for example, *CONT_SEC* and *CONT_DEPT*. The data structure diagram for this modification is shown in Figure 8.7.

At this point we might ask the following questions:

- Can the EMPLOYEE record occurrence Carrie in Figure 8.5 be a member of the two occurrences of the type set *WORKS_IN* where the owner records are the occurrences Adult_Sec and Childrn_Sec?

Figure 8.5 Sample database corresponding to Figure 8.4.

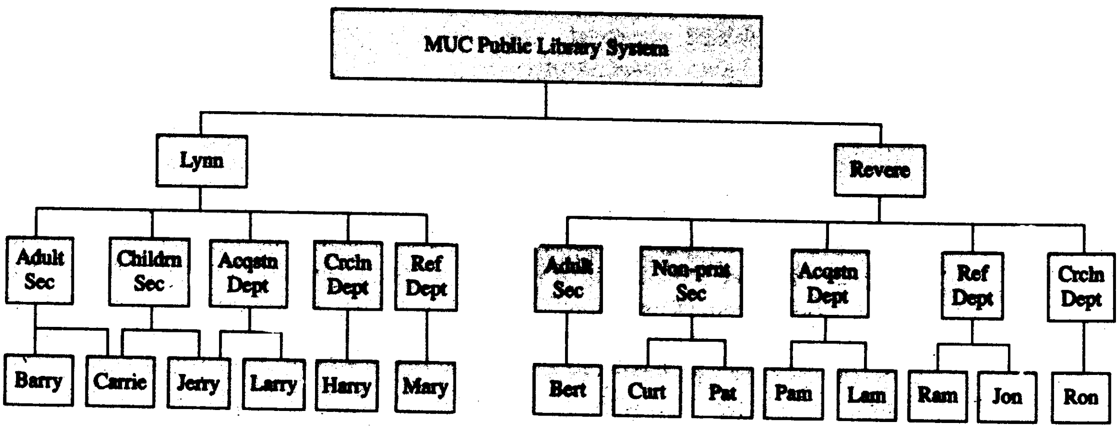
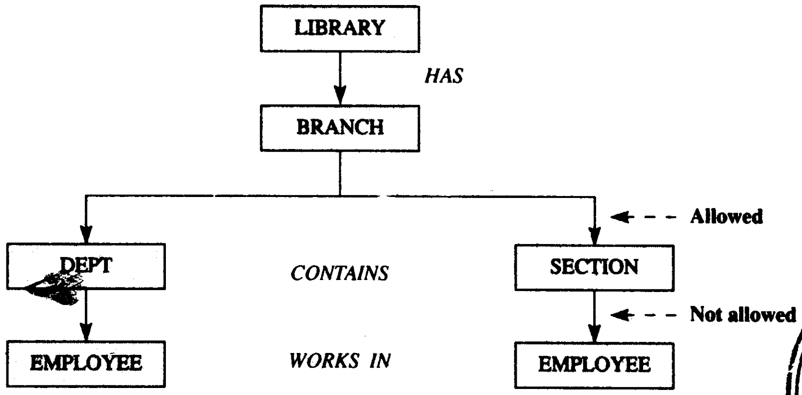


Figure 8.6 Complex multilevel set construct.



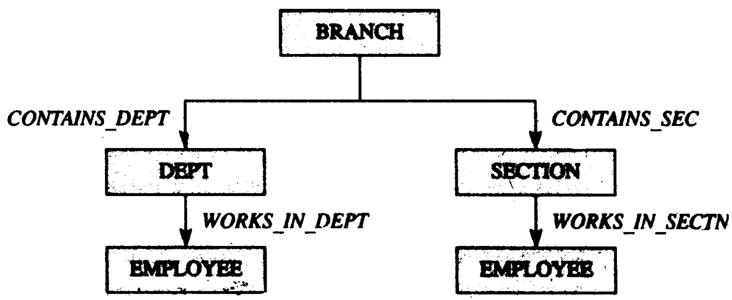
- Can the EMPLOYEE record occurrence Jerry be a member of the set *WORKS_IN* where the owner records are the occurrences Childrn_Sec of record type SECTION and Acqstn-Dept of record type DEPT?
- Can the set type *WORKS_IN* have as its owner record a record from two different record types, SECTION and DEPT?

From Figure 8.6 we also notice that the set type *WORKS_IN*, as it is shown, has two different record types as it owner record type. The DBTG proposal allows a given set type to include member records from more than one record type, but does not allow a set type to have the owner record coming from two different record types. Thus the set *WORKS_IN*, as indicated in Figure 8.6, is not allowed, The DBTG model requires that the intent of the design must be represented as two sets, for instance, *WORKS_IN_DEPT* and *WORKS_IN_SECT*. This modification is shown in the modified data structure diagram of Figure 8.7.

The network data model as proposed in the DBTG proposal has certain restrictions, which we discuss in the following section. These restrictions mean that the answer to each of the above questions is in the negative.

The data structure diagrams of Figures 8.7 and 8.8 illustrate the difference between a set type that can have records from two record types as its member record

Figure 8.7 One record type owner of two set types.



8.3 Expressing an M:N Relationship in DBTG

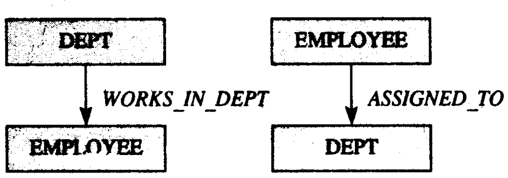
Let us now see how we can express the following relationship in the DBTG model. We would like to model a situation where an employee is able to help out in different departments depending on the workload. For example, during the evening, when there are more people in the library, it is common to increase the number of clerks at the circulation desk. An employee assigned to the acquisition department could also be designated to work in the circulation department. To allow for the possibility of an employee being assigned to work in more than one department, we need to express a many-to-many relationship. In this many-to-many relationship, a department has many employees and the employees are assigned to more than one department. This could be implemented indirectly by expressing two one-to-many relationships and using an intermediate record, the so-called intersection or common information-bearing record type. Such common information between the two original record types could, however, be null.

In the DBTG model we can express this M:N relationship by two set types. In one set type, the DEPT is the owner record type and the members are the record occurrences of the EMPLOYEE record type. In the second set type, the owner is an EMPLOYEE record occurrence and the members are the DEPT record occurrences. These sets are shown by the data structure diagram of Figure 8.10. However, the DBTG set construct does not allow the implementation of these sets. Suppose we allow an employee to work in more than one department. Then the record occurrence for that employee will appear as a member record in more than one occurrence of the set *WORKS_IN_DEPT*. This violates the DBTG restriction that a record occurrence can be a member of only one occurrence of a given set type. Similarly, for the set *ASSIGNED_TO* we find that since there are many EMPLOYEEs in a given DEPT a given occurrence of a record for that DEPT will be a member of more than one occurrence of this set type.

The above reasoning can be used to explain why we could not directly show the many-to-many relationship between a CLIENT and a BOOK as in Figures 8.2b and c.

The method for resolving this problem in the DBTG model is to introduce an intermediate record type between the two entity sets involved in the many-to-many relationship. This intermediate record type is sometimes called the **intersection record** or the **connection record.** This new record holds data common to the many-to-many relationship of the original entities represented by their respective record types.

Figure 8.10 Incorrect method of expression an M:N relationship in DBTG.



Therefore, to express the above M:N relationship we introduce the record type HOURS_ASSGND, which may be defined as follows:

type HOURS_ASSGND = *record*
Dept: string;
Employee: string;
Hours: integer;
end

A correct representation of the many-to-many relationship of Figure 8.10 is now expressed by introducing the sets *EMP_ASSGND* and *DEPT_ASSGND* with the record types DEPT and EMPLOYEE as owner and the intermediate record type HOURS_ASSGND as member in both the sets. A data structure diagram for this correct representation of the relationship is shown in Figure 8.11.

Figure 8.12 shows a possible method of implementing the M:N relationship using the intermediate record containing space for the common data and two pointers, one for each of the sets it is involved in. The common data here is the number of hours the employee is assigned to a given department. Sometimes the intermediate record contains duplicated information, e.g., department name and employee name, to facilitate the recovery and verification operations. The list of employees assigned to the Acqstn_Dept can be determined by the set *EMP_ASSGND*, where the owner is the record occurrence Acqstn_Dept (AD) and following the list containing the intermediate records AD J 40 and AD J 30. The record AD J 40 is owned by Jerry and the record AD L 30 is owned by Larry in the set type *DEPT_ASSGND*, indicating that employees Jerry and Larry work in the Acqstn_Dept. Similarly, we can see that employee Larry is assigned to the Acqstn_Dept for 30 hours and the Crcln_Dept for 10 hours. Since Larry is assigned to two departments, there are two occurrences of the intermediate record type containing the intersection data pertaining to Larry. Similarly, the circulation department has three employees assigned to it and, hence, the set occurrence of the set type *EMP_ASSGND* with the circulation department as the owner has three member record occurrences of the intermediate record type HOURS_ASSGND.

Suppose there is a need to express another M:N relationship, let us say between the employees and their participation in a number of activity clubs run by the library. This can be implemented by introducing another intermediate record type, let us say EMP_AFFILIATION, and two set types to establish this many-to-many relationship, as shown in Figure 8.13a. The corresponding sample database is shown in Figure 8.13b.

Figure 8.11 A correct representation of M:N relationship in DBTG.

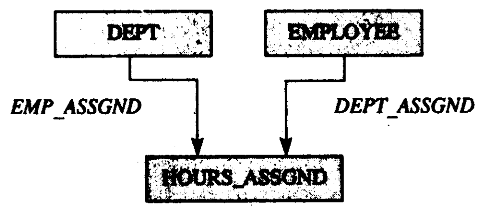
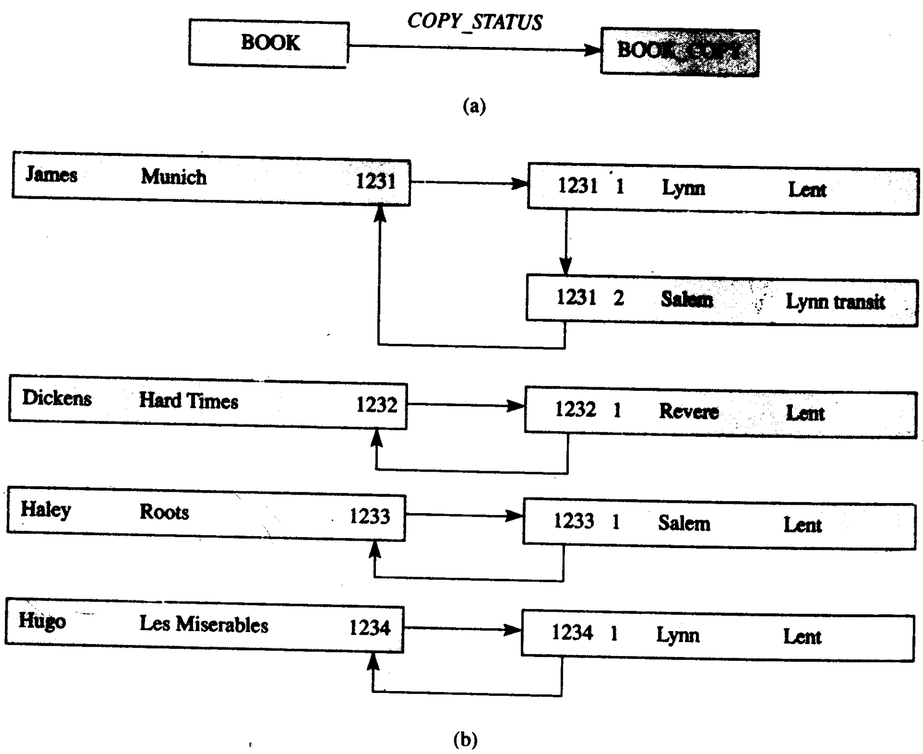
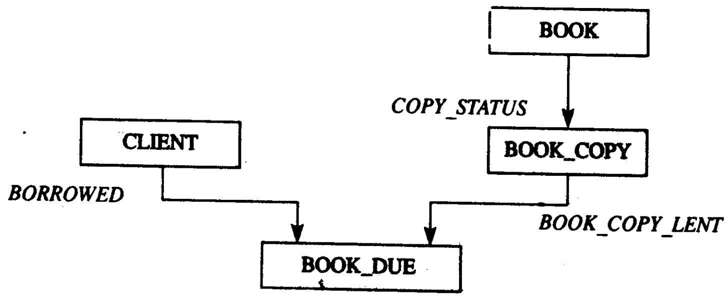


Figure 8.14 Multiple copies of BOOKs.



The many-to-many relationship of Figure 8.3b is expressed indirectly by using the one-to-many relationships between BOOK and BOOK_COPY, and CLIENT and BOOK_DUE; and a one-to-one relationship between BOOK_DUE and BOOK_COPY. These sets are shown in Figure 8.15. Each book could have a number of copies, which is shown by the set *COPY_STATUS* with owner record type being BOOK and member record type being BOOK_COPY. The BOOK_COPY taken out by a CLIENT is shown by the set *BORROWED*.

Figure 8.15 Many-to-many relationship of CLIENT and BOOKs.



8.4 Cycles in DBTG

The original DBTG set construct prohibited the same type of record to be both an owner and a member in a given set type. However, relationships of this type, sometimes called intrarecord relationships, are required to model, for example, the organizational structure of an enterprise or the part explosion of a subassembly or an assembly, as shown in Figure 8.16. The DBTG set to express this relationship contains the same type of records as the owner and member record tvoes: EMPLOYEEs for the former relationship and PARTs for the latter.

The 1978 modification of the DBTG proposal removed this restriction and allowed a set type to have the same record type as both a member and an owner. However, a given occurrence of a record could only be involved in one set occurrence as an owner and in one set occurrence as a member. This modification to the original DBTG set construct allows for the presence of cycles in the database.

A **cycle** is a path in a single-level or multilevel hierarchy of DBTG sets such that the path starting from a given record type leads back to the same record type while traversing the sets from an owner to a member. However, the return need not be to the same record occurrence.

When the same record type is declared to be both the owner record type and the member record type in the same set type, a cycle called the **single-level cycle** occurs. We illustrate this type of cycle in Figure 8.16 and discuss it in Section 8.4.1.

When a sequence of set types exists in the database such that the member record type in one set is the owner record type in the next set, a cycle called the **multilevel cycle** is said to be present. If we start with one record type, which is the owner record type in this sequence of set types, the final member record type reached as we go through this sequence of owner-member record types is the starting owner record type. (We illustrate the multilevel cycle in Figure 8.22 and focus on it in Section 8.4.2.)

8.4.1 Set Involving Only One Type of Record

Consider the set type *TEAM* (a work group or a play group) wherein the owner anu member record types are EMPLOYEE. The owner of a set occurrence of this set

Figure 8.16 Single-level cycles.

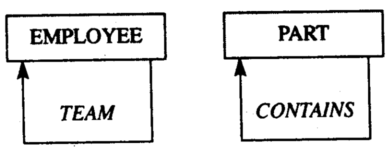
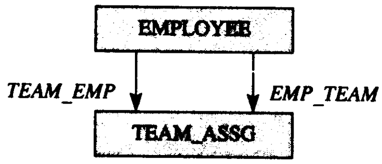


Figure 8.20 One record type with intersection record.



The sets *TEAM_EMP* and *EMP_TEAM* can be defined as follows:

sets *TEAM_EMP*
    owner is EMPLOYEE
    member is TEAM_ASSG
    end
set is EMP_TEAM
    owner is EMPLOYEE
    member is TEAM_ASSG
    end

A sample database involving this many-to-many relationship between occurrences of the record type EMPLOYEE is given in Figure 8.21. Here the owner of the two set occurrences of the set type *TEAM_EMP* are the records Barry and Harry of the record type EMPLOYEE. The members in the sets are the record occurrences {Barry Jerry 10, Barry Larry 15}, and {Harry Jerry 30, Harry Larry 25, Harry Mary 40} respectively. There are three occurrences of the set type *EMP_ASSG* with owners Jerry, Larry, and Mary. The corresponding members are the record occurrences {Barry Jerry 10, Harry Jerry 30}, {Barry Larry 15, Harry Larry 25}, and {Harry Mary 40}, respectively.

Figure 8.21 M:N relationship involving single record type.

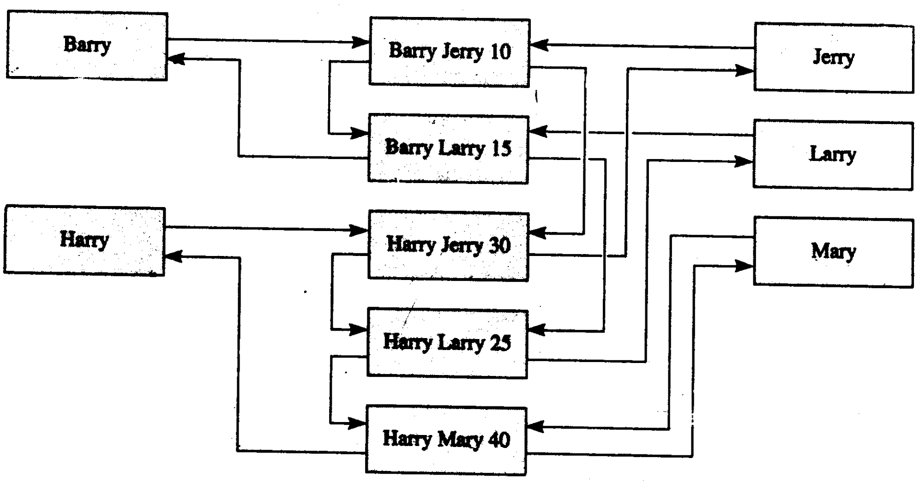
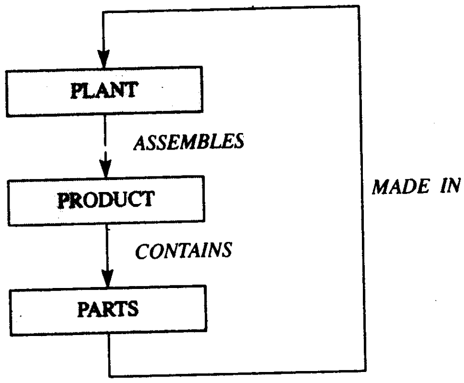


Figure 8.22 A cycle involving different record types.



8.4.2 Sets Involving Different Record Types in a Cycle

Figure 8.22 is an example of a data structure diagram showing a cycle involving different record types. In this figure we indicate that a plant assembles a number of products. Each product is made from a number of parts and these parts are made in some plants.

With the automatic set insertion rule (described below in Section 8.5.4) it is obvious that no data can be inserted in a database with the above type of cycle. (See exercise 8.9.)

The designer of the database, using the NDM, can decide whether to include cycles in the database, provided the DBMS software correctly handles such cycles. As in the case of loops, the cycle can be eliminated with the introduction of one or more intermediate record types.

8.5 Data Description in the Network Model

Our discussion of the data description facility of a network database model closely follows the CODASYL model.

8.5.1 Record

A DBTG record is made up of smaller units of data called data-items, vectors, and repeating groups. Records of one type or several types are related via a set, and provide the basic unit of access in the database. In previous discussions we have used a number of records, such as CLIENT, EMPLOYEE, and so on.

type may be declared as a member of one or more set types. Therefore, a record type can be both an owner and a member in one or more set types. A record may be both owner and member in the same set type. However, a record cannot be a member or an owner of more than one occurrence of a given set type. If a record type is declared as the owner type as well as the member type in the declaration of the set type, then the same record can be both an owner and a member in the same occurrence of a set type, or it can be the owner in one occurrence and a member in another.

A set contains precisely one occurrence of the owner record and any number of occurrences of each of its member record types. A set containing only an occurrence of its owner record type is an empty set. This contradicts the definition of the mathematical set which, when empty, does not contain any element. The DBTG set occurrence always has an owner record occurrence even when empty. An empty DBTG set cannot exist without the owner record occurrence.

8.5.3 Order of Members in a Set

Each set type declared in the schema must have an ordering specified for it. This ordering indicates the logical ordering for the insertion of member records into the set. The ordering specified could be ascending or descending and is based on data items in each of the member record types. The ordering could also be given as the order of insertion, in the reverse order of insertion, or before or after a selected record.

The DBTG allows the user to specify the insertion point where a member record will be connected into an occurrence of a set type. The possible order that could be defined is first, last, next, prior, system default, sorted.

If we consider the set to be implemented via a doubly linked list, starting with the owner record occurrence, then the **order** can be explained as follows:

- **order first** indicates that the member records are to be inserted immediately following the owner record, thus giving a reverse chronological order. The member record most recently inserted into a set occurrence will be the first member in the set.
- **order last** indicates that the member records are to be inserted immediately before the owner record occurrence, thus giving a chronological order. The member record most recently inserted into the set will be the last member in the set.
- **order next** and **order prior** indicates that the member records are to be inserted relative to the currency indicator (discussed in Section 8.7.2) of the run unit for the set type. If the currency indicator is pointing to the owner record, order next is equivalent to order first and order prior is equivalent to **order last.**
- **sorted** indicates that the member records are to be maintained in a sorted sequence. If the sorting is based on the value of key items of the member record types, this is specified by the user.
- **system default** indicates that the DBMS maintains the member records in an order most convenient to it.

8.5.4 Set Membership

The set membership criteria consist of the insertion and retention status or a member record type with respect to a set. The insertion status indicates how the membership of a record occurrence, within a set occurrence of a set type of which it is a member, is established. If the status is **automatic,** the insertion of the record as a member in the appropriate occurrence of the set type is performed by the DBMS when a new occurrence of the record type is stored in the database. In the following example, we declare the set *BORROWED* to be owned by the record type CLIENT and to contain the record type BOOK_DUE as its member, the membership being defined as automatic. This ensures that the library will know exactly which client has borrowed a given volume.

```
type  CLIENT = record
          Client_No: string;
          Name: string;
          Address: string;
          end
type  BOOK_DUE = record
          Call_No: integer;
          Copy_No: integer;
          Client_No: string;
          Due_Date: string;
          end;
set is  BORROWED
          owner is CLIENT
          member is BOOK_DUE automatic
          end
```

A **manual** membership status indicates that the membership is not automatic. In effect, with a manual membership, the selection of the appropriate occurrence of the set and the insertion of the record to become its member has to be done using appropriate data manipulation facilities. In the following example, the set *COLLECTION* owned by the record type BRANCH is declared to have the record type BOOK_COPY as member record, the membership being manual. Therefore, the application program is responsible for inserting an occurrence of the record type BOOK_COPY in the appropriate occurrence of the set type.

```
type BRANCH = record
          Br_Name: string;
          Address: string;
          Phone_No: string;
          end
type BOOK_COPY = record
          Call_No: string;
          Copy_No: integer;
          Branch_Id: string;
          Current_Status: string;
          end
```

set is COLLECTION
owner is BRANCH
member is BOOK_COPY *manual*
end

The retention or removal status of a record indicates the continuance of the relationship of a member record occurrence with the set type once it becomes a member of an occurrence of the set type. The retention status could be defined as fixed, mandatory, or optional.

Fixed status indicates that once a record becomes a member of an occurrence of a set type, it will continue that relationship with that particular set occurrence until the record if deleted. *('til death do us part!)* When the owner of the record in a set is deleted, if the membership retention status had been defined as fixed, all member record occurrences are deleted along with the owner. In the following example, the set *CONTAINS* owned by the BRANCH record type has DEPT and SECTION as member record types; the membership insertion status is manual and the retention status is declared to be fixed. Thus, once a department or section is assigned to a given branch, it remains in that branch and, if the branch is closed, the department and the branch is deleted as well.

set is CONTAINS
owner is BRANCH
member is DEPT *manual fixed*
member is SECTION *manual fixed*
end

Mandatory status indicates that once a record becomes a member of an occurrence of a set type, it continues that relationship with an occurrence of that set type. The particular set occurrence of which the record occurrence is a member may change but the relationship in the set type must continue. When the membership status is defined as mandatory, an attempt to delete the owner record occurrence with a nonempty set will fail until all the members are moved to another set occurrence. In the following example, the set *WORKS_IN_DEPT* is owned by the record type dept and has as its members occurrences of the record type EMPLOYEE, the insertion and retention statuses being manual and mandatory, respectively. Thus, an occurrence of the employee record type is to be inserted in the appropriate set occurrence of the set type *WORKS_IN_DEPT*. Employees could, however, be moved from one department to another. Also, once a number of employees are assigned to a department, we cannot delete that department until we move all the employees to another department.

set is WORKS_IN_DEPT
owner is DEPT
member is employee *manual mandatory*
end

Optional status allows a member record occurrence to discontinue a relationship in a set type. When the membership status is defined as optional, an attempt to delete the owner record occurrence will cause the members of the set occurrence owned by the owner record to be disconnected and the owner record occurrence to be deleted; the member record occurrence will continue to exist in the database. In the following