An Introduction to Database Systems

2nd Edition

Bipin C. DESAI

Concordia University
Montreal

BytePress

Limit of Liability/Disclaimer of Warranty:

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

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

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

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

6. Relational Database Design

A relation in a relational database is based on a relation scheme which consists of a number of attributes. A relational database is made up of a number of relations and the relational database scheme is, in turn, consists of a number of relation schemes. In this chapter, we focus on the issues involved in the design of a database schema using the relational model. In section 6.2, the importance of having a consistent database without repetition of data is discussed and the anomalies that could be introduced in the database with an undesirable design are pointed out. The universal relation assumption is presented in section 6.3. In section 6.4, we look at some of the theoretical results from the functional dependency theory and present basic algorithms for the design process. In section 6.5, the relational database design process is presented. Effectively, this process uses the functional dependencies among attributes to arrive at their desirable groupings. The first, second, third and the Boyce Codd normal forms are discussed and algorithms for converting a relation in the first normal form into higher order normal forms are given. The synthesis approach to relational database design and higher order normal forms are discussed in Chapter 7.

6.1 Relation Scheme and Relational Design

A relation scheme **R** is a plan which indicates the attributes involved in one or more relations. The scheme consists of a set **S** of attributes $\{A_1, A_2, \ldots, A_n\}$, where attribute A_i is defined on domain $\mathbf{D_i}$ for $1 \le i \le n$. We will use **R**(**S**), or **R** if there is no confusion to indicate both the logical construction of the relation (its scheme) as well the name of this set **S** of attributes. Relation R on the relation scheme **R** is a finite set of mappings or tuples $\{t_1, t_2, \ldots, t_p\}$ such that for each $t_j \in R$, each of the attribute value $t_j(A_i)$ must be in the corresponding domain $\mathbf{D_i}$.

Example 6.1: Consider the relation SCHEDULE shown in Figure A. It contains the attributes *Prof*, *Course*, *Room*, *Max_Enrollment* (enrollment limit), *Day*, *Time*. Thus, the relation scheme for the relation SCHEDULE, say **SCHEDULE**, is *(Prof*, *Course*, *Room*, *Max_Enrollment*, *Day*, *Time*).

Prof	*Course*	*Room*	*Max_Enrollment*	*Day*	*Time*
Smith	353	A532	40	mon	1145
Smith	353	A532	40	wed	1145
Smith	351	C320	60	tue	115
Smith	351	C320	60	thu	115
Clark	355	H940	300	tue	115
Clark	355	H940	300	thu	115
Turner	456	B278	45	mon	845
Turner	456	B278	45	wed	845
Jamieson	459	D110	45	tue	1015
Jamieson	459	D110	45	thu	1015

Figure A *The SCHEDULE relation*

The domain of the attribute *Prof* (professors) is all the faculty members of the university; the domain of the attribute *Course* is the courses offered by the university; that of *Room* is all the rooms in the buildings of the university; that of *Max_Enrollment* is an integer value and indicates the maximum

enrollment in the course (which is related to the capacity of the room i.e., it should be less than or equal to the capacity of the room in which the course is scheduled*)*. The domain of *Day* is {MON, TUE, WED, THU, FRI, SAT, SUN} and that of *Time* is the possible times of day.

The characteristics of each of these domains is determined by the application involved; for example the domain of the attribute *Prof* would be a character string of a appropriate length.

The relation SCHEDULE of Figure A has ten tuples, the first one being *Prof*= Smith, *Course* = 353, *Room* = A532, *Max_Enrollment* = 40, *Day* = MON, *Time* = 1145. As mentioned earlier, the tabular representation of a relation is only for the purpose of illustration. The explicit naming of the columns of the table to show the mapping or association of an attribute and its value for a particular tuple avoids the requirement of a particular ordering of the attributes in the relation scheme and hence in the representation of the time varying tuples of the relation. We will continue to represent relations as tables. We will also write the attributes of the relation in a particular order and the tuples of the relation will be shown with the list of values for the corresponding attributes in the same order. The attribute names will be attached to the columns of the table when the tuples of a relation are shown in a tabular manner.

Since a relation is an abstraction of some portion of the real world that is being modelled in the database, and since the real world changes with time, the tuples of a relation are also time varying. Thus, tuples may be added or deleted or updated over a period of time. However, the relation scheme itself does not change. (At least until the database or part of it is reorganized.)

6.2 Anomalies in Database: A Consequence of Bad Design

Consider the following relation scheme pertaining to the information about a student maintained by an university.

STDINF(***Name***, ***Course***, *Phone_No*, *Major*, *Prof*, *Grade*)

Name	*Course*	*Phone_No*	*Major*	*Prof*	*Grade*
Jones	353	237-4539	Comp Sci	Smith	A
Ng	329	427-7390	Chemistry	Turner	B
Jones	328	237-4539	Comp Sci	Clark	B
Martin	456	388-5183	Physics	James	A
Dulles	293	371-6259	Decision Sci	Cook	C
Duke	491	823-7293	Mathematics	Lamb	B
Duke	356	823-7293	Mathematics	Bond	in prog
Jones	492	237-4539	Comp Sci	Cross	in prog
Baxter	379	839-0827	English	Broes	C

Figure 6.1 *Student Data Represented in Relation STDINF*

Figure 6.1 shows some tuples of a relation on the relation scheme **STDINF** (***Name***, ***Course***, *Phone_No*, *Major*, *Prof*, *Grade*). The functional dependencies[1] among its attributes are shown in Figure 6.2. The key of the relation is (*Name*, *Course*) and the relation has, in addition, the following functional

1 Recall the definition of functional dependency from Chapter 2 repeated here. Given attribute sets **X** and **Y** (each of which may be have one or more attributes), **Y** is said to be functionally dependent on X if, given a value for each attribute in **X**, uniquely determined the value of the attributes in **Y**. **X** is called the determinant of the functional dependency FD - and the FD is denoted as $\mathbf{X} \rightarrow \mathbf{Y}$.

dependencies {*Name* → *Phone_No*; *Name* → *Major*; ***Name*, *Course*** → *Grade*; *Course* → *Prof*}.

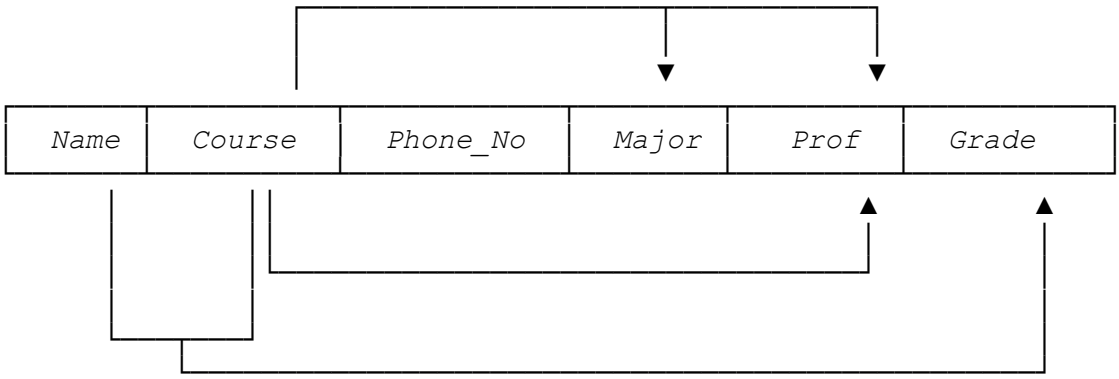


Figure 6.2 *Functional dependencies in* ***STDINF***

Here the attribute *Phone_No*, which is not in any key of the relation scheme **STDINF**, is not functionally dependent on the whole key but only on part of the key, namely, the attribute *Name*. Similarly, the attributes *Major* and *Prof*, which are not in any key of the relation scheme **STDINF**, are fully functionally dependent on the attribute *Name* and *Course* respectively. Thus the determinants of these functional dependencies are again not the entire key but only part of the key of the relation: only the attribute *Grade* is fully functionally dependent on the key (***Name***, ***Course***).

The relation scheme **STDINF** can lead to several undesirable problems as indicated below.

- **Redundancy**: The aim of the database system is to reduce redundancy, meaning that the same information is to be stored only once. If the information is stored several times then it leads to the waste of storage space and increase in the total size of the data stored. Updates to the database with such redundancies will have the potential of becoming inconsistent as explained below. In the relation of Figure 6.1, the *Major* and the phone number(*Phone_No*) of a student are stored several times in the database: once for each course that is or was taken by a student.

- **Update Anomalies**: The multiple copies of the same fact may lead to update anomalies or inconsistencies when an update is made and only some of the multiple copies are updated. Thus, the change in the *Phone_No* of Jones, for consistency, must be made in all tuples pertaining to the student Jones. If one of the three tuples of Figure 6.1 is not changed to reflect the new *Phone_No* of Jones, there will be an inconsistency in the data.

- **Insertion Anomalies**: If this is the only relation in the database showing the association between a faculty member and the course he or she teaches, then the fact that a given professor is teaching a given course cannot be entered in the database unless a student is registered in the course. Also if there is another relation which also establishes a relationship between a course and a professor who teaches that course (for example the SCHEDULE relation of Figure A), then the information stored in these relations has to be consistent.

- **Deletion Anomalies**: If the only student registered in a given course discontinues the course, then the information as to which professor is offering the course will be lost if this is the only relation in the database showing the association between a faculty member and the course she or he teaches. If there is another relation in the database which also establishes the relationship between a course and a professor who teaches that course, then the deletion of the last tuple in STDINF for

a given course will not cause the information about the course's teacher to be lost.

The problem of database inconsistency and that of redundancy of the data are similar to the ones that exist in the hierarchical and the network models. These problems are addressed in the network model by introduction of virtual fields, and in the hierarchical model the problem is resolved by the introduction of virtual records. In the relational model, the above problems can be remedied by decomposition. We define decomposition as follows:

> ***Definition***: **Decomposition**: The decomposition of a relation scheme $\mathbf{R} = (A_1, A_2, ..., A_n)$ is its replacement by a set of relation schemes $\{\mathbf{R_1}, \mathbf{R_2}, ..., \mathbf{R_m}\}$, such that $\mathbf{R_i} \subseteq \mathbf{R}$ for $1 \leq i \leq m$ and $\mathbf{R_1} \cup \mathbf{R_2} \cup .... \cup \mathbf{R_m} = \mathbf{R}$.

A relation scheme **R** can be decomposed into a collection of relation schemes $\{\mathbf{R_1}, \mathbf{R_2}, \mathbf{R_3}, ...,\mathbf{R_m}\}$ to eliminate some of the anomalies contained in the original relation **R**. Here the relations schemes $\mathbf{R_i}(1 \leq i \leq m)$ are subsets of **R** and the intersection of $\mathbf{R_i} \cap \mathbf{R_j}$, for $i \neq j$ need not be empty. Furthermore, the union of $\mathbf{R_i}$ is equal to **R,** i.e., $\mathbf{R} = \mathbf{R_1} \cup \mathbf{R_2} \cup ... \cup \mathbf{R_m}$.

The problems in the relation scheme **STDINF** can be resolved if we replace it with the following relation schemes:

STUDENT_INFO (***Name***, *Phone_No*, *Major*)
TRANSCRIPT (***Name***, ***Course***, *Grade*)
TEACHER (***Course***, *Prof*)

The first relation schemes gives the phone number and the major of each student and such information will be stored only once for each student. Any change in the phone number will thus require a change in only one tuple of this relation.

The second relation scheme stores the grade of each student in each course that the student is or was enrolled in. (Note: In our database we assume that either the student takes the course only once, or if he or she has to repeat it to improve his or her grade, then the TRANSCRIPT relation stores only the highest grade![2])

The third relation scheme records the teacher of each course.

One of the disadvantages of replacing the original relation scheme **STDINF** with the three relation schemes is that the retrieval of certain information requires a natural join operation to be performed. For instance to find the majors of student who obtained a grade of A in the course 353 requires a join to be performed: (STUDENT_INFO ⋈ TRANSCRIPT). The same information could be derived from the original relation STDINF by selection and projection.

When we replace the original relation scheme **STDINF** with the relation schemes **STUDENT_INFO**, **TRANSCRIPT**, and **TEACHER**, the consistency and referential integrity constraints have to be enforced. The referential integrity enforcement implies that if a tuple in the relation TRANSCRIPT, such as (Jones, 353, inprog) exists, then it requires that a tuple must exist in STUDENT_INFO with *Name* = Jones and, furthermore, a tuple in TEACHER must also exist with *Course* = 353. The attribute *Name* which forms part of the key of the relation TRANSCRIPT, is a key of the relation STUDENT_INFO. Such an attribute (or a group of attributes), which establishes a relationship between specific tuples (of the same or two distinct relations) is called a ***foreign key***. We notice that the attribute *Course* in relation TRANSCRIPT is also a foreign key, since it is a key of the relation TEACHER.

---

2 In these discussions, for simplicity, we have ignored the time factor, In a real applications the time factor needs to be included.

Note that the decomposition of **STDINF** into the relation schemes **STUDENT**(*Name*, *Phone_No*, *Major*, *Grade*) and **COURSE**(*Course*, *Prof*), is a bad decomposition for the following reasons:

1. Redundancy and Update Anomaly: since the data for the attributes *Phone_No* and *Major* is repeated,

2. Loss of information: we lose the fact that a student has a given grade in a particular course.

The rest of this chapter examines the problem of the design of the relational database and how to decide whether a given set of decomposed relations is better than another set.

6.3 Universal Relation

Let us consider the problem of designing a database. Such a design will be required to represent a finite number of entity sets and their relationships. Each entity set will be represented by a number of its attributes of interest for the applications to be supported by the database. If we refer to the set of all attributes as the universal scheme **U** then a relation R(**U***)* is called the ***universal relation***. The universal relation is a single relation made up of all the attributes in the database. The term ***universal relation assumption*** is the assumption that all relations in a database are derived from the universal relation by appropriate projection. The attribute names in the universal relation scheme **U** have to be distinct to avoid obvious confusion. One reason for using the universal relation assumption is to allow the user to view the database using such a relation. Consequently, the user does not have to remember the relation schemes and which attributes are grouped together in each such scheme.

Course	*Department*
353	Comp Sci
355	Mathematics
456	Mathematics
221	Decision Sci

Figure 6.3 *Relation* R_1

Professor	*Department*
Smith	Comp Sci
Clark	Comp Sci
Turner	Chemistry
Jamieson	Mathematics

Figure 6.4 *Relation* R_2

Consider the relation R_1(*Course*, *Department*) in Figure 6.3. The attribute **Department** is used to indicate the department which is responsible for the course. For instance, the course 353 is offered by and is under the jurisdiction of the Comp(uter) Sci(ence) Department.

The relation R_2(*Professor*, *Department*) of Figure 6.4 shows another interpretation of the attribute *Department*: here it is used to signify that a given professor is assigned to a given department. Thus, Smith is a member of the Comp Sci department. Note from Figures A, 6.3, and 6.4 that we are allowing for the incidence of a professor teaching a course in a outside department. Prof. Clark of the Comp Sci department is teaching course 355 of the Mathematics department, and Prof. Turner of the Chemistry department is teaching course 456, also of the Mathematics department.

The domain of the attribute *Department* in the relations R_1 and R_2 is the same, that is, all the departments in the university. Let us consider the representation of the data in the limited database indicated in Figures 6.3 and 6.4 as an universal relation U_1, where U_1 is defined as U_1(*Course*, *Department*, *Professor*). The problem of using the universal relation U_1 becomes obvious when we try to represent the data from the relations R_1 and R_2 as shown in Figures 6.3 and 6.4. Here we have to decide whether data from different relations could appear in the same tuple of the universal relation or not. In

Figure 6.5 we do not allow the data from different relations to appear in the same tuple of U_1 giving rise to a large number of empty or null values ($\perp$). These null values could signify one of three things: (i) the values are not known, but they exist, (ii) the values do not exist or (iii) the attribute does not apply. In case (i) we have to distinguish the null values by indicating them as $\perp_i$, and thus the two null values $\perp_i$ and $\perp_j$ (for $i \neq j$) are not equal and indicate that the values are not known to be the same.

Course	*Department*	*Professor*
353	Comp Sci	$\perp$
456	Mathematics	$\perp$
355	Mathematics	$\perp$
221	Decision Sci	$\perp$
$\perp$	Comp Sci	Smith
$\perp$	Comp Sci	Clark
$\perp$	Chemistry	Turner
$\perp$	Mathematics	Jamieson

Figure 6.5 *Relation U_1*

Course	*Department*	*Professor*
353	Comp Sci	Smith
353	Comp Sci	Clark
456	Mathematics	Jamieson
355	Mathematics	Jamieson
221	Decision Sci	$\perp$
$\perp$	Chemistry	Turner

Figure 6.6 *Relation U_2*

In Figure 6.6, we have combined the data from the relations R_1 and R_2 in the same tuple of the universal relation U_2 with the scheme (*Course*, *Department*, *Professor*). Now the number of null values have been reduced at the expense of a certain amount of duplication. For instance, course 353 appears in two tuples of U_2 as being offered by the Comp Sci department.

When the roles that the attribute Department play in the relation R_1 and R_2 are explicitly expressed, we get the universal relation U_3 with the scheme (*Course*, *Crs_Dept*, *Fac_Dept*, *Professor*). Here, *Crs_Dept* is the attribute *Department* in the relation R1 renamed to indicate the department responsible for a given course and *Fac_Dept* is the attribute *Department* in the relation R_2 renamed to indicate the department of a Professor. In Figure 6.7 we have allowed tuples from different relations to appear in a tuple of the universal relation. For symmetry, we express the cross product of the tuple of relations R_1 and R_2 in the universal relation U_3. This gives a representation which does not involve any null values, but leads to an extensive amount of duplication of data and the associated problems of maintaining data consistencies.

We can retrieve the original relations R_1 and R_2 by a projection operation as follows:

$$R_1 = \prod_{\{Course,\ Department\}}(U_1)$$

$$R_2 = \prod_{\{Professor,\ Department\}}(U_1)$$

However, we will get some tuples with null values which did not exist in the original R_1 and R_2

relations. These tuples are called *spurious tuples* and they have to be *ignored*! The above example of representing data by the universal relation shows some of the problems of this assumption.

The universal relation is obtained by including all database attributes in a single relation. There is controversy in the database community as to the validity of the universal assumption. The universal relation assumption is helpful in providing some consistency in the use of attribute names in the database. A given attribute name appearing in the database must have the same meaning to make meaningful interpretation of the natural join operation. Without such universal meaning of an attribute, we will be forced to assume that multiple occurrences of an attribute in multiple relation schemes have different meanings and hence, the interrelation connection cannot be made.

Course	*Crs_Dept*	*Fac_Dept*	*Professor*
353	Comp Sci	Comp Sci	Smith
456	Mathematics	Comp Sci	Smith
355	Mathematics	Comp Sci	Smith
221	Decision Sci	Comp Sci	Smith
353	Comp Sci	Comp Sci	Clark
456	Mathematics	Comp Sci	Clark
355	Mathematics	Comp Sci	Clark
221	Decision Sci	Comp Sci	Clark
353	Comp Sci	Chemistry	Turner
456	Mathematics	Chemistry	Turner
355	Mathematics	Chemistry	Turner
221	Decision Sci	Chemistry	Turner
353	Comp Sci	Mathematics	Jamieson
456	Mathematics	Mathematics	Jamieson
355	Mathematics	Mathematics	Jamieson
221	Decision Sci	Mathematics	Jamieson

Figure 6.7 Relation U_3

We will refer to the universal relation assumption in the synthesis approach to relational database design in Chapter 7.

6.4 Functional Dependency

As we discussed in Chapter 2, functional dependencies are the consequence of the interrelationship among attributes of an entity represented by a relation or due to the relationship among entities that is also represented by a relation. Thus, if **R** represents an entity, and if the set **X** of attributes represents the key of **R**, then for any other set of attribute **Y** of **R**, $\mathbf{X} \rightarrow \mathbf{Y}$. This is due to the fact that the key of a relation identifies a tuple and hence a particular instance of the corresponding entity. Two tuples of a relation having the same key must represent the same instance of the corresponding entity and since duplicate tuples are not allowed, these two tuples must indeed be the same tuple and the value of the attributes in **Y** determined by the key value must be identical. Similarly if **R** represents a many-to-one relationship between two entities, say from E_1 to E_2, and if **X** contains attributes which form a key of E_1 and **Y** contains attributes which contain a key of E_2, then again the FD $\mathbf{X} \rightarrow \mathbf{Y}$ will hold. On the other hand, if **R** represents a one-to-one relationship between entity E_1 and E_2, then the FD $\mathbf{Y} \rightarrow \mathbf{X}$ will hold in addition to the FD $\mathbf{X} \rightarrow \mathbf{Y}$.

Let **R** be a relation scheme where each attribute A_i is defined on some domain $\mathbf{D_i}$ for $1 \le i \le n$. Let **X**,**Y**, **Z**, etc. be subsets of $\{ A_1, A_2, \ldots, A_n\}$. We will write $\mathbf{X} \cup \mathbf{Y}$ as simply **XY**.

Let R be a relation on the relation scheme **R**. Then R satisfies the functional dependency $\mathbf{X} \rightarrow \mathbf{Y}$ if a given set of values for each attribute in **X** uniquely determines each of the values of the attributes in **Y**. **Y** is said to be functionally dependent on **X**. The functional dependency (FD) is denoted as $\mathbf{X} \rightarrow \mathbf{Y}$, where **X** is the left hand side or the **determinant** of the FD and **Y** is the right hand side of the FD. We can say that the FD $\mathbf{X} \rightarrow \mathbf{Y}$ is satisfied on the relation R if the cardinality of $\prod_{Y}(\sigma_{X=x}(R))$ is at most one. In other words if two tuples t_i and t_j of R have the same **X** value, then the corresponding value of **Y** will be identical.

A functional dependency $\mathbf{X} \rightarrow \mathbf{Y}$ is said to be **trivial** if $\mathbf{Y} \subseteq X$

Example 6.2: In the relation SCHEDULE*(Prof, Course, Room, Max_Enrollment , Day, Time*) of Figure 6.8, the FD *Course* → *Prof* is satisfied. However, the FD *Prof* → *Course* is not satisfied.

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

Figure 6.8 *The SCHEDULE Relation*

In order to verify if a given FD $\mathbf{X} \rightarrow \mathbf{Y}$ is satisfied by a relation R on a relation scheme **R**, we find any two tuples with the same **X** value; now if the FD $\mathbf{X} \rightarrow \mathbf{Y}$ is satisfied in R then the **Y** values in these tuples must be the same. We repeat this procedure until we have examined all such pairs of tuples with the same **X** value. A simpler approach involves ordering the tuples of R on the **X** values so that all tuples with the same **X** values are together. Then it is easy to verify if the corresponding **Y** values are also the same and hence verify if R satisfies the FD $\mathbf{X} \rightarrow \mathbf{Y}$.

The FD $\mathbf{X} \rightarrow \mathbf{Y}$ on a relation scheme must hold for all possible relations defined on the relation scheme **R**. Thus, we cannot look at a table representing a relation on the scheme **R** at a point in time and say, simply by inspection, that some FD $\mathbf{X} \rightarrow \mathbf{Y}$ holds. For example, if the relation SCHEDULE at some point in time contained the tuples as shown in Figure 6.8, we might erroneously conclude that the FD {*Prof* → *Course*} holds. The examination of the real world situation corresponding to the relation scheme **SCHEDULE** tells us that a particular Professor may be teaching more than one course.

Example 6.3: In the relation scheme **STDINF** (*Name*, *Course*, *Phone_No*, *Major*, *Prof*, *Grade*), the following functional dependencies are satisfied {*Name* → *Phone_No*; *Name* → *Major*; *Name*, *Course* → *Grade*; *Course* → *Prof*}.

6.4.1 Dependencies and Logical Implications

Given a relation scheme **R** and a set of functional dependencies **F**, let us consider a functional dependency $\mathbf{X} \rightarrow \mathbf{Y}$, which is not in **F**. **F** can be said to logically imply $\mathbf{X} \rightarrow \mathbf{Y}$ if for every relation R on the relation scheme **R** that satisfies the functional dependencies in **F**, R also satisfies $\mathbf{X} \rightarrow \mathbf{Y}$.

F logically implies $\mathbf{X} \rightarrow \mathbf{Y}$ is written as $\mathbf{F} \models \mathbf{X} \rightarrow \mathbf{Y}$.

Example 6.4: **R** = (*A, B, C, D*) and **F** = $\{A \rightarrow B, A \rightarrow C, BC \rightarrow D\}$, then $\mathbf{F} \models A \rightarrow D$.

Inference Axioms

Suppose we have **F**, a set of functional dependencies: in order to determine if a functional dependency $\mathbf{X} \rightarrow \mathbf{Y}$ is logically implied by **F** (i.e., $\mathbf{F} \models \mathbf{X} \rightarrow \mathbf{Y}$) we use a set of rules or axioms. Note the symbol $\models$, used here is read as "logically implies". The axioms are numbered **F1** through **F6** to indicate that they pertain to functional dependencies (as opposed to multivalued dependencies which we will examine in Chapter 7).

In the following discussions, we assume that we have a relation scheme $\mathbf{R}(A_1, A_2, A_3, ..., A_n)$; R is a relation on the relation scheme **R** and **W**, **X**, **Y**, **Z** are subsets of **R**. .

- **F1: Reflexivity:** $\mathbf{X} \rightarrow \mathbf{X}$
- **F2: Augmentation**: $\mathbf{X} \rightarrow \mathbf{Y} \models (\mathbf{XZ} \rightarrow \mathbf{Y}$, and $\mathbf{XZ} \rightarrow \mathbf{YZ})$
- **F3: Transitivity**: $(\mathbf{X} \rightarrow \mathbf{Y}$ and $\mathbf{Y} \rightarrow \mathbf{Z}) \models (\mathbf{X} \rightarrow \mathbf{Z})$
- **F4: Additivity**: $(\mathbf{X} \rightarrow \mathbf{Y}$ and $\mathbf{X} \rightarrow \mathbf{Z}) \models (\mathbf{X} \rightarrow \mathbf{YZ})$
- **F5: Projectivity**: $(\mathbf{X} \rightarrow \mathbf{YZ}) \models (\mathbf{X} \rightarrow \mathbf{Y}$ and $\mathbf{X} \rightarrow \mathbf{Z})$
- **F6: Pseudo-transitivity**: $(\mathbf{X} \rightarrow \mathbf{Y}$ and $\mathbf{YZ} \rightarrow \mathbf{W}) \models (\mathbf{XZ} \rightarrow \mathbf{W})$

Example 6.5: We use the relation R of Figure B to illustrate the above inference axioms.

Reflexivity: This is obvious since any set of attributes implies the same set of attributes. The consequence of this axiom, along with F5, is that for any $\mathbf{Y} \subseteq \mathbf{X}$, $\mathbf{X} \rightarrow \mathbf{Y}$. A FD $\mathbf{X} \rightarrow \mathbf{Y}$ is said to be a **trivial functional dependency** if $\mathbf{Y} \subseteq \mathbf{X}$.

Augmentation: This axiom indicates that the left hand side alone or both sides of a FD can be augmented.

If the relation R satisfies the FD $\mathbf{X} \rightarrow \mathbf{Y}$ then for a given **X** value that appears in R, the number of tuples having some **Y** value will be exactly one. In other words, the cardinality of $\prod_{\mathbf{Y}}(\boldsymbol{\sigma}_{\mathbf{X}=x}(R))$, written as $\left| \prod_{\mathbf{Y}} (\boldsymbol{\sigma}_{\mathbf{X}=x}(R)) \right|$ is equal to 1.

If $\mathbf{Z} \subseteq \mathbf{R}$, then $\boldsymbol{\sigma}_{\mathbf{XZ}=xz}(R) \subseteq \boldsymbol{\sigma}_{\mathbf{X}=x}(R)$, i.e., the set of tuples selected with a given value of **XZ** is a subset of the set of tuples selected for a given value of **X** alone**.** Now the number of tuples having a given **Y** value in $\boldsymbol{\sigma}_{\mathbf{XZ}=xz}(R)$ will be a subset of the tuples having the same **Y** value in $\boldsymbol{\sigma}_{\mathbf{X}=x}(R)$; since the latter is at most one, the number of tuples having a given **Y** value in **XZ** will be at most 1. Hence $\mathbf{XZ} \rightarrow \mathbf{Y}$.

It follows that $\mathbf{XZ} \rightarrow \mathbf{Y} \models \mathbf{XZ} \rightarrow \mathbf{YZ}$ and $\mathbf{X} \rightarrow \mathbf{Y} \models \mathbf{XZ} \rightarrow \mathbf{YV}$ for $\mathbf{V} \subseteq \mathbf{Z}$.

In Figure B, the FD $B \rightarrow C$ is satisfied and by augmentation we find that the FD's $AB \rightarrow C$, $BC \rightarrow C$, $BD \rightarrow C$, $BE \rightarrow C$ and $ABC \rightarrow C$, $BCD \rightarrow C$ etc. are also satisfied.

Additivity: The axiom indicates that if there are two FD's with the same left hand side, then the right

hand side of these FD's can be added to give a FD where the left hand side is the original one and the right hand side is the union of the right hand sides of the two FD's. Thus, if $\mathbf{X} \rightarrow \mathbf{Y}$, then $\prod_{Y}(\sigma_{X=x}(R))$ has at most one tuple and similarly, if $\mathbf{X} \rightarrow \mathbf{Z}$, then $\prod_{Z}(\sigma_{X=x}(R))$ has at most one tuple. Hence, $\prod_{ZY}(\sigma_{X=x}(R))$ cannot have more than one tuple. The additivity axiom follows from these observations.

We note from Figure B, that the FD's $B \rightarrow C$ and $B \rightarrow D$, and, consequently, the FD $B \rightarrow CD$, are all satisfied.

R

A	*B*	*C*	*D*	*E*
a_1	b_1	c_2	d_1	e_1
a_2	b_2	c_1	d_2	e_2
a_3	b_1	c_2	d_1	e_3
a_3	b_3	c_3	d_3	e_4
a_1	b_2	c_1	d_2	e_5
a_4	b_4	c_4	d_4	e_6
a_3	b_2	c_1	d_2	e_7
a_5	b_4	c_4	d_4	e_8

Figure B *Relation R on the scheme* **R***(A, B, C, D, E)*

Projectivity: This axiom is the inverse of the additivity axiom; it splits up or projects a FD, with a right hand side which is a union of attributes, into a number of FD's. Each projected FD has the same left hand side as the original FD and each contains a subset of the original right hand side.

For the relation R of Figure B, the FD $B \rightarrow CD$ is satisfied and hence, by projectivity, $B \rightarrow C$ and $B \rightarrow D$.

Transitivity: For the relation R of Figure 6.10, the FD's $B \rightarrow C$ and $C \rightarrow D$ are satisfied and hence, by transitivity, $B \rightarrow D$. Thus, when the value for B is b_1 in R, then the value of C is c_2. Similarly when the value of C is c_2, then the value of D is d_1. Hence, when the value of B is b_1, the value of D is d_1.

Pseudotransitivity: This axiom follows from axioms **F2** and **F3**. Given $\mathbf{X} \rightarrow \mathbf{Y}$, hence by **F2**, $\mathbf{XZ} \rightarrow \mathbf{YZ}$ and since $\mathbf{YZ} \rightarrow \mathbf{W}$ is given then by **F3**, $\mathbf{XZ} \rightarrow \mathbf{W}$.

The relation R of Figure B satisfies the FD's $C \rightarrow B$ and $AB \rightarrow E$, hence by pseudotransitivity, the FD $CA \rightarrow E$ is also satisfied.

The inference rules **F1** through **F3** are variations of the Armstrong axioms, so called after the person who first proposed them[Arms74]. In the above, we gave informal argument showing that each of the inference axioms **F1** through **F6** is sound (i.e., correct). This means that whenever a FD $\mathbf{X} \rightarrow \mathbf{Y}$ can be derived from a set of FD's **F** using these axioms, then $\mathbf{F} \models \mathbf{X} \rightarrow \mathbf{Y}$. It has been shown that the converse also holds, even for the subset **F1** through **F3**. This means that whenever $\mathbf{F} \models \mathbf{X} \rightarrow \mathbf{Y}$, then $\mathbf{X} \rightarrow \mathbf{Y}$ can be

derived from **F** using these inference axioms. This means that these axioms form a complete axiom system for FD's. Thus, in particular rules **F4** through **F6** can be derived from the rules set **F1** through **F3**.

6.4.2 Closure of a set of Functional Dependencies

The set of functional dependencies that is logically implied by **F** is called the **closure** of **F** and is written as $\mathbf{F}^+$.

Definition: If **F** is a set of FD's on a relation scheme **R** then $\mathbf{F}^+$, the ***closure*** of **F**, is the smallest set of FD's such that[3] $\mathbf{F}^+ \supseteq \mathbf{F}$ and no FD can be derived from **F** by using the inference axioms, that are not contained in $\mathbf{F}^+$. If **R** is not specified, then it is assumed to contain all the attributes that appear in **F**.

$\mathbf{F}^+$ is the set of FD's that are implied by the FD's in **F** i.e., $\mathbf{F}^+ = \{\mathbf{X} \rightarrow \mathbf{Y} \mid \mathbf{F} \models \mathbf{X} \rightarrow \mathbf{Y}\}$.

A FD *f* in $\mathbf{F}^+$ is logically implied by **F** since any relation R on the relation scheme **R** that satisfies the FD's in **F**, also satisfies the FD in $\mathbf{F}^+$, and, hence, *f*.

Example 6.6: Let **R** = (*A, B, C, D*) and **F** = $\{A \rightarrow B, A \rightarrow C, BC \rightarrow D\}$. Since $A \rightarrow B$ and $A \rightarrow C$ then by **F4** $A \rightarrow BC$. Now since $BC \rightarrow D$ then by **F3** $A \rightarrow D$, i.e., $\mathbf{F} \models A \rightarrow D$ and thus $A \rightarrow D$ is in $\mathbf{F}^+$.

An example of a FD not implied by a given set of FD is illustrated below.

Example 6.7: Let $\mathbf{F} = \{\mathbf{W} \rightarrow \mathbf{X}, \mathbf{X} \rightarrow \mathbf{Y}, \mathbf{W} \rightarrow \mathbf{XY}\}$ then $\mathbf{F}^+$ includes the set $\{\mathbf{W} \rightarrow \mathbf{W}, \mathbf{X} \rightarrow \mathbf{X}, \mathbf{Y} \rightarrow \mathbf{Y}, \mathbf{W} \rightarrow \mathbf{X}, \mathbf{X} \rightarrow \mathbf{Y}, \mathbf{W} \rightarrow \mathbf{XY}, \mathbf{W} \rightarrow \mathbf{Y}\}$. The first three FD's follow from axiom **F1**, the next three FD's are in **F**, and hence in $\mathbf{F}^+$. Since $\mathbf{W} \rightarrow \mathbf{XY}$ then by axiom **F5** $\mathbf{W} \rightarrow \mathbf{X}$ and $\mathbf{W} \rightarrow \mathbf{Y}$. However, $\mathbf{F}^+$ does not contain a FD, e.g. $\mathbf{W} \rightarrow \mathbf{Z}$, since **Z** is not contained in the set of attributes that appear in **F**.

6.4.3 Testing if $\mathbf{F} \models \mathbf{X} \rightarrow \mathbf{Y}$: Algorithm to compute a Closure

To compute the closure $\mathbf{F}^+$ for a set of FD **F** is a lengthy process because the number of dependencies in $\mathbf{F}^+$, though finite, can be very large. The reason for computing $\mathbf{F}^+$ is to determine if the set of FD's $\mathbf{F} \models \mathbf{X} \rightarrow \mathbf{Y}$; this would be the case if and only if $\mathbf{X} \rightarrow \mathbf{Y} \in \mathbf{F}^+$. However, there is an alternative method to test if $\mathbf{F} \models \mathbf{X} \rightarrow \mathbf{Y}$ without generating $\mathbf{F}^+$. The method depends on generating $\mathbf{X}^+$, the closure of **X** under **F**.

Definition: The closure of **X** under a set of functional dependencies **F** and written as $\mathbf{X}^+$, is the set of attributes $\{A_1, A_2, .., A_m\}$ such that the FD $\mathbf{X} \rightarrow A_i$ for $A_i \in \mathbf{X}^+$ follows from **F** by the inference axioms for functional dependencies.

$\mathbf{X}^+$, the closure of **X** with respect to the set of functional dependencies **F**, is the set of attributes $\{A_1, A_2, A_3, \ldots, A_m\}$ such that each of the FD's $\mathbf{X} \rightarrow A_i$, $1 \le i \le m$ can be derived from **F** by the inference axioms. Also by the additivity axiom for functional dependency, $\mathbf{F} \models \mathbf{X} \rightarrow \mathbf{Y}$, if $\mathbf{Y} \subseteq \mathbf{X}^+$. (By the completeness of the axiom system, if $\mathbf{F} \models \mathbf{X} \rightarrow \mathbf{Y}$, then $\mathbf{Y} \subseteq \mathbf{X}^+$ -see lemma below.)

Having found $\mathbf{X}^+$, we can test if $\mathbf{F} \models \mathbf{X} \rightarrow \mathbf{Y}$ by checking if $\mathbf{Y} \subseteq \mathbf{X}^+$: $\mathbf{X} \rightarrow \mathbf{Y}$ is logically implied by **F**, if and only if $\mathbf{Y} \subseteq \mathbf{X}^+$.

---

3 $\mathbf{F}^+ \supseteq$ F denotes that $\mathbf{F}^+$ contains **F**.

Let us now present the algorithm to compute the closure $\mathbf{X}^+$ given a set of FD's **F** and a set of attributes **X**. The importance of computing the closure $\mathbf{X}^+$ is that it can be used to decide if any FD $\mathbf{X} \rightarrow \mathbf{Y}$ can be deduced from **F**. The following lemma establishes that if $\mathbf{Y} \subseteq \mathbf{X}^+$ then $\mathbf{F} \models \mathbf{X} \rightarrow \mathbf{Y}$.

Lemma: $\mathbf{F} \models \mathbf{X} \rightarrow \mathbf{Y}$ if and only if $\mathbf{Y} \subseteq \mathbf{X}^+$.

Proof: Suppose that $\mathbf{Y} \subseteq \mathbf{X}^+$. Then by the definition of $\mathbf{X}^+$, $\mathbf{X} \rightarrow A$ can be derived from **F** using the inference rules, for each $A \in \mathbf{Y}$. Now, by the soundness of these rules, $\mathbf{F} \models \mathbf{X} \rightarrow A$ for each $A \in \mathbf{Y}$ and by the additivity rule, $\mathbf{F} \models \mathbf{X} \rightarrow \mathbf{Y}$. Now, suppose that $\mathbf{F} \models \mathbf{X} \rightarrow \mathbf{Y}$. Then by completeness of the inference rules, $\mathbf{X} \rightarrow \mathbf{Y}$ can be derived from **F** using them. By projectivuty, $\mathbf{X} \rightarrow A$ can be derived for each $A \in \mathbf{Y}$. This clearly implies that $\mathbf{Y} \subseteq \mathbf{X}^+$ by the definition of $\mathbf{X}^+$.

The Algorithm 6.1 to compute $\mathbf{X}^+$ is given below.. It starts with the set $\mathbf{X}^+$ initialized to **X**, the left hand side of the FD $\mathbf{X} \rightarrow \mathbf{Y}$, which is to be tested for logical implication under **F**. For each FD $\mathbf{W} \rightarrow \mathbf{Z}$ in **F**, if $\mathbf{W} \subseteq \mathbf{X}^+$,then the algorithm modifies $\mathbf{X}^+$ by forming a union of $\mathbf{X}^+$ and **Z**. The algorithm terminates when there is no change in $\mathbf{X}^+$.

Title: **Algorithm 6.1: Compute closure: $\mathbf{X}^+$**

Input: A set of functional dependencies **F** and a set of attributes **X**.

Output: The closure $\mathbf{X}^+$ of **X** under the FD's in **F**.

Body:

```
X+ := X; (* initialize X+ to X *)
change := true;
while change do
        begin
        change := false;
        for each FD W → Z in F do
                begin
                if W ⊆ X+ then do
                                        begin
                                        X+ := X+ ∪ Z;
                                        change := true;
                                        end
                end
        end
 (* X+ now contains the closure of X under F *)
```

Example 6.8: Let $\mathbf{X} = BCD$ and $\mathbf{F} = \{ A \rightarrow BC, CD \rightarrow E, E \rightarrow C, D \rightarrow AEH, ABH \rightarrow BD, DH \rightarrow BC \}$. We want to compute the closure $\mathbf{X}^+$, of **X** under **F**.

We initialize $\mathbf{X}^+$ to **X** i.e., $\mathbf{X}^+ := BCD$. Now since the left hand side of the FD $CD \rightarrow E$ is a subset of current set $\mathbf{X}^+$ i.e., $CD \subseteq \mathbf{X}^+$, $\mathbf{X}^+$ is augmented by the right hand side of the FD i.e., E; thus $\mathbf{X}^+$ now becomes equal to $BCDE$. Similarly, since $D \subseteq \mathbf{X}^+$, the right hand side of the FD $D \rightarrow AEH$ is added to $\mathbf{X}^+$ which now becomes $ABCDEH$. $\mathbf{X}^+$ cannot be augmented any further and the algorithm ends with $\mathbf{X}^+$ equal to $ABCDEH$.

The time complexity of the closure algorithm can be derived as follows. Suppose the number of

attributes in **F** is **a** and the number of FD's in **F** is **f** where each FD in **F** involves only one attribute on right hand side. Then the inner *for* loop will be executed at most **f** times, one for each FD in **F** and each such execution can take the time proportional to **a** to check if one set is contained in another set. Thus the order of execution of the for loop is **O(af)**. In the worst case each execution of the *while* loop can increase the closure by one element and since there are **f** FD's, the while loop can be repeated at most **f** times. Hence the time complexity of the algorithm is $\mathbf{O(af^2)}$. The algorithm can be modified to run in time proportional to the number of symbols needed to represent the FD's in **F**. The modification takes into account the fact that the FD's whose right hand sides are already added to $\mathbf{X}^+$ need not be reconsidered in the *for* loop. Furthermore, the FD's whose left hand side length is greater than the current length of $\mathbf{X}^+$ need not be tested in the *for* loop. The reader is referred to the bibliographic notes for reference to a closure algorithm with these modifications.

6.4.4 Testing if a FD is in a closure

As mentioned earlier, in order to find out whether $\mathbf{F} \models \mathbf{X} \rightarrow \mathbf{Y}$ without computing $\mathbf{F}^+$ requires the computation of $\mathbf{X}^+$ under the set of FD's **F**, and if $\mathbf{Y} \subseteq \mathbf{X}^+$ then **F** logically implies the functional dependency $\mathbf{X} \rightarrow \mathbf{Y}$, otherwise it does not. Algorithm 6.2 gives the steps to test the membership of $\mathbf{X} \rightarrow \mathbf{Y}$ in $\mathbf{F}^+$ by this indirect scheme. It uses the Algorithm 6.1 to compute the closure of **X** under **F**.

Title: Algorithm 6.2: Membership Algorithm

Input: A set of functional dependencies **F**, and the functional dependency $\mathbf{X} \rightarrow \mathbf{Y}$.

Output: Is $\mathbf{X} \rightarrow \mathbf{Y} \in \mathbf{F}^+$ or not? *True*; or *false*

Body:

Compute $\mathbf{X}^+$ using the Algorithm 6.1 .
if $\mathbf{Y} \subseteq \mathbf{X}^+$ *then* $\mathbf{X} \rightarrow \mathbf{Y} \in \mathbf{F}^+ :=$ *true*;
else $\mathbf{X} \rightarrow \mathbf{Y} \in \mathbf{F}^+ :=$ *false*;

Example 6.9: Let $\mathbf{F} = \{A \rightarrow BC,\ CD \rightarrow E,\ E \rightarrow C,\ D \rightarrow AEH,\ ABH \rightarrow BD,\ DH \rightarrow BC\}$. We want to find if $\mathbf{F} \models BCD \rightarrow H$.

Having computed BCD^+, in Example 6.8, as being $ABCDEH$ we can clearly see that the FD $BCD \rightarrow H$ is implied by the FD **F** since $H \subseteq BCD^+$.

The time complexity of the Algorithm 6.2 is similar to the complexity of the Algorithn 6.1 being part of the former.

6.4.5 Covers

Given a set of FD's **F**, $\mathbf{F}^+$ is the closure of **F** and contains all FD's that can be derived from **F**. As mentioned earlier, $\mathbf{F}^+$ can be very large; hence, we will look for a smaller set of FD's which are representative of the closure of **F**. Suppose we have another set of FD's **G**; we say that **F** and **G** are equivalent if the closure of **F** is identically equal to the closure of **G**, i.e., $\mathbf{F}^+ = \mathbf{G}^+$. If the sets of FD's **F** and **G** are equivalent, then we can consider one to be representative of the other or one ***covers*** the other. Thus **F** covers **G** and **G** covers **F**.

Definition: Given two sets of FD's **F** and **G** over a relation scheme **R**. **F** and **G** are equivalent (i.e., $\mathbf{F} \equiv \mathbf{G}$) if the closure of **F** is identically equal to the closure of **G** (i.e., $\mathbf{F}^+ = \mathbf{G}^+$). If **F** and **G** are equivalent then **F covers G** and **G covers F**.

If **G** covers **F** and if no proper subset **G'**($\mathbf{G'} \subseteq \mathbf{G}$) covers **F**, then **G** is called a *nonredundant cover*

Definition: Given a set of FD's **F**, we say that it is **nonredundant** if no proper subset **F'** of **F** is equivalent to **F**, i.e., no **F'** exists such that $\mathbf{F'}^+ = \mathbf{F}^+$.

Given a functional dependency $\mathbf{X} \rightarrow \mathbf{Y}$, where $\mathbf{Y} = A_1A_2A_3 \ldots A_n$, then the functional dependency $\mathbf{X} \rightarrow \mathbf{Y}$ can be replaced by an equivalent set of FD's $\{\mathbf{X} \rightarrow A_1, \mathbf{X} \rightarrow A_2, \mathbf{X} \rightarrow A_3, \ldots, \mathbf{X} \rightarrow A_n\}$ by using the inference axioms **F4** and **F5** (additivity and projectivity). A nontrivial FD of the form $\mathbf{X} \rightarrow A_i$ where the right hand side has only one attribute is called a *simple* FD. Thus every set of FD's **F** can be replaced by an equivalent set of FD's G where G contains only simple FD's.

6.4.6 Nonredundant and Minimum Covers

Given **F** a set of FD's, then if a proper subset **F'** of **F** covers **F**, (i.e., $\mathbf{F'} \subset \mathbf{F}$ and $\mathbf{F'}^+ = \mathbf{F}^+$) then, **F** is redundant and we can remove some FD, say $\mathbf{X} \rightarrow \mathbf{Y}$ from **F** to find a nonredundant cover of **F**. The Algorithm 6.3 finds a nonredundant cover of **F**. It does so by removing one FD $\mathbf{X} \rightarrow \mathbf{Y}$ from **F** and then checking if this FD is implied by the FD set $\{\mathbf{F} - (\mathbf{X} \rightarrow \mathbf{Y})\}$ by using the Algorithms 6.1 and 6.2 - finding the cover $\mathbf{X}^+$ under the set of FD's $\{\mathbf{F} - (\mathbf{X} \rightarrow \mathbf{Y})\}$). If $\{\mathbf{F} - (\mathbf{X} \rightarrow \mathbf{Y})\} \models \mathbf{X} \rightarrow \mathbf{Y}$, then $\mathbf{X} \rightarrow \mathbf{Y}$ can be removed from **F**. Algorithm 6.3 repeats this procedure for each FD that remains in **F**. Note that the nonredundant cover so obtained depends on the order in which the functional dependencies are considered. Thus, starting with a set **F** of functional dependencies we can derive more than one nonredundant cover. (See Exercise 6.7).

Title: Algorithm 6.3: Nonredundant cover

Input: A set of FD's **F**

Output: A nonredundant cover of **F**

Body

```
G := F; (* initialize G to F *)
for each FD X → Y in G do
        if X → Y ∈ {F -(X → Y )}+  (*  i.e., {F-(X→Y)}⊨ X→Y *)
          then   F :=  {F - (X → Y )};  (* remove  the FD  X → Y *)
G := F; (*  G is the nonredundant cover of F  *)
end;
```

Example 6.10: If $\mathbf{F} = \{A \rightarrow BC, CD \rightarrow E, E \rightarrow C, D \rightarrow AEH, ABH \rightarrow BD, DH \rightarrow BC\}$ then the FD's $CD \rightarrow E$ and $DH \rightarrow BC$ are redundant. We find that $(CD)^+$ under $[\mathbf{F} - \{CD \rightarrow E\}]$ is equal to $ABCDEH$, and since the right hand side of the FD $[CD \rightarrow E] \in (CD)^+$ under $[\mathbf{F} - \{CD \rightarrow E\}]$, $[\mathbf{F} - \{CD \rightarrow E\}] \models [CD \rightarrow E.]$ We now remove this redundant FD from **F** and then find that for the FD $DH \rightarrow BC$, $(DH)^+$ under $[\mathbf{F} - \{DH \rightarrow BC\}]$ is $ABCDEH$. Since the right hand side of the FD $[DH \rightarrow BC] \subseteq (DH)^+$ the FD $[DH \rightarrow BC]$ is also redundant. No remaining FD's can be removed from the

modifies **F**. Thus a non-redundant cover for **F** is: $\{A \rightarrow BC,\ E \rightarrow C, D \rightarrow AEH,\ ABH \rightarrow BD\}$.

If **F** is a set of FD's and if **G** is a non-redundant cover of **F**, then it is not true that **G** has the minimum number of FD's. In fact, there may exist a cover **G'** of **F** which has fewer FD's then **G**. Thus, a minimum cover **G'** of **F** has as small a number of FD's as any other cover of **F**. It is needless to add that a minimum cover **G'** of **F** has no redundant FD's; however, a non redundant cover of **F** need not be minimal as we see in Example 6.11. We will not discuss an algorithm to derive a minimum cover in this text. The interested reader is referred to the bibliographic notes at the end of the chapter.

6.4.7 Canonical Cover

Definition: A set of functional dependencies $\mathbf{F_c}$ is a ***canonical cover*** if every FD in $\mathbf{F_c}$ satisfies the following:

- *each* FD *in* $\mathbf{F_c}$ *is simple*, (recall that in a simple FD the right hand side has a single attribute i.e., each FD is of the form $\mathbf{X} \rightarrow A$);

- for no FD $\mathbf{X} \rightarrow A$ with $\mathbf{Z} \subset \mathbf{X}$ is $\{(\mathbf{F_c} - (\mathbf{X} \rightarrow A)) \cup (\mathbf{Z} \rightarrow A)\} \models \mathbf{F_c}$. In other words the left hand side of each FD does not have any extraneous attributes i.e., the *FD's in* $\mathbf{F_c}$ *are left reduced*;

- *no* FD $\mathbf{X} \rightarrow$ A *is redundant* i.e., $\{\ \mathbf{F_c} - (\mathbf{X} \rightarrow A)\}$ does not logically imply $\mathbf{F_c}$.

A canonical cover is sometimes called ***minimal***.

Given a set **F** of functional dependencies we can find a canonical set $\mathbf{F_c}$; Obviously $\mathbf{F_c}$ covers **F**.

Example 6.11: If $\mathbf{F} = \{A \rightarrow BC,\ CD \rightarrow E,\ E \rightarrow C,\ D \rightarrow AEH,\ ABH \rightarrow BD,\ DH \rightarrow BC\}$ then a non-redundant cover for **F** is $\{\ A \rightarrow BC,\ E \rightarrow C,\ D \rightarrow AEH,\ ABH \rightarrow BD\ \}$. The FD $ABH \rightarrow BD$ can be decomposed into the FD's $ABH \rightarrow B$ and $ABH \rightarrow D$. Now since the FD $A \rightarrow B$ is in **F**, we can left reduce these decomposed FD into $AH \rightarrow B$ and $AH \rightarrow D$. We also notice that $AH \rightarrow B$ is redundant since the FD $A \rightarrow B$ is already in **F**. This gives us the canonical cover as being $\{A \rightarrow B,\ A \rightarrow C,\ E \rightarrow$ Ü $C,\ D \rightarrow A,\ D \rightarrow E,\ D \rightarrow H,\ AH \rightarrow D\}$

Note: If $\mathbf{F_c}$ is a canonical cover, and if we form **G** using the additivity axiom (such that the FD's with the same left hand sides are merged into a single FD with the right hand sides combined), then $\mathbf{F_c}$ and **G** are equivalent. However, G will contain non-simple FD's.

6.4.8 Functional Dependencies and Keys

We have discussed earlier the concept of uniquely identifying an entity within an entity set by the concept of key; the key being a set of attributes of the entity. A relation scheme **R** has a similar concept which can be explained using functional dependencies.

Definition: **Key** Given a relation scheme **R** $\{A_1A_2A_3 \ldots A_n\}$,and a set of functional dependencies **F**, a **key** **K** of **R** is a subset of **R** such that the following are satisfied:

- $\mathbf{K} \rightarrow A_1A_2A_3 \ldots A_n$ is in $\mathbf{F}^+$

- For any $\mathbf{Y} \subset \mathbf{K}$, $\mathbf{Y} \rightarrow A_1A_2A_3 \ldots A_n$ is not in $\mathbf{F}^+$

The first requirement indicates that the dependency of all attributes of **R** on **K** is given explicitly in **F** or it can be logically implied from **F**. The second requirement indicates that no proper subset of **K** can

determine all the attributes of **R**. Thus, the key used here is ***minimal*** with respect to this property and the FD **K** → **R** is left reduced. A superset of **K** can then be called a ***superkey***.

If there are two or more subsets of **R** such that the above conditions are satisfied, then such subsets are called **candidate keys**. In such a case one of the candidate keys is designated as the **primary** key or simply as the key: the others are alternate keys.

We do not allow any attribute in the key of a relation to have a null value.

Example 6.12: If **R** (*ABCDEH*) and **F** = {$A \rightarrow BC$, $CD \rightarrow E$, $E \rightarrow C$, $D \rightarrow AEH$, $ABH \rightarrow BD$, $DH \rightarrow BC$}, then *CD* is a key of **R** since $CD \rightarrow ABCDEH$ is in $\mathbf{F}^+$ (since **(CD)+** under **F** is equal to *ABCDEH* and $ABCDEH \subseteq ABCDEH$). Other candidate keys of **R** are *AD* and *ED*.

Full Functional Dependency

The concept of left-reduces FDs and fully functional dependency is defined below and illustrated tin Example 6.13.

Definition: **Full Functional Dependency** Given a relational scheme **R** and a FD **X** → **Y**, then **Y** is ***fully functionally dependent*** on **X** if there is no **Z**, where **Z** is a proper subset of **X** such that **Z** → **Y**. Thus, the dependency **X** → **Y** is left reduced, there being no extraneous attributes in the left hand side of the dependency.

Example 6.13: In the relation scheme **R** (*ABCDEH*) with the FD's, **F** = {$A \rightarrow BC$, $CD \rightarrow E$, $E \rightarrow C$, $CD \rightarrow AH$, $ABH \rightarrow BD$, $DH \rightarrow BC$}, the dependency $A \rightarrow BC$ is left reduced and *BC* is fully functionally dependent on *A*. However, the functional dependency $ABH \rightarrow D$, is not left reduced, the attribute *B* being extraneous in this dependency.

Prime Attribute and Nonprime Attribute

Definition: **Prime, nonprime attribute** An attribute *A* in a relation scheme **R** is a **prime attribute** or simply **prime**, if *A* is part of any candidate key of the relation. If *A* is not a part of any candidate key of **R**, *A* is called a **nonprime attribute** or simply **nonprime**.

We defined the key of a relation scheme earlier,We distinguish the attributes that participate in any such key as indicated in the above definition.

Example 6.14: If **R** (*ABCDEH*) and **F** = {$A \rightarrow BC$, $CD \rightarrow E$, $E \rightarrow C$, $AH \rightarrow D$}; then *AH* is the only candidate key of R. The attributes *A* and *H* are prime, and the attributes *B*, *C*, *D*, and *E* are nonprime.

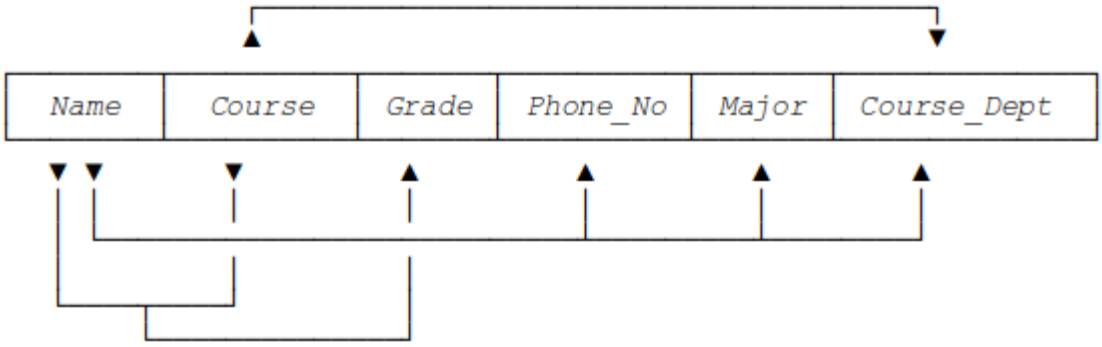
Partial Dependency

Let us introduce the concept of partial dependency below. We illustrate partial dependencies in Example 6.15.

Definition: **Partial Dependency** Given a relation scheme **R** with the functional dependencies **F** defined on the attributes of **R**. Let **K** be a candidate key. If **X** is a proper subset of **K**, and if **F** $\models$ **X** → *A*, then, *A* is said to be ***partially dependent*** on **K**.

Example 6.15:

(a) In the relation scheme STUDENT_COURSE_INFO(*Name*, *Course*, *Grade*, *Phone_No*, *Major*, *Course_Dept*) with the FD's, F = {*Name* → *Phone_NoMajor*, *Course* → *Course_Dept, NameCourse* → *Grade*}. Then *NameCourse* is a candidate key, *Name* and *Course* are prime attributes. *Grade* is fully functionally dependent on the candidate key. *Phone_No*, *Course_Dept*, and *Major* are partially dependent on the candidate key.



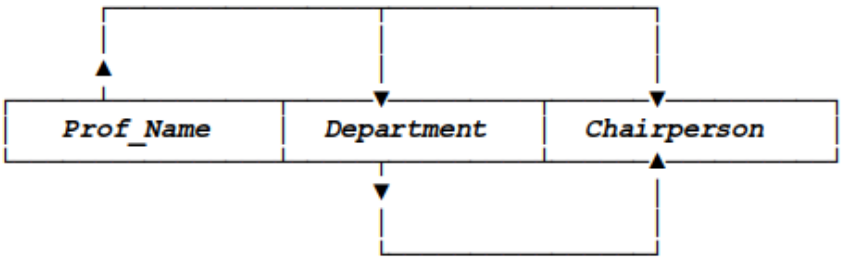
(b) Given **R** (*A, B, C, D*) and the **F** = {*AB* → *C*, *B* → *D*}. The key of this relation is *AB* and *D* is partially dependent on the key.

Transitive Dependency

Another type of dependency which we have to recognize in database design is introduced below and illustrated in Example 6.16.

Definition: **Transitive Dependency** - Given a relation scheme **R** with the functional dependencies **F** defined on the attributes of **R**. Let **X** and **Y** be subsets of **R** and let *A* be an attribute of **R** such that **X** $\not\subset$**Y**, *A* $\not\subset$ **XY**. If the set of functional dependencies {**X** → **Y**, **Y** → *A*} is implied by **F** (i.e., **F** ⊨ **X** → **Y** → *A* and **F** ¬⊨**Y** → **X**), then *A* is ***transitively dependent*** on **X**.

Example 6.16:



(a) In the relation scheme **PROF_INFO**(*Prof_Name*, *Department*, *Chairperson*) and the function dependencies **F** = {*Prof_Name* → *Department*, *Department* → *Chairperson*}, *Prof_Name* is the key and *Chairperson* is transitively dependent on the key since *Prof_Name* → *Department* →

Chairperson.
(b) Given **R** *(A, B, C, D, E)* and the function dependencies **F** = $\{AB \rightarrow C, B \rightarrow D, C \rightarrow E\}$, then AB is the key and E is transitively dependent on the key since $AB \rightarrow C \rightarrow E$.

6.5 Relational Database Design

Relational database design, like database design using any other data model, is far from a completely automated process[4] under the current state of the database technology. It is an activity that requires the close attention of the database designer, who may be one individual, for example the database administrator (DBA), or it may involve a team working with the DBA. This activity consists of identifying that portion of the enterprise for which the database application is being designed. The entity sets, their attributes, the domains on which the attributes are defined and the constraints that these attributes have to satisfy need to be identified. It is only then that the design of the relational schemes can begin.

Two approaches are generally used in designing a relational database: these are the **decomposition** approach and the **synthesis** approach. The decomposition approach starts with one (the universal) relation and the associated set of constraints in the form of functional dependencies, multivalued dependencies and join dependencies. A relation that has any undesirable properties in the form of insertion, deletion, or update anomalies is replaced by its projections. A number of desirable forms of projections have been identified which we will examine in the following sections. A number of algorithms for decomposing the input relation have been developed and reported in the database literature. We will examine some of these. Each of these algorithms produces relations that are desirable from the point of view of some of the criteria described below. We will discuss the synthesis approach, multivalued dependencies and join dependencies in chapter 7. The synthesis approach starts with a set of functional dependencies on a set of attributes. It then synthesizes relations of the third normal form.

Regardless of the approach used, the criteria for the design are the following.

- The design is **content preserving**: if the original relation R can be derived from the relations which result from the design process. Since the join operation is used in deriving the original relation from its decomposed relations, this criterion is also called a lossless join decomposition. The design is minimally content preserving if here are no redundant relations which are required in recovering the original relation R.

- The relation design is **dependency preserving** if the original set of constraints can be derived from the dependencies in the output of the design process. The design is minimally dependency preserving if there are no extraneous dependencies in the output of the design process and the original dependencies cannot be derived from a subset of the dependencies in the output of the design process.

- The relation design is free from "**interrelation join constraints**" if there are no dependencies that can only be derived from the join of two or more relations in the output of the design process. This criterion is significant. If the design produces a database scheme in which some dependencies are only enforceable in a relation that is derived from the join of two or more

4 However, design aid tools do exist: most current database management systems have some form of workbench for helping in the design.

relations, then in order to enforce these dependencies, joins will have to be produced. Consider for instance a FD $\mathbf{X} \rightarrow \mathbf{Y}$. Suppose the decomposition doesn't contain any relation $\mathbf{R}_i$ such that $\mathbf{XY} \in \mathbf{R}_i$, but contain $\mathbf{R}_j$ and $\mathbf{R}_k$ such that $\mathbf{X} \in \mathbf{R}_j$ and $\mathbf{Y} \in \mathbf{R}_k$. Then the FD $\mathbf{X} \rightarrow \mathbf{Y}$ can only be enforced by joining $\mathbf{R}_j$ and $\mathbf{R}_k$. Since the join operation is a computationally expensive process, it is desirable that the database design be free of such inter-relational join constraints.

6.5.1 Re-characterizing Relational Database Schemes

Let us extend the relation scheme to not only include the set of attributes but also the set of functional dependencies among these attributes. We therefore indicate a relation scheme as: $\mathbf{R}_i$<$\mathbf{S}_i$,$\mathbf{F}_i$>. Here $\mathbf{S}_i$ is a set of attributes $\{A_{i1}, A_{i2}, ..., A_{im}\}$ and $\mathbf{F}_i$ is a set of constraints on these attributes. Given **S**, a set of attributes each of which is defined over some designated domain, a relational database scheme is a collection of relation schemes $\boldsymbol{R} = \{\mathbf{R}_1, \mathbf{R}_2, ..., \mathbf{R}_p\}$ where each $\mathbf{R}_j = <\mathbf{S}_j = \{A_{j1}, A_{j2}, ..., A_{jm}\}, \mathbf{F}_j >$.

A relational database **D** on a relational database scheme $\boldsymbol{R}$ is a collection of relations $\{R_1, R_2, ..., R_p\}$ such that the relation R_i is defined on the relation scheme $\mathbf{R}_i$<$\mathbf{S}_i$, $\mathbf{F}_i$>.

As indicated, a relation scheme **R**<**S**, **F**> consists of two components: a set **S** of attributes and a set of constraints **F**. However, we will continue to use **R** to also mean **S**, the set of attributes. Thus, to define a subset of attributes, we may use $\mathbf{X} \subseteq \mathbf{R}$ to denote $\mathbf{X} \subseteq \mathbf{S}$. Also, unless there is confusion, we will simply use the term relation to denote a relation scheme as well as a relation on a relation scheme.

6.5.2 Normal Forms - Anomalies and Data Redundancies

A number of normal forms have been defined for classifying relations. Each normal form has associated with it a number of constraints on the kind of functional dependencies that could be associated with the relation. The normal forms are used to ensure that various types of anomalies and inconsistencies are not introduced into the database. We will describe below these normal forms which are related either to the form of the relations or based on the type of functional dependencies that are allowed to exist between the attributes of the relations or among different relations.

Unnormalized Relation:

Fac_Dept	*Prof*	*Course Preferences*	
		Course	*Course_Dept*
Comp Sci	Smith	353	Comp Sci
		379	Comp Sci
		221	Decision Sci
	Clark	353	Comp Sci
		351	Comp Sci
		379	Comp Sci
		456	Mathematics
Chemistry	Turner	353	Comp Sci
		456	Mathematics
		272	Chemistry

Mathematics	**Jamieson**	**353**	**Comp Sci**
		379	**Comp Sci**
		221	**Decision Sci**
		456	**Mathematics**
		469	**Mathematics**

Figure 6.9 *Course Preferences*

Consider the table of Figure 6.9 which shows the preferences that faculty members have for teaching courses. As before, we allow, the possibility of cross-departmental teaching. For instance, a faculty member in the Computer Science Department may have a preference for a course in the Mathematics Department, and so on. The table of Figure 6.9 is said to be **unnormalized**. Each row may contain multiple set of values for some of the columns; these multiple values in a single row are also called **nonatomic** values. In Figure 6.9 the row corresponding to the preferences of faculty in the Computer Science Department has two professors. Furthermore, Prof. Smith of the Computer Science Department prefers to teach three different courses, and Prof. Clark prefers four.

Definition: **Non-Normal Form:** An *unnormalized relation contains* ***nonatomic*** *values.*

First Normal Form

The data of Figure 6.9, which has non-atomic values, can be normalized into a relation, say CRS_PREF (*Prof*, *Course*, *Fac_Dept*,*Crs_Dept*), as shown in Figure 6.10. Note that we have shown the attributes in Figure 6.10 in a different order than that given in Figure 6.9; however, as mentioned earlier, as long as the columns are labelled there is no significance in the order of the columns of a relation. Now, suppose the set of FD's that have to be satisfied is given by {*Prof* → *Fac_Dept*, *Course* → *Crs_Dept*}; then the only key of the relation CRS_PREF is (*Prof*, *Course*).

Definition: **First Normal Form (1NF)** -A relation scheme is said to be in the **first normal form** (**1NF**) if the values in the domain of each attribute of the relation are atomic. In other words, only one value is associated with each attribute ,in each tuple, and the value is not a set of values or a list of values. A database scheme is in the first normal form if every relation scheme included in the database scheme is in the 1NF.

The first normal form pertains to the tabular format of the relation as shown in Figure 6.10.

Prof	*Course*	*Fac_Dept*	*Crs_Dept*
Smith	353	Comp Sci	Comp Sci
Smith	379	Comp Sci	Comp Sci
Smith	221	Comp Sci	DecisionSci
Clark	353	Comp Sci	Comp Sci
Clark	351	Comp Sci	Comp Sci
Clark	379	Comp Sci	Comp Sci
Clark	456	Comp Sci	Mathematics
Turner	353	Chemistry	Comp Sci
Turner	456	Chemistry	Mathematics
Turner	272	Chemistry	Chemistry
Jamieson	353	Mathematics	Comp Sci

Jamieson	379	Mathematics	Comp Sci
Jamieson	221	Mathematics	DecisionSci
Jamieson	456	Mathematics	Mathematics
Jamieson	469	Mathematics	Mathematics

Figure 6.10 *The relation CRS_PREF*

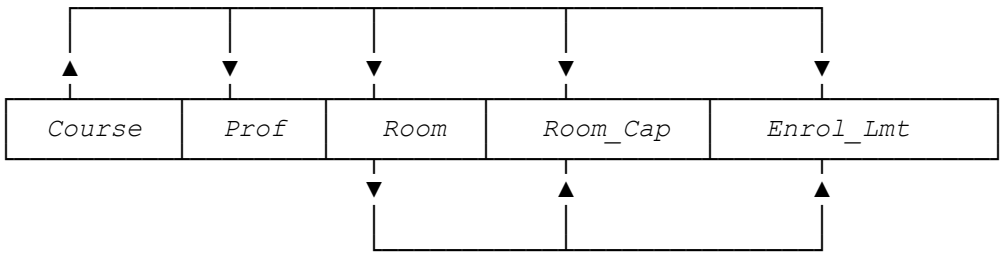
The representation of the data for the courses that a faculty member would like to teach by the relation CRS_PREF has the following drawbacks. The fact that a given professor is assigned to a given department (Fac_Dept) is repeated a number of times. In addition, the fact that a given course is offered by a given department(Crs_Dept) is also repeated a number of times. These replications could lead to some anomalies. For example, if a professor changes department, then unless all the rows of Figure 6.10 where that professor appears are changed, we could have inconsistencies in the database. In addition, if the association between a course and its department is only kept in this relation, then a new course cannot be entered (without null values) unless someone would like to teach it. Deletion of the only professor who teaches a given course on the other hand, will cause the loss of the information about the department to which the course belonged.

Second Normal Form

A second normal form does not permit partial dependency between a non-prime attribute and the relation key(s). The STDINF relation given in Section 6.2 involves partial dependency and hence is not in second normal form.

Definition: **Second Normal Form (2NF)** - A relation scheme **R**<**S**, **F**> is in the **second normal form** (**2NF**) if all non-prime attributes are fully functionally dependent on the relation key(s). A database scheme is in the second normal form if every relation scheme included in the database scheme is in the second normal form.

Even though the second normal form does not permit partial dependency between a non-prime attribute and the relation key(s), it does not rule out the possibility that a non-prime attribute may also be functionally dependent on another non-prime attribute. This latter type of dependency between non-prime attributes also causes anomalies, as we will see below.



Course	*Prof*	*Room*	*Room_Cap*	*Enrol_Lmt*
353	Smith	A532	45	40
351	Smith	C320	100	60
355	Clark	H940	400	300
456	Turner	B278	50	45
459	Jamieson	D110	50	45

Figure 6.11 *The TEACHES Relation*

Consider the TEACHES relation of Figure 6.11. It contains the attributes *Prof(essor), Course, Room, Room_Cap* (capacity of room), *Enrol_Lmt*(enrollment limit). The relation scheme for the relation TEACHES is *(Prof, Course, Room, Room_Cap, Enrol_Lmt*). The possible domain of the attribute[5] *Prof* is all the faculty members of the university. The domain of the *attribute course* is the courses offered by the university. The domain of *Room* is the rooms in the buildings of the university. The domain of *Room_Cap* is an integer value and indicates the seating capacity of the room. The domain of *Enrol_Lmt* is also an integer value and should be less than or equal to the corresponding value for *Room_Cap.*

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) COURSE_DETAILS

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

(b) ROOM_DETAILS

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) Decomposition of COURSE_DETAILS to eliminate transitive dependency

Figure 6.12 *Decomposition of TEACHES relation*

The TEACHES relation is in the 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 a given maximum number of available seats, there is a functional dependency *Room* → *Room_Cap*, and hence by transitivity, *Course* → *Room* → *Room_Cap*. Thus, the functional dependencies in this relation are {*Course* → (*Prof*, *Room*, *Room_Cap*, *Enrol_Lmt*), *Room* → *Room_Cap*}. Also, there is another transitive dependency4 *Room* → *Room_Cap* → *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. In addition since 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 6.12(a) and ROOM_DETAILS (*Room, Room_Cap*) of Figure 6.12(b). The set of functional dependencies in COURSE_DETAILS is given by {*Course* → *Prof*, *Course* → *Room*, *Course* → *Enrol_Lmt*} and the functional dependency in ROOM_DETAILS is {*Room* → *Room_Size*}. These relations do not have any partial dependencies: each of the attributes is fully

5 Here we are not concentrating on the type of attribute,.e.g. character string, but the values of these strings!

functionally dependent on the key attribute, namely *Course* and *Room*, respectively. Hence, these relations are in the 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 the third normal form does not allow partial or transitive dependencies. We can thus define a third normal form relation scheme as given below.

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

In a third normal form relation, every non-prime attribute is non-transitively and fully dependent on the key. A relation scheme **R** is *not* in the third normal form if any functional dependency such as $\mathbf{X} \rightarrow \mathbf{Y}$ implied by **F** is in conflict with the above definition of the 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 $\mathbf{X} \rightarrow A$ is a partial dependency.
- **X** is not a subset of any key of **R**: in this case there is a transitive dependency in $\mathbf{F}^+$. Since for a key **Z** of R, $\mathbf{Z} \rightarrow \mathbf{X}$ with $\mathbf{X} \notin \mathbf{Z}$, and $\mathbf{X} \rightarrow$ A, with A $\notin$ **X**, means that $\mathbf{Z} \rightarrow \mathbf{X} \rightarrow$ A is a nontrivial chain of dependencies.

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

If a relation scheme **R** contains a transitive dependency, $\mathbf{Z} \rightarrow \mathbf{X} \rightarrow A$, then we cannot insert an **X** value in the relation along with a *A* value unless we have a **Z** value to go along with the **X** value. This is the insertion anomaly. Similarly, the deletion of a $\mathbf{Z} \rightarrow \mathbf{X}$ association will also require the deletion of a 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, as in the case of 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 6.12(a), has a transitive dependency since *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 6.12(c). Note that to verify the constraint that *Enrol_Lmt* be less than the *Room_Cap* now requires a join of three relations!

Normalization Through Decomposition (based on FD's)

We noted above that if **R** contains a transitive dependency, $\mathbf{Z} \rightarrow \mathbf{X} \rightarrow A$, then we cannot insert a **X** value in the relation along with an *A* value, unless we have a **Z** value to go along with the **X** value. 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*, then the **Z** to **X** association cannot be represented without a corresponding *A* value. Similarly, the deletion of an $\mathbf{Z} \rightarrow \mathbf{X}$ association will also require the deletion of a $\mathbf{X} \rightarrow A$ association leading to the deletion anomaly. If **R** contains a partial dependency, i.e., an attribute *A* depends on a subset **X** of **K** (the key of **R**), then the association between **X** and *A* cannot be expressed unless the remaining parts of **K** are present in a tuple.

In this section we will examine how starting with a relation scheme **R** and a set of functional dependencies **F** such that **R** is not in the 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 which are 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 $\{Student\_Name \rightarrow Phone\_No, Student\_Name \rightarrow Department, Student\_Name, Course \rightarrow Grade\}$. 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.

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 of the relation. The other problem with this relation is that the changes in the *Phone_No* or *Department* of a student can lead to inconsistencies in the database.

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

Figure C *The ENROLLMENT relation*

We can rectify the problems cited in Example 6.17 for the ENROLLMENT relation by decomposing it into the following relations: STUDENT (*Student_Name*, *Phone_No*, *Department*) with the FD's {*Student_Name* → *Phone_No*, *Student_Name* → *Department*}, and ENROL(*Student_Name*, *Course*, *Grade*) with the FD's {*Student_Name*,*Course* → *Grade*}. The relations STUDENT and ENROL are shown in Figure 6.13.

Student_Name	Phone_No	Department
Jones	237-4539	Comp Sci
Ng	427-7390	Chemistry
Martin	388-5183	Physics
Dulles	371-6259	Decision Sci
Duke	823-7293	Mathematics
Evan	842-1729	Comp Sci
Baxter	839-0827	English

(a) STUDENT relation

Student_Name	Course	Grade
Jones	353	A
Ng	329	A
Jones	328	B
Martin	456	C
Dulles	293	B
Duke	491	C
Duke	353	B
Jones	491	C
Evan	353	A+
Baxter	379	B

(b) ENROLL relation

Figure 6.13 *Decomposition of ENROLLMENT*

Example 6.18: Consider the relation MAJOR(*Student_Name*, *Major*, *Department*) of Figure D with the functional dependencies {*Student_Name* → *Major*, *Student_Name* → *Department*, *Major* → *Department*}. Since the attribute *Major* is not in the key, and because of the functional dependency of *Department* on *Major*, we have a transitive dependency in this relation.

Student_Name	*Major*	*Department*
Jones	Information Systems	Comp Sci
Ng	Bio- Chemistry	Chemistry
Martin	Honours Physics	Physics
Dulles	Quantitative Methods	Decision Sci
Duke	Statistics	Mathematics
James	Systems Architecture	Comp Sci
Evan	Information Systems	Comp Sci
Baxter	Creative Writing	English

Figure D *The MAJOR relation*

The problem with the relation MAJOR is that unless a student is registered in one of the majors offered by a department, that major cannot be shown to be offered by the given department. Similarly, deleting the only student in a major loses the information of that major being offered by a given department.

This problem can be overcome by decomposing the relation MAJOR of Figrgure D into the relations: STUDENT_MAJOR(*Student_Name*, *Major*) with the functional dependency {*Student_Name* → *Major*} and MAJOR_DEPT (*Major*, *Department*) with the functional dependency {*Major* → *Department*}. These relations are shown in Figure 6.14.

Student_Name	Major
Jones	Information Systems
Ng	Bio- Chemistry
Martin	Honours Physics
Dulles	Quantitative Methods
Duke	Statistics
James	Systems Architecture
Evan	Information Systems
Baxter	Creative Writing

(a) The STUDENT_MAJOR relation

Major	Department
Information Systems	Comp Sci
Bio- Chemistry	Chemistry
Honours Physics	Physics
Quantitative Methods	Decision Sci
Statistics	Mathematics
Systems Architecture	Comp Sci
Creative Writing	English

(b) The MAJOR_DEPARTMENT relation

Figure 6.14 *A decomposition of MAJOR*

The relations of Figures 6.13 and 6.14 do not exhibit the anomaly and inconsistency problems that were present in the relations of Figures **C** and **D** respectively. Elimination of some of these anomalies is the motivation behind the decomposition of a scheme **R**<**S**,**F**> (which suffers from anomalies and inconsistency problems) into relation schemes $\mathbf{R_1}$, $\mathbf{R_2}$ etc., each of which is not necessarily a disjoint

subset of **R** so that the resulting relation schemes contain the same data as the original scheme.

6.5.3 Lossless Join and Dependencies Preserving Decomposition

A relation scheme **R** can be decomposed into a collection of relation schemes to eliminate some of the anomalies contained in the original relation scheme **R**. However, any such decomposition requires that the information contained in the original relation be maintained. This in turn requires that the decomposition must be such that a join of the decomposed relations gives the same set of tuples as the original relation and that the dependencies of the original relation must be preserved. Let us illustrate, with an example, a decomposition which violates these requirements.

The terms **lossless decomposition** and **dependency preserving decomposition** 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 ***lossless join decomposition*** or simply ***lossless*** if for every relation R(**R**) that satisfies the FD's in **F**, the natural join of the projections of R gives the original relation R: i.e.,

$R = \Pi_{R1}(R) \bowtie \Pi_{R2}(R) \bowtie \quad \bowtie \Pi_{Rn}(R)$

If $R \subset \Pi_{R1}(R) \bowtie \Pi_{R2}(R) \bowtie ... \bowtie \Pi_{Rn}(R)$ then the decomposition is called ***lossy***[6].

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 the functional dependencies of the original relation must be implied by the dependencies in the decompositions.

Example 6.19: Consider the relation STUDENT_ADVISOR(*Name*, *Department*, *Advisor*) of Figure E(a) with the functional dependencies F{*Name* → *Department*, *Name* → *Advisor*, *Advisor* → *Department*}. The decomposition of STUDENT_ADVISOR into STUDENT_DEPARTMENT (*Name*, *Department*), and DEPARTMENT_ADVISOR (*Department*, *Advisor*) given in Figures E(b) and E(c). The join of these decomposed relations is given in Figure E(d) and contains tuples that did not exist in the original relation of Figure E(a). This decomposition is called lossy**.**

6 $R \subseteq \Pi_{R1}(R) \bowtie \Pi_{R2}(R) \bowtie ... \bowtie \Pi_{Rn}(R)$ is always true**.**

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

(a) The STUDENT_ADVISOR relation

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

(b) STUDENT_DEPARTMENT

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

(c) DEPARTMENT_ADVISOR

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	

(d) Join of STUDENT_DEPARTMENT and DEPARTMENT_ADVISOR showing tuples that did not exist in the original relation.

Figure E *Example of a lossy decomposition.*

Definition: Given a relation scheme **R**<**S**,**F**> where **F** is the associated set of functional dependencies on the attributes in **S**. Consider that **R** is decomposed into the relation schemes $\mathbf{R_1}$, $\mathbf{R_2}$, ... , $\mathbf{R_n}$ with the functional dependencies $\mathbf{F_1}$, $\mathbf{F_2}$,.. , $\mathbf{F_n}$. Then this decomposition of **R** is ***dependencies preserving decomposition***, if the closure of **F'** (where $\mathbf{F'} = \mathbf{F_1} \cup \mathbf{F_2} \cup ... \cup \mathbf{F_n}$) is identical to $\mathbf{F}^+$ (i.e., $\mathbf{F'}^+ = \mathbf{F}^+$).

If we decompose a relation into relation schemes that do not preserve dependencies, then the enforcement of the original FD's can only be done 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 FD's.

We summarize these observations in the following theorem: we will not give a formal proof of this theorem but will 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 say $\mathbf{R_1}$<(**X**, **Y**), $\mathbf{F_1}$> and $\mathbf{R_2}$<(**X**,**Z**), $\mathbf{F_2}$> is:

(i) dependency preserving if every functional dependency in **R** can 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

(ii) 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 $\mathbf{X} \rightarrow \mathbf{Z}$.

Example 6.19 illustrated a decomposition which is both lossy and which doesn't preserve the dependencies in the original relation. It is lossy since the common attribute *Department* is not a key of either of the resulting relations and consequently, the join of these projected relations produces tuples which are not in the original relation. In addition the decomposition is not dependency preserving since the FD *Name* → *Advisor* is not implied by the FD's of the decomposed relation.

Example 6.20 illustrates a lossless decomposition.

Example 6.20: Let **R**(*A,B,C*) and **F** = {$A \rightarrow B$}. Then the decomposition of **R** into $\mathbf{R_1}$(*A,B*) and $\mathbf{R_2}$(*A,C*) is lossless since 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 in Example 6.21.

Example 6.21: Let **R**(*A,B,C*) and **F** = {$A \rightarrow B$}. Then the decomposition of **R** into $\mathbf{R_1}$(*A,B*) and $\mathbf{R_2}$(*B,C*) is not lossless since the common attribute *B* does not functionally determine either *A* or *C*.

Example 6.22 illustrates a decomposition which is both lossless ans dependence preserving.

Example 6.22: **R**(*A,B,C,D*) with the functional dependencies **F** = {$A \rightarrow B, A \rightarrow C, C \rightarrow D$ }. Consider the decomposition of **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 FD's can be logically derived from $\mathbf{F_1}$ and $\mathbf{F_2}$, and hence the decomposition is dependence preserving. Also, the common attribute *C* forms a key of $\mathbf{R_2}$. Hence, the decomposition of **R** into $\mathbf{R_1}$ and $\mathbf{R_2}$ is lossless.

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

Example 6.23: **R**(*A,B,C,D*) with the functional dependencies F = {$A \rightarrow B, A \rightarrow C, A \rightarrow D$}. Then the decomposition of **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 since 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 FD's in $\mathbf{R_1}$ or $\mathbf{R_2}$. Hence, the decomposition is not dependence preserving.

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

6.5.4 Algorithms to check if a Decomposition is Lossless and Dependency Preserving

Given a relation scheme **R** and a set of functional dependencies **F:** suppose **R** is decomposed into the relations $\mathbf{R_1}$, $\mathbf{R_2}$, ... , $\mathbf{R_n}$ with the functional dependencies $\mathbf{F_1}$, $\mathbf{F_2}$,.. , $\mathbf{F_n}$ respectively. We want to ascertain

(i) if the decomposition is lossless and (ii) if it is dependency preserving. The following algorithms could be used to check for these requirements: Algorithm 6.4 can be used to determine if a decomposition is lossless and Algorithm 6.5 can be used to determine if the decomposition is dependency preserving. Note that if the decomposition is into only two relations, then it would be easier to test for lossless decomposition using Theorem 6.1. However, if the decomposition is into a number of relations, Algorithm 6.4 could be used. It must be noted that a decomposition could have one of these properties without having the other.

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 E(a) illustrates some instances of tuples of a relations on this relation scheme. This relation can be decomposed by projection into the relation schemes $\mathbf{SM_mA}(S, M_m, A)$ and $\mathbf{F_sA}(F_s, A)$. The decomposition of the relation of Figure 6.F(a) into these two relations is shown in Figure 6.F(b) and (c). This decomposition is lossless since the common attribute A determines F_s.

However, the decomposition does not preserve the dependencies; the only non-trivial 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** dependence preserving. We note that the dependency (S, M_m, F_s) $\rightarrow$ A can be recovered from the join of the projected relations.

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

(a) The CONCENTRATION relation

Student	Mm	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

(b)The SM_mA Relation

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

(c) The F_sA Relation

Figure F *Example of a lossless decomposition that is not dependency preserving.*

In Algorithm 6.4, we initialize the table element (i,j) with with α_{Aj} if the attribute A_j is included in the decomposed relation R_i, otherwise we place the symbol $ß_{iAj}$. The table is then used to verify if an arbitrary tuple with all αs which is in the join of the decomposed relation is also in the relation R. If this is the case, then the decomposition is lossless, otherwise it is lossy. The interested reader is referred to the bibliographic notes for a reference to the proof of this algorithm.

Title: **Algorithm 6.4: Check if a decomposition is lossless.**

Input: A relation scheme **R**(A_1, A_2, A_3, ..., A_k), decomposed into the relation schemes $\mathbf{R}_1$, $\mathbf{R}_2$, $\mathbf{R}_3$, ..., $\mathbf{R}_i$, ..., $\mathbf{R}_n$.

Output: Whether the decomposition is lossless or lossy.

Body

(*A table, TABLE_LOSSY(1:n, 1:k) is used to test for the type of decomposition. The row i is for relation scheme $\mathbf{R}_i$ of the decomposed relation and column j is for attribute A_j in the original relation.*)

for each decomposed relation $\mathbf{R}_i$ *do*
 if an attribute A_j is included in $\mathbf{R}_i$,
  then TABLE_LOSSY(i,j) := α_{Aj} (*place a symbol α_{Aj} in row i, column j of *)
  else TABLE_LOSSY(i,j) := $ß_{iAj}$ (* place a symbol $ß_{iAj}$ *)
change := *true*
while (change) *do*
 for each FD $\mathbf{X} \rightarrow \mathbf{Y}$ in F *do*
  if rows i and j exist such that the same symbol appears in each column corresponding to the attributes of **X**
   then if one of the symbol in the **Y** column is α_r
    then make the other α_r

```
        else if the symbols are ß_pm and ß_qm
                then make both of them, say, ß_pm;
                else change := false
i := 1
lossy := true
while (lossy and i ≤ n) do
 for each row i of TABLE_LOSSY
   if all symbols are αs
     then lossy := false
     else i := i +1
```

We use Algorithm 6.4 to verify that the decomposition of Example 6.25 is lossless and that of Example 6.26 is lossy,.

Example 6.25: Given **R**(*A,B,C,D*) with the functional dependencies **F** $\{A \rightarrow B, A \rightarrow C, C \rightarrow D\}$. Consider the dependence preserving decomposition of **R** into $\mathbf{R_1}(A,B,C)$ and $\mathbf{R_2}(C,D)$. Let us verify whether it is lossless as well using the Algorithm 6.4.

We initialize the TABLE_LOSSY as shown below, on the left. Then we consider the FD $C \rightarrow D$ and find the symbols in the C columns are the same and since one of the symbol in the D column is an α, consequently we make the other element (1,4) in the D column same.

	A	*B*	*C*	*D*
R_1	α_A	α_B	α_C	β_{1D}
R_2	β_{2A}	β_{2B}	α_C	α_D

A	*B*	*C*	*D*
α_A	α_B	α_C	α_D
β_{2A}	β_{2B}	α_C	α_D

For the other FD's we are unable to find two rows with identical entries for the columns of the determinant, so there are no further changes and the final version of TABLE_LOSSY, as shown above on the right. Finally we find a row in the table with αs in all columns indicating to us that the decomposition is lossless. Since the common attribute, *C*, is a key of one of the projection, we could have used Theorem 6.1 to come to the same conclusion.

Example 6.26: R(A, B, C, D, E) with the functional dependencies F $\{AB \rightarrow CD, A \rightarrow E, C \rightarrow D\}$. Then the decomposition of **R** into $\mathbf{R_1}(A,B,C)$ and $\mathbf{R_2}(B,C,D)$ and $\mathbf{R_3}(C,D,E)$ is lossy.

We initialize the TABLE_LOSSY as shown above, on the left. Now we consider the FD's $AB \rightarrow CD$, $A \rightarrow E$ in turn but since we find that there are no two rows with identical entries in the *A* columns, we are unable to make 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 above. Finally we find no rows in the table with all αs and conclude that the decomposition is lossy.

	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

As we discussed earlier, a decomposition is dependence preserving if the closure of **F'** (where **F'**= $\mathbf{F_1} \cup \mathbf{F_2} \cup \ ...\cup \ \mathbf{F_n}$) 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 F and computes the closure $\mathbf{X'}^+$ of **X** with respect to **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 F are logically implied by **F'**, then we can conclude that the decomposition is dependency preserving. Obviously, even if a single dependency in F is not covered by **F'**, then the decomposition is not dependency preserving. Algorithm 6.5 checks if a decomposition is dependency preserving.

If the union of the dependencies of the decomposed relation is the same as the original set of dependencies, then the decomposition is dependence preserving; this is illustrated in Example 6.27.

Example 6.27: Consider **R**(*A,B,C,D*) with the functional dependencies **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 dependence preserving since all the original FD's can be logically derived from $\mathbf{F_1}$ and $\mathbf{F_2}$.

Title: Algorithm 6.5: Check if a decomposition is dependency preserving

Input: A relation scheme and a set F of functional dependencies: a projection ($\mathbf{R_1}$, $\mathbf{R_2}$, ..., $\mathbf{R_n}$) of **R** with the functional dependencies ($\mathbf{F_1}$, $\mathbf{F_2}$, ..., $\mathbf{F_n}$).

Output: Whether the decomposition is dependency preserving or not.

Body:
$\mathbf{F'}^+ = \mathbf{F}^+$:= *true*; (* Assume $\mathbf{F'}^+ = \mathbf{F}^+$, used as a variable: if it remains true at the end

of the algorithm, the decomposition is dependency preserving *)
F' := **Φ**;
for i:= 1 **to** n *do*
  F' := **F'** ∪ $\mathbf{F_i}$;
for each FD **X** → **Y** ∈ **F** *and while* ($\mathbf{F'^{+}\_=\_F^{+}}$) *do*
    (* compute $\mathbf{X'^{+}}$, the closure of **X** under **F'**, using the Algorithm 6.1 *)
  if **Y** ⊄ $\mathbf{X'^{+}}$ *then* $\mathbf{F'^{+}\_=\_F^{+}}$:= *false*; (* i.e., the decomposition is not dependency preserving *);

Example 6.28 illustrates a decomposition which is not dependencies preserving.

Example 6.28: **R**(*A,B,C,D*) with the functional dependencies F {$A \rightarrow B, A \rightarrow C, A \rightarrow D$} is decomposed 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 $\mathbf{F_2}$ = { }. This decomposition is not dependence preserving since the FD $A \rightarrow C$ is not implied by any FD's in $\mathbf{R_1}$ or $\mathbf{R_2}$.

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 6.22(a) 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 dependence preserving, since 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 E into the relations SM_mA and F_sA is not dependence preserving since **F'** = $A \rightarrow F_s$ and the FD $SM_mF_s \rightarrow A$ is not implied by **F'**.

6.5.5 Decomposition into Third Normal Form

Let us start from a normalized relation scheme **R**<**S**,**F**>, where **S** is a set of attributes with atomic domains and **F** is a set of functional dependencies such that **R** is not in the 3NF. Since **R** is normalized, we know that it is in the **1NF** (Note: here we do not insist that **R** be in **2NF**). The reason that **R** is not in the **3NF** is that it has at least one FD **Y** → *A*, where *A* is non-prime attribute which violates the 3NF requirements.

If **Y** → *A* is a partial dependency (i.e., **Y** is a subset of a key of **R**), then **R** is not in the second normal form and these partial dependencies have to be removed by decomposition. In order to ensure that this decomposition is lossless and dependency preserving, we decompose **R** into two relations schemes, say $\mathbf{R_1}$<$\mathbf{S_1}$,$\mathbf{F_1}$> and $\mathbf{R_2}$<$\mathbf{S_2}$,$\mathbf{F_2}$>: here $\mathbf{S_1}$ is **S** - *A*, $\mathbf{F_1}$ is (**F** - (**Y** → *A*)), $\mathbf{S_2}$ is **Y***A*, and $\mathbf{F_2}$ is **Y** → *A*. This decomposition is lossless since **Y** is the common attribute in $\mathbf{R_1}$ and $\mathbf{R_2}$ and it forms a key of $\mathbf{R_2}$; it is dependency preserving since the union of $\mathbf{F_1}$ and $\mathbf{F_2}$ is equal to F. The decomposition process can be

hastened by removing from **R** any other non prime attribute A_1, A_2, A_3, ... such that $\mathbf{Y} \rightarrow AA_1A_2A_3$.. . Thus **R** could be decomposed into $\mathbf{R_1}$<(**S** - $AA_1A_2A_3$...), (**F** - (**Y** → $AA_1A_2A_3$...)) > and $\mathbf{R_2}$<(Y$AA_1A_2A_3$...), **Y** → $AA_1A_2A_3$... >.

Let us now consider how we can handle the situation where $\mathbf{Y} \rightarrow A$ is a transitive dependency in **R** (if this type is the only offending form of dependency in the set **F**, then **R** is not only in the 1NF but it is also in the 2NF). If **K** is a key of **R** then $\mathbf{K} \subseteq \mathbf{S}$. Now let $\mathbf{Y} \subseteq \mathbf{S}$, with $\mathbf{Y} \not\subset \mathbf{K}$, be a set of attributes so that for some nonprime attribute $A \in \mathbf{S}$ the FD $\mathbf{K} \rightarrow \mathbf{Y} \rightarrow A$ holds under **F**, and **Y** is not a key of **R**. As before, the decomposition of **R** into $\mathbf{R}_1$ and $\mathbf{R}_2$ is done by removing from **R** the attribute A and forming a new relation $\mathbf{R_1}$<(**S**-A), {**F** - (**Y** → A)}> and $\mathbf{R_2}$<**Y**A, **Y** → A>.

The decomposition process, in the case of a transitive dependency, can be hastened by removing from the set of attributes (**R** - **KY**) any other nonprime attribute e.g. A_i, such that $\mathbf{Y} \rightarrow A_i$. These other attributes will also be transitively dependent upon the key **K** of **R**. Such further attributes A_i are also placed in the relation scheme $\mathbf{R_2}$ and removed from **R**. Thus we will get the decomposition of **R** as being $\mathbf{R_1}$<(S - $AA_1A_2A_3...A_k$), {**F** - (**Y** → $AA_1A_2A_3...A_k$)}>, and $\mathbf{R_2}$<(**Y**$AA_1A_2A_3...A_k$), **Y** → $AA_1A_2A_3...A_k$ >. As before, this decomposition is lossless since **Y** is the common attribute in $\mathbf{R_1}$ and $\mathbf{R_2}$ and it forms a key of $\mathbf{R_2}$. The decomposition is dependency preserving since the union of $\mathbf{F_1}$ and $\mathbf{F_2}$ is equal to **F**.

If either $\mathbf{R_1}$ or $\mathbf{R_2}$ with the functional dependencies $\mathbf{F_1}$ and $\mathbf{F_2}$ is not in 3NF, then we can continue the decomposition process until we get a database scheme say <$\mathbf{R_i}$,$\mathbf{F_i}$>, <$\mathbf{R_j}$,$\mathbf{F_j}$>, <$\mathbf{R_k}$,$\mathbf{F_k}$>, .. <$\mathbf{R_m}$,$\mathbf{F_m}$>.

Algorithm 6.6 is the formal method to decompose a normalized relation scheme **R**<**S**,**F**> into a number of 3NF relation schemes. The decomposition is lossless and dependence preserving. The algorithm uses the canonical cover of the set of FD's **F** (see section 6.4.7). The algorithm preserves dependency by building a relation scheme for each FD in the set of the canonical cover of **F**. The lossless join decomposition is assured in the algorithm by including in the decomposition a relation scheme that contains a candidate key of **R**. The algorithm also includes a relation scheme which contains all the attributes of **R** that are not involved in any FD in the canonical cover; this caters to any possible many-to-many association between these attributes.

Algorithm for Lossless and Dependence Preserving Third Normal Form Decomposition

For this algorithm we assume that we have a canonical cover $\mathbf{F_c}$ for the set of FD's **F** for the relation scheme **R** and that **K** is a candidate key of **R**. The algorithm 6.6 produces a decomposition of **R** into a collection of relation schemes $\mathbf{R_1}$, $\mathbf{R_2}$, ..., $\mathbf{R_n}$. Each relation scheme $\mathbf{R_i}$ is in the third normal form with respect to the projection of $\mathbf{F_c}$ onto the scheme of $\mathbf{R_i}$.

Title: Algorithm 6.6: Lossless and dependencies preserving Third Normal Form Decomposition Algorithm

Input: A relation Scheme **R**, a set of Canonical (minimal) functional Dependencies $\mathbf{F_c}$, and **K** a candidate key of **R**.

Output: A collection of third normal form relation schemes ($\mathbf{R_1}$, $\mathbf{R_2}$, ... $\mathbf{R_i}$) which are dependence preserving and lossless.

Body:
i := 0

Find all the attributes in **R** which are not involved in any FD's in $\mathbf{F_c}$ either on the left or right hand side. If any such attributes {**A**} are found then

```
    begin
     i := i+1;
     form a relation Ri{A}; ( involving attributes not in any FD's)
     R := R -{ A }; (remove the attributes {A} from R)
     end;
if there is a dependency X → Y in Fc such that all the
            attributes that remain in R are included in it
 then
  begin
   i:= i+1;
   output R as Ri{ X, Y};
   end
 else
   begin
    for each FD X → A in Fc do
            begin
             i:= i+1;
             form Ri{ X,A }, Fi{ X → A }
             end;
   combine all relation schemes corresponding to FD's with the same LHS
          (i.e., <(X,A), {X → A}> and <(X,B), {X → B}>
          could be replaced by <(X,AB), {X → AB}>)
           if none of left hand side of the FD in  Fj for 1 ≤ j ≤ i satisfies K ⊆ X
                  then begin
                          i := i+1;
                          form Ri <{ K }>;( make sure that a  relation contains the
                                                  candidate key of R)
                          end;
   end;
```

In Example 6.31 we give a decomposition into a set of 3NF relations schemes which is both lossless and dependencies preserving.

Example 6.31 Let us 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 → *Capacity*,

ShipDate → *Cargo*,

CargoCapacity → *Value*

Let us first find the cannonical cover of the given set of FD's. The FD's are simple since each has a single attribute on the right hand side. There are no redundant FD'S in the set and none of the FD contains extraneous attributes on the left hand side. Hence the given set of FD's is in canonical form. A candidate key of the relation is *ShipDate*.

Now let us use the 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 such attributes. Since there is no single FD in the canonical cover that contains all remaining attributes in **SHIPPING**, 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 in 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 $\{S \rightarrow Sd$, $RTD \rightarrow P$,$TDC \rightarrow R$, $TPD \rightarrow C$, $TSD \rightarrow R\}$. The key of this relation scheme is *TSD*. The decomposition of this relation scheme into the third normal form gives the following relation schemes:

R_1(*SMA*) with the FD $S \rightarrow MA$,

R_2(*MSd*) with the FD $M \rightarrow Sd$,

R_3(*ASd*) with the FD $A \rightarrow Sd$,

R_4(*CCdP*) with the FD $C \rightarrow CdP$,

R_5(*PPd*) with the FD $P \rightarrow Pd$,

R_6(*RTDC*) with the FD $RTD \rightarrow C$,

R_7(*TPDR*) with the FD $TPD \rightarrow R$,

R_8(*TSDR*) with the FD $TSD \rightarrow R$,

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 relations scheme **R_8** includes a candidate key and consequently we don't need to create an explicit relation scheme for the key.)

R_1 through **R_9** form a lossless and dependence preserving decomposition of **STUDENT_INFO**.

Derivation of other canonical covers for the set of FD's in Example 6.32 and the corresponding relational schemes in 3NF is left as an exercise.

6.5.6 Boyce Codd Normal Form

Consider a relation scheme in the third normal form which has a number of overlapping composite candidate keys. In particular consider the relation **GRADE**(*Name*, *Student#*, *Course*, *Grade*) of Figure 6.15.

Here the functional dependencies are { *Name*,*Course* $\rightarrow$ *Grade*; *Student#*, *Course* $\rightarrow$ *Grade*; *Name* $\rightarrow$ *Student#*; *Student#* $\rightarrow$ *Name*}. Here, we assume each student name and number are unique. The relation has two candidate keys, (*Name*, *Course*) and (*Student#*, *Course*). Each of these keys is a composite key and contains a common attribute *Course*. The relation scheme satisfies the criterion of the third normal form relation, i.e., for all functional dependencies $\mathbf{X} \rightarrow A$ in GRADE, when $A \notin \mathbf{X}$, either **X** is a superkey or *A* is prime.

Name	*Student#*	*Course*	*Grade*
Jones	23714539	353	A
Ng	42717390	329	A
Jones	23714539	328	in prog
Martin	38815183	456	C
Dulles	37116259	293	B
Duke	82317293	491	C
Duke	82317293	353	in prog
Jones	23714539	491	C
Evan	11011978	353	A+

Baxter	83910827	379	in prog

Figure 6.15 *The GRADE Relation*

However, this relation has a disadvantage in the form of repetition of data. The association between a name and the corresponding student number is repeated; an**d** any change in one of these (for example the change in the name to a compound name by marriage) has to be reflected in all tuples, otherwise there will be the problem of inconsistency in the database. Furthermore, the student number cannot be associated with a student name unless the student has registered in a course, and this association is lost if the student drops all the courses he or she is registered in.

The problem in the relation GRADE is that it had two overlapping candidate keys. In the **Boyce Codd normal form** (BCNF), which is stronger than the third normal form, the intent is to avoid the above anomalies. This is done by ensuring that, for all nontrivial FD's implied by the relation, the determinants of the FD's involve a candidate key.

Definition: A normalized relation scheme **R**<**S**,**F**> is in the **Boyce Codd normal form** if for every nontrivial FD in **F+** of the form $\mathbf{X} \rightarrow A$ where $\mathbf{X} \subseteq \mathbf{S}$ and $A \in \mathbf{S}$, **X** is a superkey of **R**.

A database scheme is in the BCNF if every relation scheme in the database scheme is in the BCNF.

A database is in BCNF if every relation scheme in the database scheme is in BCNF. In other words, for a relation scheme **R**<**S**,**F**> to be in the BCNF, for every FD in **F+** of the form $\mathbf{X} \rightarrow A$ where $\mathbf{X} \subseteq \mathbf{S}$ and $A \in \mathbf{S}$, at least one of the following conditions hold:

- $\mathbf{X} \rightarrow A$ is a trivial FD and hence $A \in \mathbf{X}$, or
- $\mathbf{X} \rightarrow \mathbf{R}$ i.e., **X** is a superkey of **R**.

The above definition of the BCNF relation indicates that a relation which is in the BCNF is also in the 3NF. The BCNF imposes a stronger constraint on the types of FD's that are allowed in a relation. The only non-trivial FD's that are allowed in the BCNF are those FD's whose determinants are candidate superkeys of the relation. In other words, even if *A* is a prime attribute, **X** must be a superkey to attain BCNF. In 3NF, **X** does not have to be a superkey, but in this case *A* must be a prime attribute. Effectively, 3NF allows non-trivial FD's whose determinant is not a supekey if the right hand side is contained in a candidate key.

Example 6.33: The relation GRADE of Figure 6.15 is not in the BCNF because of the dependencies *Student#* $\rightarrow$ *Name* and *Name* $\rightarrow$ *Student#* are nontrivial and their determinants are not superkeys of GRADE.

The following is an example of a BCNF relation.

Example 6.34: The relation scheme **STUDENT**(*SID*, *Name*, *Phone_No*, *Major*), where *SID* is an unique student identification number, and where *Name*, and *Phone_No* are assumed to be unique for this example. The functional dependencies satisfied on the **STUDENT** relation scheme are {*SID* $\rightarrow$ *Major*; *Name* $\rightarrow$ *Major*; *Phone_No* $\rightarrow$ *Major*; *SID* $\rightarrow$ *Name*; *SID* $\rightarrow$ *Phone_No*; *Name* $\rightarrow$ *SID*; *Name* $\rightarrow$ *Phone_No*; *Phone_No* $\rightarrow$ *SID*; *Phone_No* $\rightarrow$ *Name*}. The relation **STUDENT** is in BCNF since each FD involves a candidate key as the determinant.

Lossless Join Decomposition into Boyce Codd Normal Form

We now give an algorithm which decomposes a relation scheme into a number of relation schemes, each of which is in the Boyce Codd normal form. In algorithm 6.7, **S** is a set of relation schemes. It is initialized with the original relation scheme, which may not be in the BCNF. At the end of the algorithm, **S** will contain a number of BCNF relation schemes. We start off by finding a non-redundant cover, **F'**, of F. Then, we look at the relation schemes in **S** and find a scheme, let us say $\mathbf{R_j}$, which is not in the BCNF for a nontrivial FD $\mathbf{X \rightarrow Y}$ in **F'**. Since $\mathbf{R_j}$ is not in the BCNF, the conditions $\mathbf{XY} \subseteq \mathbf{R_j}$ and $\mathbf{X} \not\rightarrow \mathbf{R_j}$ will hold. We decompose $\mathbf{R_j}$ into two relations **XY**, and $\mathbf{R_j}$ - **Y**. The algorithm terminates with all relations in the set being in BCNF.

The decomposition is lossless and the join of the resulting relations will give the original relation. However, some of the dependencies in the original relation scheme 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.

Title: Algorithm 6.7: Lossless Boyce Codd Normal Form Decomposition Algorithm

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

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

Body:

```
begin
i := 0;
U := { R(S) };
all_BCNF := false;
Find F' from F; (* here F' is a non-redundant cover of
                                                F *)
while ( ¬all_BCNF ) do
 if there exist a nontrivial FD ( X → Y ) in F'+ such that
    XY ⊆ Rj and X ↛ Rj do  (* i.e., Rj, a relation scheme in U, 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 U
       Rj := Rj - Y;
  end;
    else  all_BCNF := true;
end;
```

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

Example 6.35: Let us 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*

Let us first find the nonredundant cover of the given set of FD's. There are no redundant FD'S in the set hence the given set of FD's is in nonredundant cover.

Now let us use the Algorithm 6.7 to find a lossless decomposition of **SHIPPING**. Since there is a FD *Ship* → *Capacity* and since *Ship* ↛ **SHIPPING** we replace **SHIPPING** with the relation $\mathbf{R}_1$(*Ship*, *Capacity*) formed involving the FD in question and $\mathbf{R}_2$(*Ship*, *Date*, *Cargo*, *Value*). Let us consider the relation $\mathbf{R}_2$: the FD *ShipDate* → *Cargo* is a non trivial FD in the non-redundant cover. However, since *ShipDate* → *ShipDateCargoValue*, the relation $\mathbf{R}_2$ is in BCNF form and we have completed the decomposition.

$\mathbf{R}_1$(*Ship*, *Capacity*) with the FD *Ship* →*Capacity*,

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

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

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

$\mathbf{R}_1$(*Cargo*, *Capacity*, *Value*) with the FD *CargoCapacity* → *Value*,

$\mathbf{R}_2$(*Ship*, *Capacity*) with the FD *Ship* → *Capacity*

$\mathbf{R}_3$(*Ship*, *Date*, *Cargo*) with the FD *ShipDate* → *Cargo*

This decomposition is also dependence preserving.

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

Example 6.36: Consider the relation scheme <(*ABCD*), {*AB* → *C*, *C* → *A*}>. Since none of the FD's are redundant, the given set is a non redundant cover. Using the FD *AB* → *C* we decompose this into the relation schemes: <(*ABC*), {*AB* → *C*, *C* → *A*}> and <(*ABD*), {}>. The scheme <(*ABC*), {*AB* → *C*, *C* → *A*}> can be further decomposed into the schemes: <(*AC*), {*C* → *A*}> and <(*BC*), {}>.

In Example 6.37, we demonstrate the non-uniqueness of the BCNF decomposition. We see from the this example that for different orders of considering the FD's, 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*), (*SSd*), (CC_d), (*CP*), (*RDTC*), (P_dRDT), (*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* → 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.

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 of Figure 6.7 gives a decomposition tree shown in Figure G. The left tree is obtained by considering the FD's 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*), (*SSd*), (CC_d), (*CP*), (*RDTC*), and (SGP_dRDT). The right decomposition is obtained by considering the FD $SC \rightarrow G$ first.

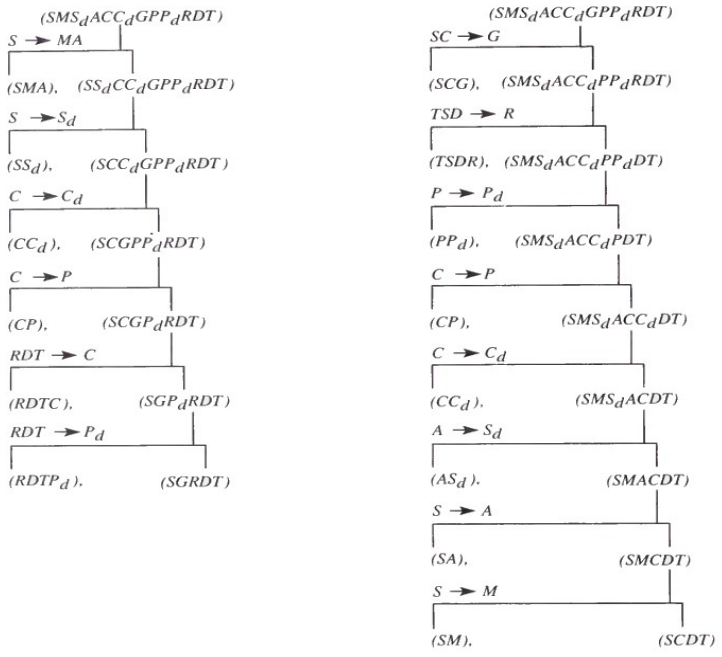


Figure G *Two different Decomposition Trees*

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

6.6 Concluding Remarks

Let us return to the relation STUDENT_ADVISOR(*Name*, *Department*, *Advisor*) of Example 6.19 and Figure E(a) 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 E(b) and E(c), 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 E(d), contains tuples that did not exist in the original relation of Figure E(i). In addition the decomposition is not dependency preserving. The FD *Name* → *Advisor* is not implied by the FD's of the decomposed relation nor could it be derived from their join.

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)

(a) ADVISOR_STUDENT,
(b) STUDENT_DEPARTMENT,
(c) ADVISOR_DEPARTMENT, and
(d) X = STUDENT_DEPARTMENT ⋈ADVISOR_DEPARTMENT.

Note the marked tuples are eliminated when X is joined with ADVISOR_STUDENT

Figure 6.16 *Decomposition of relation STUDENT_ADVISOR*

We notice, however, that there are three independent relationships in the STUDENT_ADVISOR

relation, and the only key of it is *Name, Advisor*. We can decompose it into three relations, ADVISOR_STUDENT(*Name, Advisor*), STUDENT_DEPARTMENT (*Name*, *Department*), and ADVISOR_DEPARTMENT(*Advisor*, *Department*). This decomposition of STUDENT_ADVISOR into three relations 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 FD's is preserved in one of the relations.

Note that some of these independent relationships which 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 below in parts a, b,and c of Figure 6.16. In the original relation, this data could have been only entered with null values for the unknown attribute.

The join of these relations, STUDENT_DEPARTMENT ⋈ADVISOR_DEPARTMENT, to obtain the STUDENT_ADVISOR relation gives us the tuples shown in Figure 6.16(d). It should be noted that the new tuples added in the decomposed relation participate in the joins as shown in Figure 6.16(d). 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, for this sequence of joins, are eliminated. 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 which contains the above three relations, there is a need to disambiguate 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.

The goal of the 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 dependence preserving when both lossless and dependence 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. The importance of having a consistent database without repetition of data is discussed, and the anomalies that could be introduced in the database with an undesirable design are pointed out. The criteria to be satisfied 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 modelled by the relation. These constraints are in the form of FD's. The approach we have used is to replace R by a set of more desirable relation schemes. In this chapter we have considered the decomposition approach. The synthesis approach is discussed in chapter 7. Regardless of the approach used, the criteria for the design are the following: content preserving and dependency preserving

The decomposition approach starts with one (the universal relation) 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, hence, 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 FD's only can be decomposed into relations which are in the third normal form; such a decomposition is lossless and preserves the dependencies. Any relation can also be decomposed losslessly into relations which are 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 dependence preserving. It must be noted that the goal of the decomposition approach to the relational database design using FD's is to come up with a database scheme which is in the BCNF, is lossless and preserves the original set of FD's. If this goal is not possible then the alternate goal is to derive a database scheme which is in the 3NF and is lossless and dependence preserving.

Key Terms

Armstrong's axioms
- inference axioms

anomalies
- deletion anomalies
- insertion anomalies
- update anomalies
- redundancy

attribute
- nonprime attribute
- prime attribute

covers
- canonical cover
- nonredundant cover
- closure

decomposition
- content preserving
- dependence preserving
- lossless join
- lossy - dangling tuples

functional dependency
- full functional dependency
- partial functional dependency
- transitive dependency

unnormalized relation

normal forms

normalization
- Boyce Codd normal form
- first normal form
- econd normal form
- third normal form
- redundancy

relation scheme

universal relation

Exercises:

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

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

6.3. Given **R**{*ABCDEF*} and the set of FD's on **R** is 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 the 3NF, decompose **R** and find a set of 3NF projections of **R**. Is this set lossless and dependence preserving?

6.4. Given the relation scheme **R**{*Truck*(*T*), *Capacity*(*C*), *Date*(*Y*), *Cargo*(*G*), *Destination*(*D*), *Value*(*V*)} with the following FD's{$T \rightarrow C$, $(T,Y) \rightarrow G$, $(T,Y) \rightarrow D$, $(C,G) \rightarrow V$}. 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, then find a BCNF decomposition of **R**.

6.5.Consider a relation scheme **R** with the following set of attributes and FD's. {*SID*, *Name*, *Date_of_Birth*, *Advisor*, *Department*, *Term*, *Year*, *Course*, *Grade*}, {*SID* → *Name*, *Date_of_Birth*, *Advisor*, *Department*; *Advisor* → *Department*; *SID*, *Term*, *Year*, *Course* → *Grade*}. Find the candidate keys of **R**. Does a dependence preserving and lossless join decomposition of **R** into a number BCNF relation 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* → *BCD*, *CD* → *E*, *E* → *CD*, *D* → *AH*, *ABH* → *BD*, *DH* → *BC*}. Find a nonredundant cover. Is this the only nonredundant cover?

6.8. Given: **R**{*ABCDEFGH*} with the FD's given by{ *A* → *BCDEFGH*, *BCD* → *AEFGH*, *BCE* → *ADEFGH*, *CE* → *H*, *CD* → *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* → *K*, *AI* → *BFG*, *IC* → *ADE*, *BIG* → *CJ*, *K* → *HA*}. Find a canonical cover of this set of FD's. Find a dependence preserving and lossless join 3NF decomposition of **R**. Is there a BCNF decomposition of **R** which is both dependence preserving and also lossless? If so, find one such decomposition.

6.10. Given the relation **R** {*ABCDE*} with the FD's {*A* → *BCDE*, *B* → *ACDE*, *C* → *ABDE*}. Give the lossless decomposition of **R**.

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

6.12. Does another canonical cover of the set of FD's 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* →*CE*, *B* → *D*, *C* → *ADE*, *BD* → *F* }. Find the closure of *BCD*.

6.14. Explain why there is a renewed interest in unnormalized relations (which is dubbed as the non-1NF or NFNF). What are its advantages compared to normalized relations?

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

6.16. The Sky-high-returns Mutual Fund (SMF) Corpn. 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 annum 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 the 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. Suppose we assume that *Room_Cap* $\not\rightarrow$ *Enrol_Lmt*. This means that two different courses allocated to the same room at different day and time could have different *Enrol_Lmt*'s. What normal form is TEACHES in under this modified assumption? If it's not in the 3NF form, find a lossless and dependency preserving decomposition.

6.18. Consider the relation scheme **R**(*ABCDE*) and the FD's $\{A \rightarrow B, C \rightarrow D, A \rightarrow E\}$. 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): *Faculty → Dean, Dean → Faculty. Department → Chairperson, Professor → RankChairperson Department → Faculty, Student → DepartmentFacultyDean, ProfessorRank → DepartmentFaculty*

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 will you aim for and why?

6.21. Use the algorithm 6.4 to determine if the decomposition of STUDENT_ADVISOR(*Name*, *Department*, *Advisor*) with the functional dependencies **F**{*Name → Department*, *Name → Advisor*, *Advisor → 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 as to the FD's 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 as to the FD's involved, can you determine its normal form? Justify your answer.

6.24. Prove that the Armstrong axioms **F1** through **F3** are sound. (Hint: if $\mathbf{X} \rightarrow \mathbf{Y}$ is derived from F using the Armstrong axiom, then the dependency $\mathbf{X} \rightarrow \mathbf{Y}$ is satisfied in any relation that satisfies the dependencies in F.)

6.25. Prove that the algorithm of Figure 6.9 correctly computes $\mathbf{X}^+$.

6.26. Prove that $\mathbf{X} \rightarrow \mathbf{Y}$ follows from the inference axioms **F1** through **F3**, if and only if, $\mathbf{Y} \overline{\in} \mathbf{X}^+$.

Bibliographic Notes

Codd [Codd70] studied functional dependencies and the third normal form. The BCNF was introduced in [Codd72], and the axioms for functional dependencies were due to Armstrong[Arms74]. [Beer77] gives a set of axioms for FD's and MVDs and prove the completeness and soundness of this set. The linear membership algorithm for functional dependencies was presented in [Beer79]. An algorithm to derive a minimum cover was given in [Maier80].

The universal relation concept and the associated problems were first discussed in [Kent81]. The formal proof of theorem on lossless join and dependence preserving third normal form decomposition is given in [Bisk79]. The algorithm for testing for lossless join is based on [Aho79]. A more efficient algorithm is given in [Liu80]. An algorithm for testing the preservation of dependency is presented in [Beer81]. The complexity of finding whether a relation is in the BCNF is discussed in [Beer79a]. Recent results from the NFNF (non-1NF) relations are presented in [Ozso87] and [Roth88].

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

including relational database design.

Bibliography

[Aho79] Aho, A. V., Beeri, C., Ullman, J.D., 'The theory of Joins in Relational Databses', ACM TODS, Vol. 8-2, June 1979, pp. 297-314, Corregendum: ACM TODS, Vol. 8-2, June 1979, p287.
[Arms74] Armstrong, W. W.,"Dependency Structures of Database Relationships", Proc. of the IFIP (1974), pp.580-583.
[Beer77] Beeri,C., Fagin R., Howard, J.H., "A complete axiomatization for functional and multivalued dependencies", Proc of ACM SIGMOD International Symposium on Management of Data, 1977, pp.47-61.
[Beer79] Beeri, C., Bernstein, P.A., "Computational Problems related to the design of normal form relational schemes", ACM TODS, Vol. 4-1, March 1979, pp.113-124.
[Beer80] Beeri, C., "On the Membership Problem for Functional and Multivalued Dependencies in Relational Databases", ACM TODS, Vol. 5-3, September 1980, pp.241-259.
[Beer81] Beeri, C., Honeyman, P., 'Preserving Functional Dependencies', SIAM Journal of Computing, Vol. 10-3, pp. 647-656.
[Bisk79] Biskup, J., Dayal, U., Bernstein, P.A., "Synthesizing Independent Database Schemas" Proc. ACM SIGMOD International Symposium on Management of Data, 1979, pp.143-152.
[Bros88] Brosda, V., Vossen, G., 'Update and Retrieval in a Relational Database Through a Universal Schema Interface', ACM TODS, Vol. 13-4, December 1988, pp.449-485.
[Codd70] Codd, E.F., "A Relational Model for large shared data banks", Communications of the ACM, Vol. 13-6, June 1970, pp.377-387.
[Codd72] Codd, E.F., "Further normalization of the data base relational model" in 'Data Base Systems' ed. R. Rustin, Prentice-Hall, Englewood Cliffs, 1972, pp.33-64.
[Date85] Date, C.J., An Introduction to Database Systems, Vol. 1., Fourth edition, Addison Wesley, Reading, MA, 1985.
[Delo78] Delobel, C., "Normalization and Hierarchical Dependencies in the Relational Data Model" ACM TODS, Vol. 3-3, September 1978, pp.201-22.
[Fagi77] Fagin, R.,"Multivalued Dependencies and a New Normal Form for Relational Databases", ACM TODS, Vol. 2-3, September 1977, pp.262-278.
[Fagi79] Fagin, R., "Normal Forms and Relational Database Operators", ACM SIGMOD International Symposium on Management of Data, 1979, pp.153-160.
[Fagi81] Fagin, R., "A Normal Form for Relational Databases that is based on Domains and Keys", ACM TODS, Vol. 6-3, September 1982, pp.387-415.
[Kent81] Kent, W., "Consequences of Assuming a Universal Relation", ACM TODS, Vol6-4, December 1981, pp.539-556.
[Kort86] Korth, H.F., Silberschatz, A., Database Systems Concepts, McGraw Hill, New York, 1986.
[Lien81] Lien, Y. E.,"Hierarchical Schemata for Relational Databases", ACM TODS, Vol. 6-1, March 1981, pp.48-69.
[Lien85] Lien, Y. E.,"Relational Database Design", in Principles of Database Design, ed. S. Bing Yao. Prentice-Hall, Englewood Cliffs, 1985.
[Liu80]Liu, L., Demers, A., 'An Algorithm for Testing Lossless Joins in Relational Databases', Information Processing Letters, Vol. 11-1, pp. 73-76.
[Maie80] Maier, D., "Minimum Covers in the Relational Database Model", Journal of the ACM, Vol. 27-4, October 1980, pp.664-674.
[Maie83] Maier, D., The Theory of Relational Databases, Computer Science Press, Rockville, MD, 1983.
[Ozso87] Ozsoyoglu, Z. M., Li-Yan Yuan, 'A New Normal Form for Nested Relations', ACM TODS, Vol. 12-1, March 1987, pp.111-136.

[Riss79] Rissanen, J., "Theory of Joins for Relational Databases - A Tutorial Survey", Proc. Seventh Symposium on Mathematical Foundations of Computer Science, Lecture notes in Computer Science 64, Springer-Verlag, pp. 537-551.
[Roth88] Roth, M. A., Korth, H. F., Silberschatz, A., 'Extended Algebra and Calculus for Nested Relational Databases', ACM TODS, Vol. 13-4, December 1988, pp. 389-417.
[Ullm82] Ullman, Jeffrey D., Principle of Database Systems, 2nd edition, Computer Science Press, Rockville, MD.,1982.
[Zani81] Zaniolo, C., Melkanoff, M.A.,"On the Design of Relational Database Schemata", ACM TODS, Vol. 6-1, March 1981, pp.1-47.

Notes