Figure 6.12 Decomposition of TEACHES relation: (a) COURSE_DETAILS; (b) ROOM-DETAILS; and (c) Decomposition of COURSE_DETAILS to eliminate transitive dependency.

Course	*Prof*	*Room*	*Enrol_Lmt*
353	Smith	A532	40
351	Smith	C320	60
355	Clark	H940	300
456	Turner	B278	45
459	Jamieson	D110	45

(a)

Room	*Room_Cap*
A532	45
C320	100
H940	400
B278	50
D110	50

(b)

Course	*Prof*	*Enrol_Lmt*
353	Smith	40
351	Smith	60
355	Clark	300
456	Turner	45
459	Jamieson	45

Course	*Room*
353	A532
351	C320
355	H940
456	B278
459	D110

(c)

domain of *Enrol_Lmt* is also an integer value and should be less than or equal to the corresponding value for *Room_Cap*.

The TEACHES relation is in first normal form since it contains only atomic values. However, as mentioned earlier, since the course is scheduled in a given room and since the room has the given maximum number of available seats, there is a functional dependency *Room* $\rightarrow$ *Room_Cap*, and hence by transitivity, *Course* $\rightarrow$ *Room* $\rightarrow$ *Room_Cap*. Thus, the functional dependencies in this relation are {*Course* $\rightarrow$ *(Prof, Room, Room_Cap, Enrol_Lmt), Room* $\rightarrow$ *Room_Cap*}. Also, there is another transitive dependency[4] *Room* $\rightarrow$ *Room_Cap* $\rightarrow$ *Enrol_Lmt*. The presence of these transitive dependencies in TEACHES will cause the following problems. The capacity of a room cannot be entered in the database unless a course is scheduled in that room; and the capacity of a room in which only one course is scheduled will be deleted if the only course scheduled in that room is deleted. Because the same room can appear more than once in the database, there could be inconsistencies between the multiple occurrences of the attribute pair *Room* and *Room_Cap*.

Consider the decomposition of the TEACHES relation into the relations COURSE _DETAILS *(Course, Prof, Room, Enrol_Lmt)* of Figure 6.12a and ROOM_DETAILS *(Room, Room_Cap)* of Figure 6.12b. The set of functional dependencies in COURSE DETAILS is given by {*Course* $\rightarrow$ *Prof, Course* $\rightarrow$ *Room, Course* $\rightarrow$ *Enrol _Lmt*} and the functional dependency in ROOM_DETAILS is {*Room* $\rightarrow$ *Room_Size*}. These relations do not have any partial dependencies: each of the attributes is fully

[4]Here we assume that *Enrol_Lmt* is the upper limit on registration for a course and is based solely on the room capacity.

functionally dependent on the key attribute, namely *Course* and *Room*, respectively. Hence, these relations are in second normal form. However, the relation COURSE_DETAILS has a transitive dependency since *Course* → *Room* → *Enrol_Lmt*. In addition there is an interrelation join dependency between the relation COURSE_DETAILS and ROOM_DETAILS to enforce the constraint that the *Enrol_Lmt* be less than or equal to the *Room_Cap*.

Third Normal Form

A relation scheme in third normal form does not allow partial or transitive dependencies. The decomposition of **STDINF** into **STUDENT_INFO TRANSCRIPT** and **TEACHER** gives third normal form relations.

Definition: **A relation scheme R<S, F> is in third normal form (3NF) if for all nontrivial functional dependencies in F^+ of the form X → *A*, either X contains a key (i.e., X is a superkey) or *A* is a prime attribute. A database scheme is in third normal form if every relation scheme included in the database scheme is in third normal form.**

In a third normal form relation, every nonprime attribute is nontransitively and fully dependent on the every candidate key. A relation scheme **R** is *not* in third normal form if any functional dependency such as **X → Y** implied by **F** is in conflict with the above definition of third normal form. In this case one of the following must be true:

- **X is a subset of a key of R and in this case X → *A* is a partial dependency.**
- **X is not a subset of any key of R and in this case there is a transitive dependency in F^+. Since for a key Z of RZ → X with X not in Z, and X → *A* with *A* not in X, Z → X → *A* is a nontrivial chain of dependencies**

The problems with a relation scheme that is not in 3NF are discussed below.

If a relation scheme **R** contains a transitive dependency, **Z → X →** *A*, we cannot insert an **X** value in the relation along with an *A* value unless we have a **Z** value to go along with the **X** value. This means that we cannot independently record the fact that for each value of **X** there is one value of *A*. This is the insertion anomaly. Similarly, the deletion of a **Z → X** association also requires the deletion of an **X →** *A* association leading to the deletion anomaly. If a relation **R** contains a partial dependency, i.e., an attribute *A* depends on a subset **X** of the key **K** of R, then the association between **X** and *A* cannot be expressed unless the remaining parts of **K** are present in a tuple. Since **K** is a key, these parts cannot be null.

The 3NF scheme, like the 2NF scheme, does not allow partial dependencies. Furthermore, unlike the 2NF scheme, it does not allow any transitive dependencies.

The relation COURSE_DETAILS of Figure 6.12a has a transitive dependency because *Course* → *Room* → *Enrol_Lmt*. We can eliminate this transitive dependency by decomposing COURSE_DETAILS into the relations *(Course, Prof, Enrol_Lmt)* and *(Course, Room)*. These decomposed relations are shown in Figure 6.12c. Note that enforcing the constraint that *Enrol_Lmt* be less than the *Room_Cap* now requires a join of three relations!

Normalization through Decomposition (Based on FDs)

We noted above the presence of insertion and deletion anomalies when **R** contains a partial or transitive dependency. The insertion of values for **Z** and **X** without an *A* value may be handled by using a null value, provided the attribute *A* allows null values. If null values are not allowed for *A*, the **Z** to **X** association cannot be represented without a corresponding *A* value.

In this section we will examine how to start with a relation scheme **R** and a set of functional dependencies **F** such that **R** is not in third normal form with respect to the set **F**, and arrive at a resultant set of relation schemes that are a lossless join 3NF decomposition of **R**. The relation scheme **R** can be decomposed into a number of relation schemes by projection (the intent of the decomposition being to produce simpler schemes in 3NF).

Example 6.17 Consider the relation of Figure C, ENROLLMENT(***Student_Name*, *Course*,** *Phone_No*, *Department*, *Grade*). In this relation the key is *Student_Name*, *Course* and it has the following dependencies: {*Student_Name* → *Phone_No*, *Student_Name* → *Department*, *Student_Name Course* → *Grade*}. Here the nonprime attribute *Phone_No* is not fully functionally dependent on the key but only on part of the key, namely the attribute *Student_Name*. Similarly, the nonprime attribute *Department* is fully functionally dependent on the attribute *Student_Name*. These are examples of partial dependencies.

Figure C The ENROLLMENT relation.

Student_Name	*Course*	*Phone_No*	*Department*	*Grade*
Jones	353	237-4539	Comp Sci	A
Ng	329	427-7390	Chemistry	A
Jones	328	237-4539	Comp Sci	B
Martin	456	388-5183	Physics	C
Dulles	293	371-6259	Decision Sci	B
Duke	491	823-7293	Mathematics	C
Duke	353	823-7293	Mathematics	B
Jones	491	237-4539	Comp Sci	C
Evan	353	842-1729	Comp Sci	A+
Baxter	379	839-0827	English	B

■

The problem with the relation ENROLLMENT is that unless the student takes at least one course, we cannot enter data for the student. Note that we cannot enter a null value for the *Course* portion of a tuple since *Course* is part of the primary key

Figure E Example of a lossy decomposition: (i) The STUDENT_ADVISOR relation; (ii) STUDENT_DEPARTMENT; (iii) DEPARTMENT_ADVISOR; and (iv) Join of STUDENT_DEPARTMENT and DEPARTMENT_ADVISOR.

Name	*Department*	*Advisor*
Jones	Comp Sci	Smith
Ng	Chemistry	Turner
Martin	Physics	Bosky
Dulles	Decision Sci	Hall
Duke	Mathematics	James
James	Comp Sci	Clark
Evan	Comp Sci	Smith
Baxter	English	Bronte

(i)

Name	*Department*
Jones	Comp Sci
Ng	Chemistry
Martin	Physics
Dulles	Decision Sci
Duke	Mathematics
James	Comp Sci
Evan	Comp Sci
Baxter	English

(ii)

Department	*Advisor*
Comp Sci	Smith
Chemistry	Turner
Physics	Bosky
Decision Sci	Hall
Mathematics	James
Comp Sci	Clark
English	Bronte

(iii)

Name	*Department*	*Advisor*
Jones	Comp Sci	Smith
Jones	Comp Sci	Clark
Ng	Chemistry	Turner
Martin	Physics	Bosky
Dulles	Decision Sci	Hall
Duke	Mathematics	James
James	Comp Sci	Smith
James	Comp Sci	Clark
Evan	Comp Sci	Smith
Evan	Comp Sci	Clark
Baxter	English	Bronte

(iv)

ADVISOR into STUDENT_DEPARTMENT*(Name, Department)* and DEPARTMENT_ADVISOR *(Department, Advisor)* is given in Figures Eii and Eiii. The join of these decomposed relations is given in Figure Eiv and contains tuples that did not exist in the original relation of part i. The decomposition is called **lossy**. ■

The terms lossless and dependency preserving are defined below.

Definition: A decomposition of a relation scheme **R** <**S**, **F**> into the relation schemes $\mathbf{R}_i$ $(1 \leq i \leq n)$ is said to be a **lossless join decomposition** or simply **lossless** if for every relation R(**R**) that satisfies the FDs in **F**, the natural join of the projections of R gives the original relation R; i.e.,

$$R = \pi_{R1}(R) \bowtie \pi_{R2}(R) \bowtie \ldots \bowtie \pi_{Rn}(R)$$

If $R \subset \pi_{R1}(R) \bowtie \pi_{R2}(R) \bowtie \ldots \bowtie \pi_{Rn}(R)$ then the decomposition is called **lossy**.[5]

The lossless join decomposition enables any relation to be recovered from its projections or decompositions by a series of natural joins. Such decomposed relations contain the same data as the original relation. Another property that the decomposition of a relation into smaller relations must preserve is that the set of functional dependencies of the original relation must be implied by the dependencies in the decompositions.

Definition: Given a relation scheme **R**<**S**, **F**> where **F** is the associated set of functional dependencies on the attributes in **S**, **R** is decomposed into the relation schemes $\mathbf{R}_1, \mathbf{R}_2, \ldots, \mathbf{R}_n$ with the functional dependencies $\mathbf{F}_1, \mathbf{F}_2, \ldots, \mathbf{F}_n$. Then this decomposition of **R** is **dependency-preserving** if the closure of **F'** (where $\mathbf{F}' = \mathbf{F}_1 \cup \mathbf{F}_2 \cup \ldots \cup \mathbf{F}_n$) is the identical to $\mathbf{F}^+$ (i.e., $\mathbf{F}'^+ = \mathbf{F}^+$).

If we decompose a relation into relation schemes that do not preserve dependencies, the enforcement of the original FDs can only be accomplished by joining the decomposed relation. This operation has to be done for each update for verifying consistency. Note that the dependencies in the decomposition are always implied by the original set of FDs.

These observations are summarized in the following theorem; we will not give a formal proof of this theorem but illustrate it with examples. Formal proofs can be found in the references given in the bibliographic notes at the end of the chapter.

Theorem 6.1: A decomposition of relation scheme **R** <(**X**, **Y**, **Z**), **F**> into $\mathbf{R}_1$<(**X**, **Y**), $\mathbf{F}_1$> and $\mathbf{R}_2$<(**X**, **Z**), $\mathbf{F}_2$ $\mathbf{R}_2$ < (**X**,**Z**), $\mathbf{F}_2$ > is:

(a) dependency preserving if every functional dependency in **R** can be logically derived from the functional dependencies of $\mathbf{R}_1$ and $\mathbf{R}_2$, i.e., $(\mathbf{F}_1 \cup \mathbf{F}_2)^+ = \mathbf{F}^+$, and

(b) is lossless if the common attributes **X** of $\mathbf{R}_1$ and $\mathbf{R}_2$ form a superkey of at least one of these, i.e., $\mathbf{X} \rightarrow \mathbf{Y}$ or $\mathbf{X} \rightarrow \mathbf{Z}$.

[5] $R \subseteq \pi_{R1}(R) \bowtie \pi_{R2}(R) \bowtie \ldots \bowtie \pi_{Rn}(R)$ is always true.

Example 6.19 illustrated a decomposition that is both lossy and doesn't preserve the dependencies in the original relation. It is lossy because the common attribute *Department* is not a key of either of the resulting relations and consequently, the join of these projected relations produces tuples that are not in the original relation. The decomposition is not dependency-preserving because the FD *Name* $\rightarrow$ *Advisor* is not implied by the FDs of the decomposed relation.

Example 6.20 illustrates a lossless decomposition.

Example 6.20

Let $\mathbf{R}(A, B, C)$ and $\mathbf{F} = \{A \rightarrow B\}$. Then the decomposition of $\mathbf{R}$ into $\mathbf{R}_1(A, B)$ and $\mathbf{R}_2(A, C)$ is lossless because the FD $\{A \rightarrow B\}$ is contained in $\mathbf{R}_1$ and the common attribute A is a key of $\mathbf{R}_1$. ■

A decomposition which is lossy is given below.

Example 6.21

Let $\mathbf{R}(A, B, C)$ and $\mathbf{F} = \{A \rightarrow B\}$. Then the decomposition of $\mathbf{R}$ into $\mathbf{R}_1(A, B)$ and $\mathbf{R}_2(B, C)$ is not lossless because the common attribute B does not functionally determine either A or C, i.e. it is not a key of $\mathbf{R}_1$ or $\mathbf{R}_2$. ■

A decomposition which is both lossless and dependence preserving is given in Example 6.22.

Example 6.22

Given $\mathbf{R}(A, B, C, D)$ with the functional dependencies $\mathbf{F} = \{A \rightarrow B, A \rightarrow C, C \rightarrow D\}$, consider the decomposition of $\mathbf{R}$ into $\mathbf{R}_1(A, B, C)$ with the function dependencies $\mathbf{F}_1 = \{A \rightarrow B, A \rightarrow C\}$ and $\mathbf{R}_2(C, D)$ with the functional dependencies $\mathbf{F}_2 = \{C \rightarrow D\}$. In this decomposition all the original FDs can be logically derived from $\mathbf{F}_1$ and $\mathbf{F}_2$, hence the decomposition is dependency-preserving. Also, the common attribute C forms a key of $\mathbf{R}_2$. The decomposition of $\mathbf{R}$ into $\mathbf{R}_1$ and $\mathbf{R}_2$ is lossless. ■

Example 6.23 gives a lossy decomposition which also is not dependency preserving.

Example 6.23

Given $\mathbf{R}(A,B,C,D)$ with the functional dependencies $\mathrm{F} = \{A \rightarrow B, A \rightarrow C, A \rightarrow D\}$, the decomposition of $\mathbf{R}$ into $\mathbf{R}_1(A,B,D)$ with the functional dependencies $\mathbf{F}_1 = \{A \rightarrow B, A \rightarrow D\}$ and $\mathbf{R}_2(B,C)$ with the functional dependencies $\mathbf{F}_2 = \{\}$ is lossy because the common attribute B is not a candidate key of either $\mathbf{R}_1$ or $\mathbf{R}_2$. In addition, the FD $A \rightarrow C$ is not implied by any FDs in $\mathbf{R}_1$ or $\mathbf{R}_2$. Thus, the decomposition is not dependency-preserving. ■

Now let us consider an example involving the decomposition of relations from the familiar university-related database. This decomposition, while lossless, is not dependency-preserving.

Example 6.24 Consider the relation scheme **CONCENTRATION** {*Student*(*S*), *Major_or_Minor*(M_m), *Field_of_Study*(F_s), *Advisor*(*A*)} with the functional dependencies **F** = {$(S, M_m, F_s) \rightarrow A$, $A \rightarrow F_s$}. Figure Fi illustrates some instances of tuples of a relations on this relation scheme. This relation can be decomposed by projection into the relation schemes **SM**$_m$**A**(S, M_m, A) and **F**$_s$**A**(F_s, A). The decomposition of the relation of part i into these two relations is shown in parts ii and iii. This decomposition is lossless because the common attribute A determines F_s. However, the decomposition does not preserve the dependencies; the only nontrivial dependency in the decomposition is $A \rightarrow F_s$, but it does not imply the dependency $(S, M_m, F_s) \rightarrow A$. This is an example of a decomposition that is lossless but not dependency-preserving.

Figure F Example of a lossless decomposition that is not dependency preserving: (i) The CONCENTRATION relation; (ii) The SM$_m$A relation; and (iii) The F$_s$A relation.

Student	*Major_or_Minor*	*Field_of_Study*	*Advisor*
Jones	Major	Comp Sci	Smith
Jones	Minor	Mathematics	Jamieson
Ng	Major	Chemistry	Turner
Ng	Minor	Comp Sci	Clark
Ng	Minor	Physics	Bosky
Martin	Major	Physics	Bosky
Martin	Minor	Chemistry	Turner
James	Major	Physics	Newton
James	Minor	Comp Sci	Clark

(i)

Student	M_m	*Advisor*
Jones	Major	Smith
Jones	Minor	Jamieson
Ng	Major	Turner
Ng	Minor	Clark
Ng	Minor	Bosky
Martin	Major	Bosky
Martin	Minor	Turner
James	Major	Newton
James	Minor	Clark

(ii)

Field_of_Study	*Advisor*
Comp Sci	Smith
Mathematics	Jamieson
Chemistry	Turner
Comp Sci	Clark
Physics	Bosky
Physics	Newton

(iii)

	A	B	C	D	E
R_1	α_A	α_B	α_C	β_{1D}	β_{1E}
R_2	β_{2A}	α_B	α_C	α_D	β_{2E}
R_3	β_{3A}	β_{3B}	α_C	α_D	α_E

	A	B	C	D	E
R_1	α_A	α_B	α_C	α_D	β_{1E}
R_2	β_{2A}	α_B	α_C	α_D	β_{2E}
R_3	β_{3A}	β_{3B}	α_C	α_D	α_E

any changes to the table. When we consider the FD $C \rightarrow D$, we find that all rows of the column C, the determinant of the FD, are identical and this allows us to change the entries in the column D to α_D. No further changes are possible and the final version of the table is the same as the table on the right. Finally we find no rows in the table with all αs and conclude that the decomposition is lossy. ■

As we discussed earlier, a decomposition is dependency-preserving if the closure of $\mathbf{F}'$ (where $\mathbf{F}' = \mathbf{F}_1 \cup \mathbf{F}_2 \cup \ldots \cup \mathbf{F}_n$) is identical to $\mathbf{F}^+$. However, the task of computing the closure is time consuming and we would like to avoid it. With this in mind, we provide below an alternate method of checking for the preservation of the dependencies. This method takes each functional dependency $\mathbf{X} \rightarrow \mathbf{Y}$ in $\mathbf{F}$ and computes the closure $\mathbf{X}'^+$ of $\mathbf{X}$ with respect to $\mathbf{F}'$. If $\mathbf{Y} \subseteq \mathbf{X}'^+$, then $\mathbf{F}' \models \mathbf{X} \rightarrow \mathbf{Y}$. If we can show that all functional dependencies in $\mathbf{F}$ are logically implied by $\mathbf{F}'$, we can conclude that the decomposition is dependency-preserving. Obviously, if even a single dependency in $\mathbf{F}$ is not covered by $\mathbf{F}'$, the decomposition is not dependency-preserving. Algorithm 6.5 checks if a decomposition is dependency-preserving.

If the union of dependencies of the decomposed relations is the same as the original set of dependencies, then the decomposition is dependency-preserving. This is illustrated in the following example.

Example 6.27

Consider $\mathbf{R}(A,B,C,D)$ with the functional dependencies $\mathbf{F}$ $\{A \rightarrow B, A \rightarrow C, C \rightarrow D\}$ and its decomposition into $\mathbf{R}_1(A,B,C)$ with the functional dependencies $\mathbf{F}_1 = \{A \rightarrow B, A \rightarrow C\}$ and $\mathbf{R}_2(C,D)$ with the functional dependencies $\mathbf{F}_2 = \{C \rightarrow D\}$. This decomposition is dependency-preserving because all the original FDs can be logically derived from $\mathbf{F}_1$ and $\mathbf{F}_2$. (In this case each FD in $\mathbf{F}$ is included in $\mathbf{F}'$ (where $\mathbf{F}' = \mathbf{F}_1 \cup \mathbf{F}_2$).) ■

The following example illustrates a decomposition which is not dependency-preserving.

Algorithm 6.5 Algorithm to Check If a Decomposition is Dependency Preserving

Input: A relation scheme and a set F of functional dependencies; a projection (R_1, R_2, . . ., R_n) of R with the functional dependencies (F_1, F_2, . . ., F_n).

Output: Whether the decomposition is dependency-preserving or not.

```
F'+_=_F+ := true; (*Assume F'+_=_F+, used as a variable, is true *)
F' := φ;
for i := 1 to n do
   F' := F' ∪ Fi;
for each FD X → Y ∈ F and while (F'+_=_F+) do
      (* compute X'+, the closure of X under F', using Algorithm 6.1 ))
      if Y ⊄ X'+ then F'+_=_F+ = false; (* i.e., the decomposition is not
            dependency-preserving *);
```

Example 6.28 R(*A*,*B*,*C*,*D*) with the functional dependencies F {*A* → *B*, *A* → *C*, *A* → *D*} is decomposed into R_1(*A*,*B*,*D*) with the functional dependencies F_1 = {*A* → *B*, *A* → *D*} and R_2(*B*,*C*) with the functional dependencies F_2 = {}. This is not dependency-preserving because the FD *A* → *C* is not implied by any FDs in R_1 or R_2. ■

Now let us consider the decomposition of a relation from the university database.

Example 6.29 Consider the relation STUDENT_ADVISOR(*Name, Department, Advisor*) of Figure Ei with the functional dependencies F = {*Name* → *Department, Name* → *Advisor, Advisor* → *Department*}. Here, the decomposition of STUDENT_ADVISOR into STUDENT_PROFESSOR(*Name, Advisor*) with the functional dependency {*Name* → *Advisor*}, and DEPARTMENT_ADVISOR(*Department, Advisor*) with the functional dependency {*Advisor* → *Department* is dependency-preserving, because the dependency *Name* → *Department* is implied by *(Name* → *Advisor)* ∪ *(Advisor* → *Department)*; in addition, the decomposition is lossless. ■

On the other hand, the following decomposition is not dependency-preserving.

Example 6.30 The decomposition of the relation CONCENTRATION of Figure F into the relations SM_mA and F_sA is not dependency-preserving because F′ = *A* → F_s and the FD SM_mF_s → *A* is not implied by F′. ■

canonical cover; this caters to any possible many-to-many association between these attributes.

Algorithm for Lossless and Dependency-Preserving Third Normal Form Decomposition

For this algorithm we assume that we have a canonical cover $\mathbf{F}_c$ for the set of FDs **F** for the relation scheme **R** and that **K** is a candidate key of **R**. Algorithm 6.6 produces a decomposition of **R** into a collection of relation schemes $\mathbf{R}_1, \mathbf{R}_2, \ldots, \mathbf{R}_n$. Each relation scheme $\mathbf{R}_i$ is in third normal form with respect to the projection of $\mathbf{F}_c$ onto the scheme of **R**.

In Example 6.31 below, we give a decomposition into 3NF relation schemes which is both lossless and also dependency-preserving.

Example 6.31

Find a lossless join and dependency-preserving decomposition of the following relation scheme with the given set of functional dependencies:

SHIPPING *(Ship, Capacity, Date, Cargo, Value)*
Ship $\rightarrow$ *Capacity,*
ShipDate $\rightarrow$ *Cargo,*
CargoCapacity $\rightarrow$ *Value*

First find the canonical cover of the given set of FDs. The FDs are simple since each has a single attribute on the right-hand side. There are no redundant FDs in the set and none of the FDs contains extraneous attributes on the left-hand side. Hence the given set of FDs is in canonical form. A candidate key of the relation is *ShipDate*.

Now use Algorithm 6.6 to find a lossless and dependency-preserving decomposition of **SHIPPING**. Since all attributes appear in the canonical cover we need not form a relation for attributes not appearing in any FD. There is no single FD in the canonical cover that contains all remaining attributes in **SHIPPING**, so we proceed to form a relation for each FD in the canonical cover.

$\mathbf{R}_1$*(Ship, Capacity)* with the FD *Ship* $\rightarrow$ *Capacity*
$\mathbf{R}_2$*(Ship, Date, Cargo)* with the FD *ShipDate* $\rightarrow$ *Cargo*
$\mathbf{R}_3$*(Cargo, Capacity, Value)* with the FD *CargoCapacity* $\rightarrow$ *Value*

As a candidate key is included in the determinant of the FD of the decomposed relation scheme $\mathbf{R}_2$, we need not include another relation scheme with only a candidate key. The decomposition of **SHIPPING** into $\mathbf{R}_1$, $\mathbf{R}_2$, and $\mathbf{R}_3$ is both lossless and dependency-preserving. ■

In Example 6.32 we find a 3NF decomposition of a relation from the university database.

Example 6.32 Consider the relation scheme **STUDENT_INFO**(*Student*(S), Major(M), *Student_Department*(S_d), *Advisor*(A), *Course*(C), *Course_Department*(C_d), *Grade*(G), *Professor*(P), *Prof_Department*(P_d), *Room*(R), *Day*(D), *Time*(T)) with the following functional dependencies:

$S \rightarrow M$	each student is in an unique major
$S \rightarrow A$	each student has an unique advisor
$M \rightarrow S_d$	each major is offered in an unique department
$S \rightarrow S_d$	each student is in one department
$A \rightarrow S_d$	each advisor is in an unique department
$C \rightarrow C_d$	each course is offered by a single department
$C \rightarrow P$	each course is taught by one professor
$P \rightarrow P_d$	each professor is in an unique department
$RTD \rightarrow C$	each room has on a given day and time only one course scheduled in it
$RTD \rightarrow P$	each room has on a given day and time one professor teaching it it
$TPD \rightarrow R$	a given professor on a given day and time is in one room
$TSD \rightarrow R$	a given student on a given day and time is in one room
$TDC \rightarrow R$	a course can be in only one room on a given day and time
$TPD \rightarrow C$	on a given day and time a professor can be teaching only one course
$TSD \rightarrow C$	on a given day and time a student can be attending only one course
$SC \rightarrow G$	each student in a given course has a unique grade

A canonical cover of this set of functional dependencies will not contain the dependencies $\{S \rightarrow S_d, RTD \rightarrow P, TDC \rightarrow R, TPD \rightarrow C, TSD \rightarrow R\}$. The key of this relation scheme is TSD. The decomposition of this relation scheme into third normal form gives the following relation schemes:

$\mathbf{R}_1(SMA)$	with the FD $S \rightarrow MA$
$\mathbf{R}_2(MS_d)$	with the FD $M \rightarrow S_d$
$\mathbf{R}_3(AS_d)$	with the FD $A \rightarrow S_d$
$\mathbf{R}_4(CC_dP)$	with the FD $C \rightarrow C_dP$
$\mathbf{R}_5(PP_d)$	with the FD $P \rightarrow P_d$
$\mathbf{R}_6(RTDC)$	with the FD $RTD \rightarrow C$
$\mathbf{R}_7(TPDR)$	with the FD $TPD \rightarrow R$
$\mathbf{R}_8(TSDR)$	with the FD $TSD \rightarrow R$
$\mathbf{R}_9(SCG)$	with the FD $SC \rightarrow G$

(Note: Since all the attributes in the original relation scheme are involved with some FD we do not have to create a relation scheme with attributes not so involved. Also, the relation scheme $\mathbf{R}_8$ includes a candidate key; consequently we don't need to create an explicit relation scheme for the key.)
$\mathbf{R}_1$ through $\mathbf{R}_9$ form a lossless and dependency-preserving decomposition of **STUDENT_INFO** ■

Derivation of other canonical covers of this set of FDs and the corresponding relational schemes in 3NF is left as an exercise.

Algorithm 6.7

Lossless Boyce Codd Normal Form Decomposition Algorithm

Input: A relation scheme **R<U, F>** not in BCNF where **F** is a set of FD.

Output: Decomposition of **R(U)** into relation schemes $\mathbf{R_i(U_i)}$, $1 \le i \le n$ such that each $\mathbf{R_i(U_i)}$ is in BCNF and the decomposition is lossless.

```
begin
i := 0;
S := {R(U)};
all_BCNF := false;
Find F' from F; (* here F' is a nonredundant cover of F *)
while (¬ all_BCNF) do
   if there exists a nontrivial FD (X → Y) in F'+ such that
      XY ⊆ Rj and X ↛ Rj   (* Rj, a relation scheme in S, is not in BCNF,
                               i.e., X → Rj is not in F'+ *)
   then
     begin
        i := i+1;
        form relation Ri{X, Y} with the FD X → Y and add
          it to S
        Rj := Rj − Y;
     end;
   else  all_BCNF := true;
end;
```

may be lost. Also, the relation schemes so produced are not unique. The resulting set of decomposed schemes depends on the order in which the functional dependencies in the original relation is used.

We use Algorithm 6.7 to find BCNF decomposition of a number of relations in Examples 6.35 through 6.37.

Example 6.35

Find a BCNF decomposition of the relation scheme **SHIPPING** with the following set of functional dependencies:

SHIPPING(*Ship, Capacity, Date, Cargo, Value*)
Ship → *Capacity*
ShipDate → *Cargo*
CargoCapacity → *Value*

First find the nonredundant cover of the given set of FDs. There are no redundant FDs in the set, hence the given set of FDs is a nonredundant cover.

Now use Algorithm 6.7 to find a lossless decomposition of **SHIPPING.** Since there is an FD $Ship \rightarrow Capacity$ and since $Ship \nrightarrow$ **SHIPPING** we replace **SHIPPING** with the relation $\mathbf{R}_1$ *(Ship, Capacity)* formed with the FD in question and $\mathbf{R}_2$*(Ship, Date, Cargo, Value)*. Consider the relation $\mathbf{R}_2$: the FD $ShipDate \rightarrow Cargo$ is a nontrivial FD in the nonredundant cover. However, since $ShipDate \rightarrow ShipDateCargoValue$, the relation $\mathbf{R}_2$ is in BCNF and we have completed the decomposition.

$\mathbf{R}_1$*(Ship, Capacity)* with the FD $Ship \rightarrow Capacity$
$\mathbf{R}_2$*(Ship, Date, Cargo, Value)* with the FD $ShipDate \rightarrow Cargo$

The decomposition of **SHIPPING** into $\mathbf{R}_1$ and $\mathbf{R}_2$ is lossless but not dependency preserving because the FD $CargoCapacity \rightarrow Value$ is not implied by the set of FDs $\{Ship \rightarrow Capacity, ShipDate \rightarrow Cargo\}$.

Another BCNF decomposition of **SHIPPING** is obtained when we consider the FD $CargoCapacity \rightarrow Value$ first. This gives us the following decompositions:

$\mathbf{R}_1$*(Cargo, Capacity, Value)* with the FD $CargoCapacity \rightarrow Value$
$\mathbf{R}_2$*(Ship, Capacity)* with the FD $Ship \rightarrow Capacity$
$\mathbf{R}_3$*(Ship, Date, Cargo)* with the FD $ShipDate \rightarrow Cargo$

This decomposition is also dependency-preserving. ■

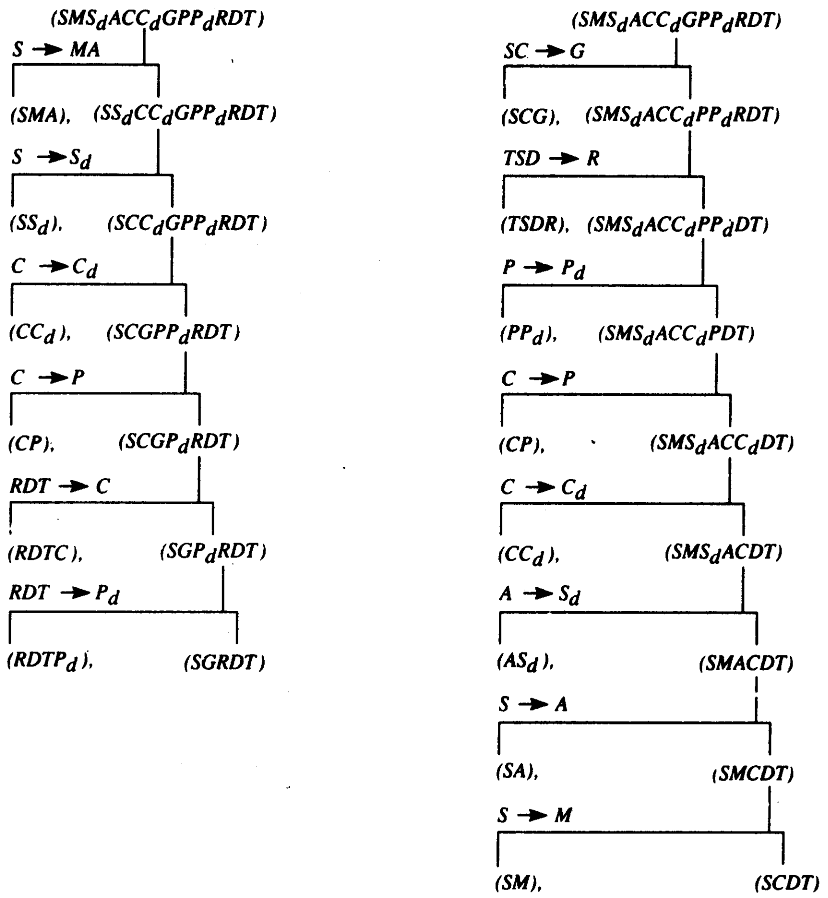
An example of a BCNF decomposition which is not dependency preserving is given below.

Example 6.36 Consider the relation scheme $<(ABCD), \{AB \rightarrow C, C \rightarrow A\}>$. None of the FDs are redundant, so the given set is a nonredundant cover. Using the FD $AB \rightarrow C$ we decompose this into the relation schemes: $<(ABC), \{AB \rightarrow C, C \rightarrow A\}>$ and $<(ABD), \{\ \}>$. The scheme $<(ABC), \{AB \rightarrow C, C \rightarrow A\}>$ can be further decomposed into the schemes $<(AC), \{C \rightarrow A\}>$ and $<(BC), \{\ \}>$. ■

In Example 6.37, we demonstrate the non-uniqueness of the BCNF decomposition.

Example 6.37 Consider the relation scheme **STUDENT_INFO**$\{S, M, S_d, A, C, C_d, G, P, P_d, R, D, T\}$ with the following functional dependencies $(S \rightarrow MA, M \rightarrow S_d, A \rightarrow S_d, C \rightarrow C_dP, P \rightarrow P_d, RDT \rightarrow C, TPD \rightarrow R, TSD \rightarrow R, SC \rightarrow G)$. The key of this relation is *TSD*. The decomposition of this relation into a number of BCNF relation schemes using Algorithm 6.7 gives the decomposition tree shown in Figure G. The left tree is obtained by considering the FDs in the order $S \rightarrow MA$, $S \rightarrow S_d$, $C \rightarrow C_d$, $C \rightarrow P$, and $RDT \rightarrow C$. This order gives the following set of BCNF relation schemes: (SMA), (SS_d), (CC_d), (CP), $(RDTC)$, and (SGP_dRDT). The right decomposition is obtained by considering the FD $SC \rightarrow G$ first.

Figure G Two Different Decomposition Trees.



■

We see from the above example that for different orders of considering the FDs, we get different decomposition trees and hence different sets of resulting relation schemes. For Example 6.37, we illustrate in Figure G two different decomposition trees giving the following sets of relations: {*(SMA)*, *(SS$_d$)*, *(CC$_d$)*, *(CP)*, *(RDTC)*, *(P$_d$RDT)*, *(SGRDT)*} and {*(SCG)*, *(TSDR)*, *(PP$_d$)*, *(CP)*, *(CC$_d$)*, *(AS$_d$)*, *(SA)*, *(SM)*, *(SCDT)*}.

One other point we notice is that some of the original dependencies are no longer preserved in the decompositions given above. For instance, in both sets of relation schemes, the FD $M \rightarrow S_d$ is no longer represented. This means that we cannot ascertain, without one or more joins, that the corresponding fact is correctly represented in the database. At each step of the algorithm we are decomposing a relation into two relations, such that the common attribute is a key of one of these relations. Consequently, the decomposition algorithm produces a set of lossless BCNF relations.

We conclude with the observation that there are relation schemes **R**<**S**, **F**> such that no decomposition of **R** under **F** is dependency-preserving. This is a worse situation than one where some decompositions are dependency-preserving while others are not.

6.6 Concluding Remarks

Let us return to the relation STUDENT_ADVISOR(*Name, Department, Advisor*) of Figure Ei with the functional dependencies **F** = {*Name* → *Department, Name* → *Advisor, Advisor* → *Department*}. When we decomposed STUDENT_ADVISOR into STUDENT_DEPARTMENT(*Name, Department*), and DEPARTMENT_ADVISOR (*Department, Advisor*), giving the relations shown in Figures Eii and Eiii, we found that the decomposition was lossy. The common attribute, *Department*, is not a key of either of the decomposed relations. The join of these decomposed relations, given in Figure Eiv, contains tuples that did not exist in the original relation of Figure Ei. In addition the decomposition is not dependency-preserving. The FD *Name* → *Advisor* is not implied by the FDs of the decomposed relation nor could it be derived from their join.

We notice, however, that there are three independent relationships in the STUDENT_ADVISOR relation, and the only key is *NameAdvisor*. We can decompose it into three relations, ADVISOR_STUDENT(*Name, Advisor*), STUDENT_DEPARTMENT(*Name, Department*), and ADVISOR_DEPARTMENT(*Advisor, Department*). This decomposition is useful in storing the independent relationships autonomously. The original relation can be obtained by joining these decomposed relations. The decomposition is lossless since the common attribute in these relations is a key of one of them. Furthermore, the decomposition is dependency preserving since each of the FDs is preserved in one of the relations.

Note that some of these independent relationships that are not involved with each other will be eliminated from the final result. For instance, a new student, Letitia, may join the Physics department without having an advisor. Similarly, a new professor, Jaffe, may join the Chemistry department and may not yet be advising students. The resulting relations are shown in parts a, b, and c of Figure 6.16. In the original relation, this data could only have been entered with null values for the unknown attribute.

The join of these relations to obtain the STUDENT_ADVISOR relation gives us the tuples shown in Figure E. The new tuples added in the decomposed relation participate in one of the joins, as shown in Figure Ed. However, these and other extraneous tuples are eliminated when the second join is performed. The tuples (Letitia, Physics) of STUDENT_DEPARTMENT and (Jaffe, Chemistry) of ADVISOR_DEPARTMENT are eliminated for this sequence of joins. Such tuples, which do not contribute to the result of the join operations, are called **dangling tuples.**

When we refer to the attributes *Name, Advisor*, and *Department* in a database containing the above three relations, we need to distinguish the various applications of the same symbol. A simple method of doing this is by preceding the attribute with the name of the relation. Another approach would be to use unique identifiers for each role that the attribute plays in the model.

Figure 6.16 Join of the decomposition of STUDENT_ADVISOR: (a) ADVISOR_STUDENT; (b) STUDENT_DEPARTMENT; (c) ADVISOR_DEPARTMENT; and (d) X = STUDENT DEPARTMENT $\bowtie$ ADVISOR_DEPARTMENT. Note: The marked tuples are eliminated when this result relation, X, is joined with ADVISOR_STUDENT, i.e., STUDENT ADVISOR = ADVISOR_STUDENT $\bowtie_{Name,Advisor}$ X.

Name	*Advisor*
Jones	Smith
Ng	Turner
Martin	Bosky
Dulles	Hall
Duke	James
James	Clark
Evan	Smith
Baxter	Bronte

(a)

Name	*Department*
Jones	Comp Si
Ng	Chemistry
Martin	Physics
Dulles	Decision Sci
Duke	Mathematics
James	Comp Sci
Evan	Comp Sci
Letitia	Physics
Baxter	English

(b)

Advisor	*Department*
Smith	Comp Sci
Turner	Chemistry
Bosky	Physics
Hall	Decision Sci
James	Mathematics
Clark	Comp Sci
Bronte	English
Jaffe	Chemistry

(c)

Name	*Department*	*Advisor*	
Jones	Comp Sci	Smith	
Jones	Comp Sci	Clark	<
Ng	Chemistry	Turner	
Ng	Chemistry	Jaffe	<
Martin	Physics	Bosky	
Dulles	Decision Sci	Hall	
Duke	Mathematics	James	
James	Comp Sci	Smith	<
James	Comp Sci	Clark	
Evan	Comp Sci	Smith	
Evan	Comp Sci	Clark	<
Letitia	Physics	Brosky	<
Baxter	English	Bronte	

(d)

The goal of database design is to ensure that the data is represented in such a way that there is no redundancy and no extraneous data is generated. This means that we would generate relations in as high an order as possible. Since we cannot always guarantee that the BCNF relations will be dependency preserving when both lossless and dependency-preserving relations are required, we have to settle for the third normal form.

6.7 Summary

In this chapter we studied the issues involved in the design of a database application using the relational model. We discussed the importance of having a consistent database without repetition of data and pointed out the anomalies that could be introduced in a database with an undesirable design. The criteria to be addressed by the design process are redundancy, insertion anomalies, deletion anomalies, and update anomalies.

A relation scheme **R** is a method of indicating the attribute names involved in a relation. In addition the relation scheme **R** has a number of constraints that have to be satisfied to reflect the real world being modeled by the relation. These constraints are in the form of FDs. The approach we have used is to replace R by a set of more desirable relation schemes. In this chapter we considered the decomposition approach. The synthesis approach is discussed in Chapter 7.

The decomposition approach starts with one relation (the universal relation) and the associated set of constraints in the form of functional dependencies. The relation has a certain number of undesirable properties (in the form of insertion, deletion, or update anomalies) and it is replaced by its projections. A number of desirable forms of projections have been identified. In this chapter we discussed the following normal forms: 1NF, 2NF, 3NF, BCNF.

Any relation having constraints in the form of FDs only can be decomposed into relations in the third normal form; such a decomposition is lossless and preserves the dependencies. Any relation can also be decomposed losslessly into relations in the Boyce Codd normal form (and hence into the third normal form). However, such decomposition into the Boyce Codd normal form may not be dependency-preserving. The goal of the decomposition approach to the relational database design using FDs is to come up with a database scheme that is in BCNF, is lossless, and preserves the original set of FDs. If this goal is not possible, an alternate goal is to derive a database scheme that is in 3NF and is lossless and dependency-preserving.

Key Terms

decomposition
universal relation
universal relation assumption
spurious tuple
trivial functional dependency
closure
cover
nonredundant cover
simple
canonical cover
minimal
full functional dependency
prime attribute
nonprime attribute
partial dependency
transitive dependency
synthesis
content preserving
dependency-preserving
interrelation join constraints
unnormalized
nonatomic value
normalized
first normal form (1NF)
second normal form (2NF)
third normal form (3NF)
lossless join decomposition
lossless
lossy
Boyce Codd normal form (BCNF)
dangling tuple

Exercises

6.1 Given **R**$\{ABCDE\}$ and **F** $= \{A \rightarrow B,\ BC \rightarrow D,\ D \rightarrow BC,\ DE \rightarrow \phi\}$. are there any redundant FDs in **F**? If so, remove them and decompose the relation **R** into 3NF relations.

6.2 Given **R**$\{ABCDE\}$ and the set of FDs on **R** given by **F** $= \{AB \rightarrow CD,\ ABC \rightarrow E,\ C \rightarrow A\}$, what is $\mathbf{X}^+$, where $\mathbf{X} = \{ABC\}$? What are the candidate keys of **R**? In what normal form is **R**?

6.3 Given **R**$\{ABCDEF\}$ and the set of FDs on **R** given by **F** $= \{ABC \rightarrow DE,\ AB \rightarrow D,\ DE \rightarrow ABCF,\ E \rightarrow C\}$, in what normal form is **R**? If it is not in 3NF, decompose **R** and find a set of 3NF projections of **R**. Is this set lossless and dependency-preserving?

6.4 Given the relation scheme **R**$\{$*Truck(T), Capacity (C), Date (Y), Cargo(G), Destination (D), Value(V)*$\}$ with the following FDs $\{T \rightarrow C,\ TY \rightarrow G,\ TY \rightarrow D,\ CG \rightarrow V\}$, is the decomposition of **R** into **R1**$\{TCD\}$ and **R2**$\{TGDVY\}$ dependency-preserving? Justify. Is this decomposition lossless? Justify. Find a lossless join and dependency-preserving decomposition of **R** into 3NF. If the 3NF decomposition is not in BCNF, find a BCNF decomposition of **R**.

6.5 Consider a relation scheme **R** with the following set of attributes and FDs: {*SID, Name, Date_of_Birth, Advisor, Department, Term, Year, Course, Grade*}, {*SID* $\rightarrow$ *NameDate_of_BirthAdvisorDepartment, Advisor* $\rightarrow$ *Department. SIDTermYearCourse* $\rightarrow$ *Grade*}. Find the candidate keys of **R**. Does a dependency-preserving and lossless join decomposition of **R** into a number of BCNF schemes exist? If so, find one such decomposition. Suppose **R** is decomposed into the relation schemes {*SID, Name, Date_of_Birth*}, {*SID, Advisor, Department*}, and {*SID, Term, Year, Course, Grade*}. Does this decomposition exhibit any redundancies or anomalies?

6.6 Prove that every set of functional dependencies **F** is covered by a set of simple functional dependencies **G**, wherein each functional dependency has no more than one attribute on the right-hand side.

6.7 Given the set of functional dependencies $\{A \rightarrow BCD,\ CD \rightarrow E,\ E \rightarrow CD,\ D \rightarrow AH,\ ABH \rightarrow BD,\ DH \rightarrow BC\}$, find a nonredundant cover. Is this the only nonredundant cover?

6.8 Given **R**$\{ABCDEFGH\}$ with the FDs $\{A \rightarrow BCDEFGH,\ BCD \rightarrow AEFGH,\ BCE \rightarrow ADEFGH,\ CE \rightarrow H,\ CD \rightarrow H\}$, find a BCNF decomposition of **R**. Is it dependency-preserving?

6.9 Given **R** $<\{A, B, C, D, E, F, G, H, I, J, K\}$, $\{I \rightarrow K,\ AI \rightarrow BFG,\ IC \rightarrow ADE,\ BIG \rightarrow CJ,\ K \rightarrow HA\}$, find a canonical cover of this set of FDs. Find a dependency-preserving and lossless join 3NF decomposition of **R**. Is there a BCNF decomposition of **R** that is both dependency-preserving and also lossless? If so, find one such decomposition.

6.10 Given the relation **R** $\{ABCDE\}$ with the FDs $\{A \rightarrow BCDE,\ B \rightarrow ACDE,\ C \rightarrow ABDE\}$, give the lossless decomposition of **R**.

6.11 Give an efficient algorithm to compute the closure of **X** under a set of FDs, using the scheme outlined in the text.

6.12 Does another canonical cover of the set of FDs of Example 6.32 exist? If so, derive it and show the corresponding relation schemes.

6.13 Given the relation **R** $\{ABCDEF\}$ with the set H $= \{A \rightarrow CE,\ B \rightarrow D,\ C \rightarrow ADE,\ BD \rightarrow F\}$, find the closure of BCD.

6.14 Explain why there is renewed interest in unnormalized relations (called the non_1NF or NFNF). What are its advantages compared to normalized relations?

6.15 Discuss the advantages and disadvantages of representing hierarchical structured data from the real world as an unnormalized relation.

6.16 The Sky-High-Returns Mutual Fund (SMF) Corp. offers a number of different no-load mutual funds (F) for investment. It sells directly to the public through a number of branches (B). Each customer (C) is assigned to an agent (A) who is an employee of SMF and works out of only one branch. Any customer is allowed to buy any number of units (U) of any of the funds. Each fund is managed out of one of the branches and the portfolio (P) of the fund is directed by a board of managers (M). The board is made up of agents of SMF; however, agents from different branches may be involved in any number of boards at any branch. The unit value of each fund is decided at the end of the last business day of the month and all purchases and redemptions are done only after the unit price is determined at that time. The funds are charged a 5% per year management fee; the agents get 1% of this fee in addition to their regular salaries. Determine the entities and their attributes that have to be maintained if SMF is to design a database system to support its operations. What are the dependencies that have to be enforced? Make any additional assumptions that you may require.

6.17 Consider the TEACHES relation. Assume that $Room\_Cap \nrightarrow Enrol\_Lmt$. This means that two different courses allocated to the same room at different day and time could have different *Enrol_Lmts*. In what normal form is TEACHES under this modified assumption? If it is not in 3NF form, find a lossless and dependency-preserving decomposition.

6.18 Consider the relation scheme **R**$(ABCDE)$ and the FDs $\{A \rightarrow B, C \rightarrow D, A \rightarrow E\}$. Is the decomposition of **R** into (ABC), (BCD), (CDE) lossless?

6.19 Find a 3NF decomposition of the following relation scheme: *(Faculty, Dean, Department, Chairperson, Professor, Rank, Student)*. The relation satisfies the following functional dependencies (and any others that are logically implied by these):

$$\begin{aligned}
&Faculty \rightarrow Dean\\
&Dean \rightarrow Faculty\\
&Department \rightarrow Chairperson\\
&Professor \rightarrow RankChairperson\\
&Department \rightarrow Faculty\\
&Student \rightarrow DepartmentFacultyDean\\
&ProfessorRank \rightarrow DepartmentFaculty
\end{aligned}$$

6.20 What are the design goals of a good relational database design? Is it always possible to achieve these goals? If some of these goals are not achievable, what alternate goals should you aim for and why?

6.21 Use Algorithm 6.4 to determine if the decomposition of STUDENT_ADVISOR*(Name, Department, Advisor)* with the functional dependencies **F**$\{Name \rightarrow Department, Name \rightarrow Advisor, Advisor \rightarrow Department\}$ into ADVISOR_STUDENT*(Name, Advisor)*, STUDENT_DEPARTMENT *(Name, Department)*, and ADVISOR_DEPARTMENT*(Advisor, Department)* is lossless.

6.22 Consider the relation scheme **R**(A, B). With no information about the FDs involved, can you determine its normal form? Justify your answer.

6.23 Consider the relation scheme **R**(A, B, C, D) where A is a candidate key. With no information about the FDs involved, can you determine its normal form? Justify your answer.

Chapter

7

Synthesis Approach and Higher Order Normal Form

Contents

The first, second, third, and Boyce Codd normal forms and algorithms for converting a relation in first normal form into higher order normal forms were discussed in Chapter 6. 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 decomposition approach and present the synthesis approach to database design in Section 7.2. We then turn our attention to the higher order normal forms, examining the concept of multivalued dependency and axioms that involve both functional dependencies and multivalued dependencies. We discuss fourth normal form and a lossless decomposition algorithm for it. Next we introduce the concept of join dependency and a normal form for it. Finally, we introduce a scheme whereby all general constraints can be enforced via domain and key constraints 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 that are in third normal form. Such a decomposition is lossless and preserves the dependencies. Any relation can also be decomposed losslessly into relations in Boyce Codd normal form (and hence in third normal form). However, decomposition into Boyce Codd normal form may not be dependency preserving. A case was illustrated in Example 6.37 in Chapter 6, 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 dependency preserving. In addition, the decomposition into 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 interrelational join constraints. This happens when the attributes **XY** corresponding to one of the functional dependencies **X** → **Y** do not appear in any of the decomposed relation schemes. In the decomposed relation schemes of Example 6.37, 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 Example 6.37. In general, to find out if a functional dependency **X** → **Y** is maintained in the decomposed schemes requires joining several of the decomposed relations. Since join operations are computationally expensive, interrelational join constraints are undesirable.

However, a lossless and dependency 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 dependency preserving properties may be possible.

The goal of the decomposition approach to relational database design using FDs is to come up with a database scheme that is in BCNF, is lossless, and preserves the original set of FDs. If this goal is not achieved the alternate goal is to derive a database scheme that is in 3NF and is lossless and dependency preserving.

$\mathbf{R}_1(ABC)$	with key A
$\mathbf{R}_2(CDE)$	with key CD
$\mathbf{R}_3(EC)$	with key E
$\mathbf{R}_4(DAEH)$	with key D
$\mathbf{R}_5(ABDH)$	with key ABH
$\mathbf{R}_6(DHBC)$	with key BC

However, **F** contains redundant FDs $CD \rightarrow E$ 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 FDs used in the synthesis approach are left reduced, i.e., there are no extraneous attributes on the left-hand side of the FDs, then we will not introduce any partial dependencies in the relations synthesized using such FDs.

Example 7.2

Consider $\mathbf{U}\{A,B,C,D\}$ with the set of FDs $\mathbf{F} = \{ABC \rightarrow D, A \rightarrow C\}$. 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 $A \rightarrow C$ were used to left reduce $ABC \rightarrow D$, we replace the latter by $AB \rightarrow D$ and hence obtain a synthesized design in the 3NF. ■

If two or more FDs have determinants that 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, written as $\mathbf{X} \longleftrightarrow \mathbf{Y}$. 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 FDs. Such a strategy produces an economic relational design.

Example 7.3

Let us return to the universal relation $\mathbf{U}\{A, B, C, D, E, H\}$ and the set of FDs $\mathbf{F} = \{A \rightarrow BC, CD \rightarrow E, E \rightarrow C, D \rightarrow AEH, ABH \rightarrow BD, DH \rightarrow BC\}$. We saw that the FDs $CD \rightarrow E$ 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 FDs $AH \rightarrow D$. Now, since $D \rightarrow AH$, we get the one-to-one dependency $AH \longleftrightarrow D$. Thus, AH and D are equivalent. We can combine these equivalent keys into one relation to give 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 equivalent groups of FDs, 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 (Bern 76) and is sometimes called the Bernstein Synthesis algorithm. The algorithm starts with a universal relation and the functional dependencies to be enforced on it and produces a third normal form database scheme that is lossless and dependency preserving. The algorithm is called a synthesis algorithm because it constructs relation schemes from the FDs 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 FDs in each group is the same. Recall that an FD is left reduced if the left-hand side does not contain any extraneous attributes. The algorithm then finds compound functional dependencies $(X_1,X_2, \ldots, X_k) \rightarrow Y$ by using the equivalent determinant $X_i \longleftrightarrow X_j$ for $1 \leq i \leq k$ and $1 \leq j \leq k$. The characteristic of the compound functional dependency $(X_1, X_2, \ldots, X_k) \rightarrow Y$ is that $X_i \rightarrow X_j$ and $X_i \rightarrow Y$ for $1 \leq i \leq k$ and $1 \leq j \leq k$.

Let us illustrate the synthesis algorithm via the following example.

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

$$\begin{array}{lcl} BC & \rightarrow & A \\ FG & \rightarrow & BC \\ B & \rightarrow & D \\ C & \rightarrow & E \\ F & \rightarrow & A \\ G & \rightarrow & A \\ ABE & \rightarrow & G \\ ACD & \rightarrow & F \end{array}$$

In step 1 we find that the canonical cover of **F** includes the above FDs.

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 $(BC \rightarrow A, B \rightarrow D, \ldots ACD \rightarrow F)$.

J is $BC \rightarrow FG, FG \rightarrow BC$.

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

(BCFG)	with keys	*(BC,FG)*
(BD)	with key	*(B)*
(CE)	with key	*(C)*

If we compare the relation schemes obtained with this approach with the ones obtained in Example 6.32 using Algorithm 6.6 for the third normal form decomposition, we find that the synthesis approach gives one less scheme. Basically we have combined the FDs $RTD \rightarrow C$ 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 is an unnormalized relation showing the relation EMPLOYEE {*Employee_Name, Dependent(Name, Relationship), Position(Title, Date), Home_City, Home_Phone#*} and containing the information about employees. Each employee can have a number of dependents and would have occupied various positions in the organization. The relation has nonatomic values and hence, is not in normal form. We can normalize this relation as shown in in Figure 7.2. We see in Figure 7.2 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* − *Depen-*

Figure 7.1 Unnormalized EMPLOYEE relation.

Employee_ Name	*Dependent*		*Positions*		*Home_ City*	*Home_ Phone*
	Name	*Relationship*	*Title*	*Date*		
Jill Jones	Bill Jones	spouse	J. Engineer	05/12/84	Lynn, MA	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	Revere, MA	452-4729
			Analyst	06/06/86		
	Chloe Smith-Briggs	daughter	Programmer	09/15/83		
			Analyst	09/06/86		
	Mark Briggs-Smith	son	Programmer	09/15/83		
			Analyst	09/06/86		

Figure 7.2 Normalized 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	Lynn, MA	794-2356
Jill Jones	Bill Jones	spouse	Engineer	10/06/86	Lynn, MA	794-2356
Jill Jones	Bob Jones	son	J. Engineer	05/12/84	Lynn, MA	794-2356
Jill Jones	Bob Jones	son	Engineer	19/06/86	Lynn, MA	794-2356
Mark Smith	Ann Briggs	spouse	Programmer	09/15/83	Revere, MA	452-4729
Mark Smith	Ann Briggs	spouse	Analyst	06/06/86	Revere, MA	45204729
Mark Smith	Chloe Smith-Briggs	daughter	Programmer	09/15/83	Revere, MA	452-4729
Mark Smith	Chloe Smith-Briggs	daughter	Analyst	06/06/86	Revere, MA	452-4729
Mark Smith	Mark Briggs-Smith	son	Programmer	09/15/83	Revere, MA	452-4729
Mark Smith	Mark Briggs-Smith	son	Analyst	06/06/86	Revere, MA	452-4729

dent}. 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*}.

For a second example of an MVD, look at the SCHEDULE relation described in Chapter 6 and shown, with some slight modifications in Figure 7.3. Notice that a course is scheduled a number of times during the week, 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 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 $\twoheadrightarrow$ *RoomDayTime*

Figure 7.3 The SCHEDULE relation.

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

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

CourseDayTime $\rightarrow$ *Room*

Multivalued dependencies arise when a relation R having a nonatomic attribute is converted to a normalized form. 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 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 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* $\rightarrow\!\!\rightarrow$ *RoomDayTime*. Thus, if we exchange the {*Room, Day, Time*} value in 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 tuples t_3 and t_4 as follows:

t_3 = |Smith | 353 |A532 | 40 | mon | 1145 |
t_4 = |Smith | 353 |A534 | 40 | wed | 1245 |

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

The multivalued dependency *Course* $\rightarrow\!\!\rightarrow$ {*Room, Day, Time*} does not mean that the multivalued dependencies *Course* $\rightarrow\!\!\rightarrow$ *Room*, *Course* $\rightarrow\!\!\rightarrow$ *Day*, and *Course* $\rightarrow\!\!\rightarrow$ *Time* will hold. Thus, corresponding to 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** $\rightarrow\!\!\rightarrow$ **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, 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 any two tuples t_1 and t_2 of Figure 7.2, gives us tuples t_3 and t_4 as shown below. Tuples t_3 and t_4 are in the database, but these tuples are not the original t_1 and t_2 tuples.

t_1 = |J J|Bill J |spouse|J. Eng|05/12/84|Lynn, MA|794-2356|
t_2 = |J J|Bob J|son |Eng |10/06/86|Lynn, MA|794-2356|

t_3 = |J J|Bill J |spouse|Eng |10/06/86|Lynn, MA|794-2356|
t_4 = |J J|Bob J|son |J. Eng|05/12/84|Lynn, MA|795-2356|