CHAPTER

4

Logical Database Design and the Relational Model

Visit www.pearsonhighered.com/hoffer to view the accompanying video for this chapter.

Learning Objectives

After studying this chapter, you should be able to:

- Concisely define each of the following key terms: **relation, primary key, composite key, foreign key, null, entity integrity rule, referential integrity constraint, well-structured relation, anomaly, surrogate primary key, recursive foreign key, normalization, normal form, functional dependency, determinant, candidate key, first normal form, second normal form, partial functional dependency, third normal form, transitive dependency, synonyms, alias, homonym,** and **enterprise key.**
- List five properties of relations.
- State two essential properties of a candidate key.
- Give a concise definition of each of the following: first normal form, second normal form, and third normal form.
- Briefly describe four problems that may arise when merging relations.
- Transform an E-R (or EER) diagram into a logically equivalent set of relations.
- Create relational tables that incorporate entity integrity and referential integrity constraints.
- Use normalization to decompose a relation with anomalies into well-structured relations.

INTRODUCTION

In this chapter, we describe logical database design, with special emphasis on the relational data model. Logical database design is the process of transforming the conceptual data model (described in Chapters 2 and 3) into a logical data model—one that is consistent and compatible with a specific type of database technology. An experienced database designer often will do logical database design in parallel with conceptual data modeling if he or she knows the type of database technology that will be used. It is, however, important to treat these as separate steps so that you concentrate on each important part of database development. Conceptual data modeling is about understanding the organization—getting the right requirements. Logical database design is about creating stable database structures—correctly expressing the requirements in a technical language. Both are important steps that must be performed carefully.

Although there are other data models, we have two reasons for emphasizing the relational data model in this chapter. First, the relational data model is by far the one most commonly used in contemporary database applications. Second, some of the principles of logical database design for the relational model apply to the other logical models as well.

We have introduced the relational data model informally through simple examples in earlier chapters. It is important, however, to note that the relational data model is a form of logical data model, and as such it is different from the conceptual data models. Thus, an E-R data model is not a relational data model, and an E-R model may not obey the rules for a well-structured relational data model, called *normalization*, which we explain in this chapter. That is okay, because the E-R model was developed for other purposes—understanding data requirements and business rules about the data—not structuring the data for sound database processing, which is the goal of logical database design.

In this chapter, we first define the important terms and concepts for the relational data model. (We often use the abbreviated term *relational model* when referring to the relational data model.) We next describe and illustrate the process of transforming an EER model into the relational model. Many CASE tools support this transformation today at the technical level; however, it is important that you understand the underlying principles and procedures. We then describe the concepts of normalization in detail. Normalization, which is the process of designing well-structured relations, is an important component of logical design for the relational model. Finally, we describe how to merge relations while avoiding common pitfalls that may occur in this process.

The objective of logical database design is to translate the conceptual design (which represents an organization's requirements for data) into a logical database design that can be implemented via a chosen database management system. The resulting databases must meet user needs for data sharing, flexibility, and ease of access. The concepts presented in this chapter are essential to your understanding of the database development process.

THE RELATIONAL DATA MODEL

The relational data model was first introduced in 1970 by E. F. Codd, then of IBM (Codd, 1970). Two early research projects were launched to prove the feasibility of the relational model and to develop prototype systems. The first of these, at IBM's San Jose Research Laboratory, led to the development of System R (a prototype relational DBMS [RDBMS]) during the late 1970s. The second, at the University of California at Berkeley, led to the development of Ingres, an academically oriented RDBMS. Commercial RDBMS products from numerous vendors started to appear about 1980. (See the Web site for this book for links to RDBMS and other DBMS vendors.) Today RDBMSs have become the dominant technology for database management, and there are literally hundreds of RDBMS products for computers ranging from smartphones and personal computers to mainframes.

Basic Definitions

The relational data model represents data in the form of tables. The relational model is based on mathematical theory and therefore has a solid theoretical foundation. However, we need only a few simple concepts to describe the relational model. Therefore, it can be easily understood and used even by those unfamiliar with the underlying theory. The relational data model consists of the following three components (Fleming and von Halle, 1989):

1. ***Data structure*** Data are organized in the form of tables, with rows and columns.
2. ***Data manipulation*** Powerful operations (using the SQL language) are used to manipulate data stored in the relations.
3. ***Data integrity*** The model includes mechanisms to specify business rules that maintain the integrity of data when they are manipulated.

EMPLOYEE1

EmpID	Name	DeptName	Salary
100	Margaret Simpson	Marketing	48,000
140	Allen Beeton	Accounting	52,000
110	Chris Lucero	Info Systems	43,000
190	Lorenzo Davis	Finance	55,000
150	Susan Martin	Marketing	42,000

FIGURE 4-1 EMPLOYEE1 relation with sample data

We discuss data structure and data integrity in this section. Data manipulation is discussed in Chapters 6, 7, and 8.

RELATIONAL DATA STRUCTURE A **relation** is a named, two-dimensional table of data. Each relation (or table) consists of a set of named columns and an arbitrary number of unnamed rows. An attribute, consistent with its definition in Chapter 2, is a named column of a relation. Each row of a relation corresponds to a record that contains data (attribute) values for a single entity. Figure 4-1 shows an example of a relation named EMPLOYEE1. This relation contains the following attributes describing employees: EmpID, Name, DeptName, and Salary. The five rows of the table correspond to five employees. It is important to understand that the sample data in Figure 4-1 are intended to illustrate the structure of the EMPLOYEE1 relation; they are not part of the relation itself. Even if we add another row of data to the figure or change any of the data in the existing rows, it is still the same EMPLOYEE1 relation. Nor does deleting a row change the relation. In fact, we could delete all of the rows shown in Figure 4-1, and the EMPLOYEE1 relation would still exist. In other words, Figure 4-1 is an instance of the EMPLOYEE1 relation.

Relation
A named two-dimensional table of data.

We can express the structure of a relation by using a shorthand notation in which the name of the relation is followed (in parentheses) by the names of the attributes in that relation. For EMPLOYEE1 we would have

EMPLOYEE1(EmpID, Name, DeptName, Salary)

RELATIONAL KEYS We must be able to store and retrieve a row of data in a relation, based on the data values stored in that row. To achieve this goal, every relation must have a primary key. A **primary key** is an attribute or a combination of attributes that uniquely identifies each row in a relation. We designate a primary key by underlining the attribute name(s). For example, the primary key for the relation EMPLOYEE1 is EmpID. Notice that this attribute is underlined in Figure 4-1. In shorthand notation, we express this relation as follows:

Primary key
An attribute or a combination of attributes that uniquely identifies each row in a relation.

EMPLOYEE1(EmpID, Name, DeptName, Salary)

The concept of a primary key is related to the term *identifier* defined in Chapter 2. The same attribute or a collection of attributes indicated as an entity's identifier in an E-R diagram may be the same attributes that compose the primary key for the relation representing that entity. There are exceptions: For example, associative entities do not have to have an identifier, and the (partial) identifier of a weak entity forms only part of a weak entity's primary key. In addition, there may be several attributes of an entity that may serve as the associated relation's primary key. All of these situations will be illustrated later in this chapter.

A **composite key** is a primary key that consists of more than one attribute. For example, the primary key for a relation DEPENDENT would likely consist of the combination EmpID and DependentName. We show several examples of composite keys later in this chapter.

Composite key
A primary key that consists of more than one attribute.

Foreign key
An attribute in a relation that serves as the primary key of another relation in the same database.

Often we must represent the relationship between two tables or relations. This is accomplished through the use of foreign keys. A **foreign key** is an attribute (possibly composite) in a relation that serves as the primary key of another relation. For example, consider the relations EMPLOYEE1 and DEPARTMENT:

```
EMPLOYEE1(EmpID, Name, DeptName, Salary)
DEPARTMENT(DeptName, Location, Fax)
```

The attribute DeptName is a foreign key in EMPLOYEE1. It allows a user to associate any employee with the department to which he or she is assigned. Some authors emphasize the fact that an attribute is a foreign key by using a dashed underline, like this:

```
EMPLOYEE1(EmpID, Name, DeptName, Salary)
```

We provide numerous examples of foreign keys in the remainder of this chapter and discuss the properties of foreign keys under the heading "Referential Integrity."

PROPERTIES OF RELATIONS We have defined relations as two-dimensional tables of data. However, not all tables are relations. Relations have several properties that distinguish them from non-relational tables. We summarize these properties next:

1. Each relation (or table) in a database has a unique name.
2. An entry at the intersection of each row and column is atomic (or single valued). There can be only one value associated with each attribute on a specific row of a table; no multivalued attributes are allowed in a relation.
3. Each row is unique; no two rows in a relation can be identical.
4. Each attribute (or column) within a table has a unique name.
5. The sequence of columns (left to right) is insignificant. The order of the columns in a relation can be changed without changing the meaning or use of the relation.
6. The sequence of rows (top to bottom) is insignificant. As with columns, the order of the rows of a relation may be changed or stored in any sequence.

REMOVING MULTIVALUED ATTRIBUTES FROM TABLES The second property of relations listed in the preceding section states that no multivalued attributes are allowed in a relation. Thus, a table that contains one or more multivalued attributes is not a relation. For example, Figure 4-2a shows the employee data from the EMPLOYEE1 relation extended to include courses that may have been taken by those employees. Because a given employee may have taken more than one course, the attributes CourseTitle and DateCompleted are multivalued attributes. For example, the employee with EmpID 100 has taken two courses. If an employee has not taken any courses, the CourseTitle and DateCompleted attribute values are null. (See the employee with EmpID 190 for an example.)

We show how to eliminate the multivalued attributes in Figure 4-2b by filling the relevant data values into the previously vacant cells of Figure 4-2a. As a result, the table in Figure 4-2b has only single-valued attributes and now satisfies the atomic property of relations. The name EMPLOYEE2 is given to this relation to distinguish it from EMPLOYEE1. However, as you will see, this new relation does have some undesirable properties.

Sample Database

A relational database may consist of any number of relations. The structure of the database is described through the use of a schema (defined in Chapter 1), which is a description of the overall logical structure of the database. There are two common methods for expressing a schema:

a. Short text statements, in which each relation is named and the names of its attributes follow in parentheses. (See the EMPLOYEE1 and DEPARTMENT relations defined earlier in this chapter.)

FIGURE 4-2 Eliminating multivalued attributes

(a) Table with repeating groups

EmpID	Name	DeptName	Salary	CourseTitle	DateCompleted
100	Margaret Simpson	Marketing	48,000	SPSS	6/19/201X
				Surveys	10/7/201X
140	Alan Beeton	Accounting	52,000	Tax Acc	12/8/201X
110	Chris Lucero	Info Systems	43,000	Visual Basic	1/12/201X
				C++	4/22/201X
190	Lorenzo Davis	Finance	55,000		
150	Susan Martin	Marketing	42,000	SPSS	6/16/201X
				Java	8/12/201X

(b) EMPLOYEE2 relation

EMPLOYEE2

EmpID	Name	DeptName	Salary	CourseTitle	DateCompleted
100	Margaret Simpson	Marketing	48,000	SPSS	6/19/201X
100	Margaret Simpson	Marketing	48,000	Surveys	10/7/201X
140	Alan Beeton	Accounting	52,000	Tax Acc	12/8/201X
110	Chris Lucero	Info Systems	43,000	Visual Basic	1/12/201X
110	Chris Lucero	Info Systems	43,000	C++	4/22/201X
190	Lorenzo Davis	Finance	55,000		
150	Susan Martin	Marketing	42,000	SPSS	6/19/201X
150	Susan Martin	Marketing	42,000	Java	8/12/201X

b. A graphical representation, in which each relation is represented by a rectangle containing the attributes for the relation

Text statements have the advantage of simplicity. However, a graphical representation provides a better means of expressing referential integrity constraints (as you will see shortly). In this section, we use both techniques for expressing a schema so that you can compare them.

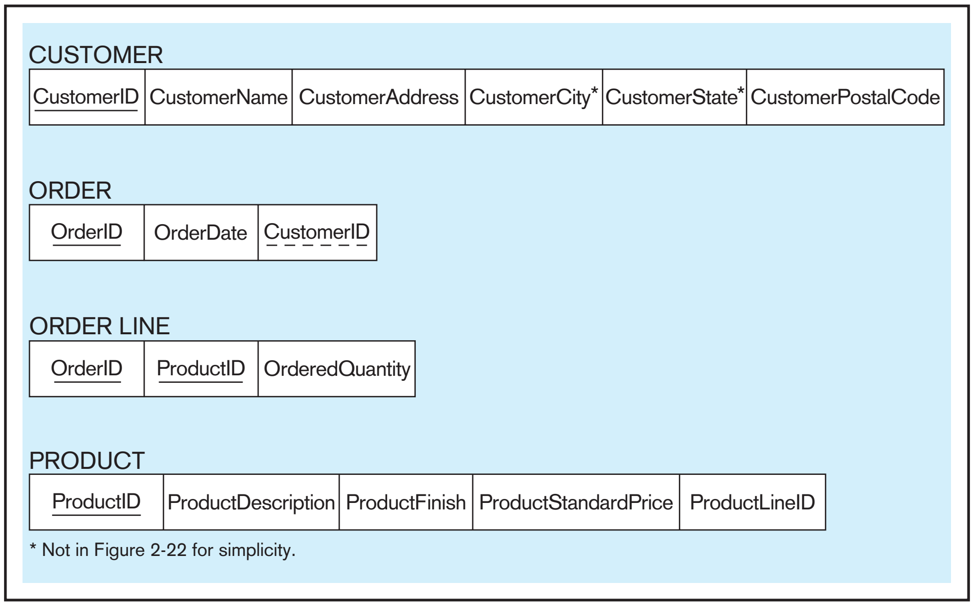
A schema for four relations at Pine Valley Furniture Company is shown in Figure 4-3. The four relations shown in this figure are CUSTOMER, ORDER, ORDER LINE, and PRODUCT. The key attributes for these relations are underlined, and other important attributes are included in each relation. We show how to design these relations using the techniques of normalization later in this chapter.

PINE VALLEY FURNITURE

Following is a text description of these relations:

CUSTOMER(CustomerID, CustomerName, CustomerAddress,
  CustomerCity, CustomerState, CustomerPostalCode)
ORDER(OrderID, OrderDate, CustomerID)
ORDER LINE(OrderID, ProductID, OrderedQuantity)
PRODUCT(ProductID, ProductDescription, ProductFinish,
  ProductStandardPrice, ProductLineID)

Notice that the primary key for ORDER LINE is a composite key consisting of the attributes OrderID and ProductID. Also, CustomerID is a foreign key in the ORDER relation; this allows the user to associate an order with the customer who submitted the order. ORDER LINE has two foreign keys: OrderID and ProductID. These keys allow the user to associate each line on an order with the relevant order and product.



CUSTOMER

CustomerID	CustomerName	CustomerAddress	CustomerCity*	CustomerState*	CustomerPostalCode

ORDER

OrderID	OrderDate	CustomerID

ORDER LINE

OrderID	ProductID	OrderedQuantity

PRODUCT

ProductID	ProductDescription	ProductFinish	ProductStandardPrice	ProductLineID

* Not in Figure 2-22 for simplicity.

FIGURE 4-3 Schema for four relations (Pine Valley Furniture Company)

An instance of this database is shown in Figure 4-4. This figure shows four tables with sample data. Notice how the foreign keys allow us to associate the various tables. It is a good idea to create an instance of your relational schema with sample data for four reasons:

1. The sample data allow you to test your assumptions regarding the design.
2. The sample data provide a convenient way to check the accuracy of your design.
3. The sample data help improve communications with users in discussing your design.
4. You can use the sample data to develop prototype applications and to test queries.

INTEGRITY CONSTRAINTS

The relational data model includes several types of constraints, or rules limiting acceptable values and actions, whose purpose is to facilitate maintaining the accuracy and integrity of data in the database. The major types of integrity constraints are domain constraints, entity integrity, and referential integrity.

Domain Constraints

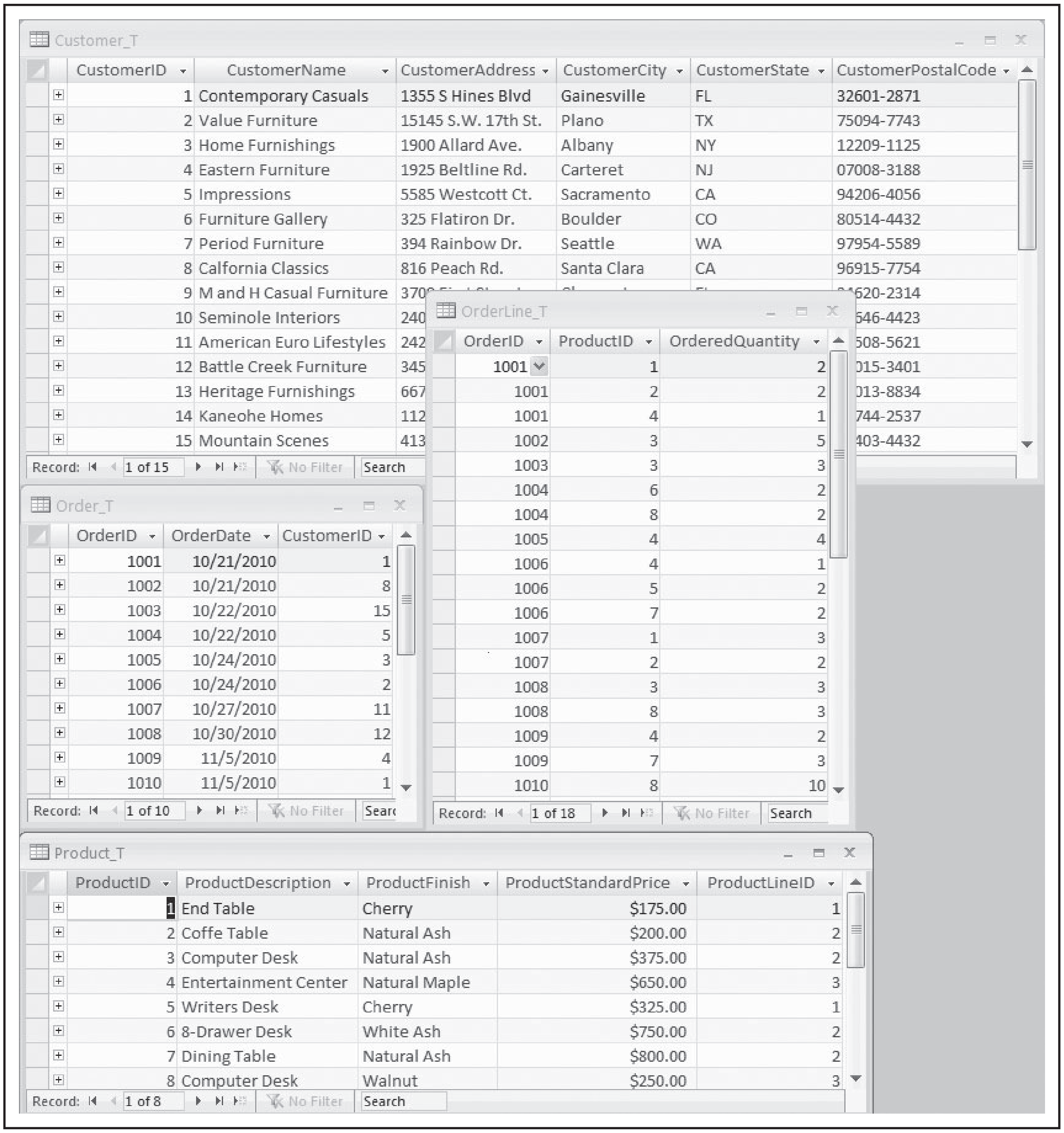
All of the values that appear in a column of a relation must be from the same domain. A domain is the set of values that may be assigned to an attribute. A domain definition usually consists of the following components: domain name, meaning, data type, size (or length), and allowable values or allowable range (if applicable). Table 4-1 (page 162) shows domain definitions for the domains associated with the attributes in Figures 4-3 and 4-4.

Entity Integrity

The entity integrity rule is designed to ensure that every relation has a primary key and that the data values for that primary key are all valid. In particular, it guarantees that every primary key attribute is non-null.

In some cases, a particular attribute cannot be assigned a data value. There are two situations in which this is likely to occur: Either there is no applicable data value or the applicable data value is not known when values are assigned. Suppose, for example, that you fill out an employment form that has a space reserved for a fax number. If you have no fax number, you leave this space empty because it does not apply to you. Or suppose that you are asked to fill in the telephone number of your previous employer. If you do not recall this number, you may leave it empty because that information is not known.

FIGURE 4-4 **Instance of a relational schema (Pine Valley Furniture Company)**



Customer_T

CustomerID	CustomerName	CustomerAddress	CustomerCity	CustomerState	CustomerPostalCode
1	Contemporary Casuals	1355 S Hines Blvd	Gainesville	FL	32601-2871
2	Value Furniture	15145 S.W. 17th St.	Plano	TX	75094-7743
3	Home Furnishings	1900 Allard Ave.	Albany	NY	12209-1125
4	Eastern Furniture	1925 Beltline Rd.	Carteret	NJ	07008-3188
5	Impressions	5585 Westcott Ct.	Sacramento	CA	94206-4056
6	Furniture Gallery	325 Flatiron Dr.	Boulder	CO	80514-4432
7	Period Furniture	394 Rainbow Dr.	Seattle	WA	97954-5589
8	Calfornia Classics	816 Peach Rd.	Santa Clara	CA	96915-7754
9	M and H Casual Furniture	370			620-2314
10	Seminole Interiors	240			546-4423
11	American Euro Lifestyles	242			508-5621
12	Battle Creek Furniture	345			015-3401
13	Heritage Furnishings	667			013-8834
14	Kaneohe Homes	112			744-2537
15	Mountain Scenes	413			403-4432

Record: 1 of 15 No Filter Search

Order_T

OrderID	OrderDate	CustomerID
1001	10/21/2010	1
1002	10/21/2010	8
1003	10/22/2010	15
1004	10/22/2010	5
1005	10/24/2010	3
1006	10/24/2010	2
1007	10/27/2010	11
1008	10/30/2010	12
1009	11/5/2010	4
1010	11/5/2010	1

Record: 1 of 10 No Filter Searc

OrderLine_T

OrderID	ProductID	OrderedQuantity
1001	1	2
1001	2	2
1001	4	1
1002	3	5
1003	3	3
1004	6	2
1004	8	2
1005	4	4
1006	4	1
1006	5	2
1006	7	2
1007	1	3
1007	2	2
1008	3	3
1008	8	3
1009	4	2
1009	7	3
1010	8	10

Record: 1 of 18 No Filter Search

Product_T

ProductID	ProductDescription	ProductFinish	ProductStandardPrice	ProductLineID
1	End Table	Cherry	$175.00	1
2	Coffe Table	Natural Ash	$200.00	2
3	Computer Desk	Natural Ash	$375.00	2
4	Entertainment Center	Natural Maple	$650.00	3
5	Writers Desk	Cherry	$325.00	1
6	8-Drawer Desk	White Ash	$750.00	2
7	Dining Table	Natural Ash	$800.00	2
8	Computer Desk	Walnut	$250.00	3

Record: 1 of 8 No Filter Search

The relational data model allows us to assign a null value to an attribute in the just described situations. A **null** is a value that may be assigned to an attribute when no other value applies or when the applicable value is unknown. In reality, a null is not a value but rather it indicates the absence of a value. For example, it is not the same as a numeric zero or a string of blanks. The inclusion of nulls in the relational model is somewhat controversial, because it sometimes leads to anomalous results (Date, 2003). However, Codd, the inventor of the relational model, advocates the use of nulls for missing values (Codd, 1990).

Null
A value that may be assigned to an attribute when no other value applies or when the applicable value is unknown.

Everyone agrees that primary key values must not be allowed to be null. Thus, the **entity integrity rule** states the following: No primary key attribute (or component of a primary key attribute) may be null.

Entity integrity rule
A rule that states that no primary key attribute (or component of a primary key attribute) may be null.

TABLE 4-1 Domain Definitions for INVOICE Attributes

Attribute	Domain Name	Description	Domain
CustomerID	Customer IDs	Set of all possible customer IDs	character: size 5
CustomerName	Customer Names	Set of all possible customer names	character: size 25
CustomerAddress	Customer Addresses	Set of all possible customer addresses	character: size 30
CustomerCity	Cities	Set of all possible cities	character: size 20
CustomerState	States	Set of all possible states	character: size 2
CustomerPostalCode	Postal Codes	Set of all possible postal zip codes	character: size 10
OrderID	Order IDs	Set of all possible order IDs	character: size 5
OrderDate	Order Dates	Set of all possible order dates	date: format mm/dd/yy
ProductID	Product IDs	Set of all possible product IDs	character: size 5
ProductDescription	Product Descriptions	Set of all possible product descriptions	character: size 25
ProductFinish	Product Finishes	Set of all possible product finishes	character: size 15
ProductStandardPrice	Unit Prices	Set of all possible unit prices	monetary: 6 digits
ProductLineID	Product Line IDs	Set of all possible product line IDs	integer: 3 digits
OrderedQuantity	Quantities	Set of all possible ordered quantities	integer: 3 digits

Referential Integrity

In the relational data model, associations between tables are defined through the use of foreign keys. For example, in Figure 4-4, the association between the CUSTOMER and ORDER tables is defined by including the CustomerID attribute as a foreign key in ORDER. This of course implies that before we insert a new row in the ORDER table, the customer for that order must already exist in the CUSTOMER table. If you examine the rows in the ORDER table in Figure 4-4, you will find that every customer number for an order already appears in the CUSTOMER table.

Referential integrity constraint
A rule that states that either each foreign key value must match a primary key value in another relation or the foreign key value must be null.

A **referential integrity constraint** is a rule that maintains consistency among the rows of two relations. The rule states that if there is a foreign key in one relation, either each foreign key value must match a primary key value in another relation or the foreign key value must be null. You should examine the tables in Figure 4-4 to check whether the referential integrity rule has been enforced.

The graphical version of the relational schema provides a simple technique for identifying associations where referential integrity must be enforced. Figure 4-5 shows the schema for the relations introduced in Figure 4-3. An arrow has been drawn from each foreign key to the associated primary key. A referential integrity constraint must be defined for each of these arrows in the schema.

How do you know whether a foreign key is allowed to be null? If each order must have a customer (a mandatory relationship), then the foreign key CustomerID cannot be null in the ORDER relation. If the relationship is optional, then the foreign key could be null. Whether a foreign key can be null must be specified as a property of the foreign key attribute when the database is defined.

Actually, whether a foreign key can be null is more complex to model on an E-R diagram and to determine than we have shown so far. For example, what happens to order data if we choose to delete a customer who has submitted orders? We may want to see sales even if we do not care about the customer any more. Three choices are possible:

1. Delete the associated orders (called a cascading delete), in which case we lose not only the customer but also all the sales history
2. Prohibit deletion of the customer until all associated orders are first deleted (a safety check)

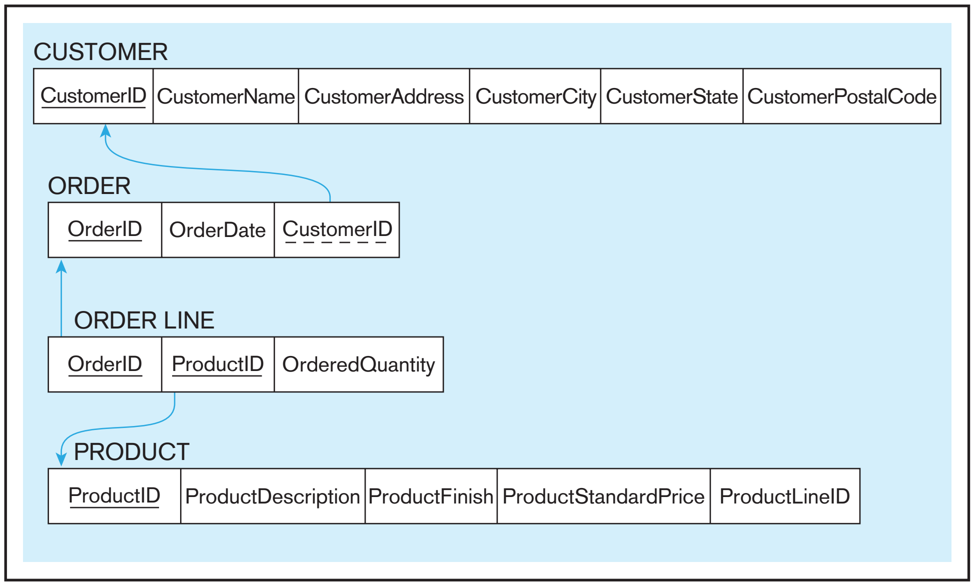


FIGURE 4-5 Referential integrity constraints (Pine Valley Furniture Company)

3. Place a null value in the foreign key (an exception that says although an order must have a CustomerID value when the order is created, CustomerID can become null later if the associated customer is deleted)

We will see how each of these choices is implemented when we describe the SQL database query language in Chapter 6. Please note that in practice, organizational rules and various regulations regarding data retention often determine what data can be deleted and when, and they therefore govern the choice between various deletion options.

Creating Relational Tables

In this section, we create table definitions for the four tables shown in Figure 4-5. These definitions are created using **CREATE TABLE** statements from the SQL data definition language. In practice, these table definitions are actually created during the implementation phase later in the database development process. However, we show these sample tables in this chapter for continuity and especially to illustrate the way the integrity constraints described previously are implemented in SQL.

The SQL table definitions are shown in Figure 4-6. One table is created for each of the four relations shown in the relational schema (Figure 4-5). Each attribute for a table is then defined. Notice that the data type and length for each attribute is taken from the domain definitions (Table 4-1). For example, the attribute CustomerName in the Customer_T table is defined as VARCHAR (variable character) data type with length 25. By specifying **NOT NULL**, each attribute can be constrained from being assigned a null value.

The primary key is specified for each table using the **PRIMARY KEY** clause at the end of each table definition. The OrderLine_T table illustrates how to specify a primary key when that key is a composite attribute. In this example, the primary key of OrderLine_T is the combination of OrderID and ProductID. Each primary key attribute in the four tables is constrained with **NOT NULL**. This enforces the entity integrity constraint described in the previous section. Notice that the **NOT NULL** constraint can also be used with non-primary-key attributes.

Referential integrity constraints are easily defined, using the graphical schema shown in Figure 4-5. An arrow originates from each foreign key and points to the related primary key in the associated relation. In the SQL table definition, a **FOREIGN KEY REFERENCES** statement corresponds to each of these arrows. Thus, for the table Order_T, the foreign key CustomerID references the primary key of Customer_T, which is also called CustomerID. Although in this case the foreign key and primary keys have the same name, this need not be the case. For example, the foreign key attribute could be named CustNo instead of CustomerID. However, the foreign and primary keys must be from the same domain.

FIGURE 4-6 SQL table definitions

```
CREATE TABLE Customer_T
            (CustomerID                     NUMBER(11,0)      NOT NULL,
            CustomerName                    VARCHAR2(25)      NOT NULL,
            CustomerAddress                 VARCHAR2(30),
            CustomerCity                    VARCHAR2(20),
            CustomerState                   CHAR(2),
            CustomerPostalCode              VARCHAR2(9),
CONSTRAINT Customer_PK PRIMARY KEY (CustomerID));

CREATE TABLE Order_T
            (OrderID                        NUMBER(11,0)      NOT NULL,
            OrderDate                       DATE DEFAULT SYSDATE,
            CustomerID                      NUMBER(11,0),
CONSTRAINT Order_PK PRIMARY KEY (OrderID),
CONSTRAINT Order_FK FOREIGN KEY (CustomerID) REFERENCES Customer_T (CustomerID));

CREATE TABLE Product_T
            (ProductID                      NUMBER(11,0)      NOT NULL,
            ProductDescription              VARCHAR2(50),
            ProductFinish                   VARCHAR2(20),
            ProductStandardPrice            DECIMAL(6,2),
            ProductLineID                   NUMBER(11,0),
CONSTRAINT Product_PK PRIMARY KEY (ProductID));

CREATE TABLE OrderLine_T
            (OrderID                        NUMBER(11,0)      NOT NULL,
            ProductID                       NUMBER(11,0)      NOT NULL,
            OrderedQuantity                 NUMBER(11,0),
CONSTRAINT OrderLine_PK PRIMARY KEY (OrderID, ProductID),
CONSTRAINT OrderLine_FK1 FOREIGN KEY (OrderID) REFERENCES Order_T (OrderID),
CONSTRAINT OrderLine_FK2 FOREIGN KEY (ProductID) REFERENCES Product_T (ProductID));
```

The OrderLine_T table provides an example of a table that has two foreign keys. Foreign keys in this table reference both the Order_T and Product_T tables.

Well-Structured Relations

Well-structured relation
A relation that contains minimal redundancy and allows users to insert, modify, and delete the rows in a table without errors or inconsistencies.

To prepare for our discussion of normalization, we need to address the following question: What constitutes a well-structured relation? Intuitively, a **well-structured relation** contains minimal redundancy and allows users to insert, modify, and delete the rows in a table without errors or inconsistencies. EMPLOYEE1 (Figure 4-1) is such a relation. Each row of the table contains data describing one employee, and any modification to an employee's data (such as a change in salary) is confined to one row of the table. In contrast, EMPLOYEE2 (Figure 4-2b) is not a well-structured relation. If you examine the sample data in the table, you will notice considerable redundancy. For example, values for EmpID, Name, DeptName, and Salary appear in two separate rows for employees 100, 110, and 150. Consequently, if the salary for employee 100 changes, we must record this fact in two rows (or more, for some employees).

Anomaly
An error or inconsistency that may result when a user attempts to update a table that contains redundant data. The three types of anomalies are insertion, deletion, and modification anomalies.

Redundancies in a table may result in errors or inconsistencies (called **anomalies**) when a user attempts to update the data in the table. We are typically concerned about three types of anomalies:

1. ***Insertion anomaly*** Suppose that we need to add a new employee to EMPLOYEE2. The primary key for this relation is the combination of EmpID and CourseTitle (as noted earlier). Therefore, to insert a new row, the user must supply values for both EmpID and CourseTitle (because primary key values cannot be null or nonexistent). This is an anomaly because the user should be able to enter employee data without supplying course data.
2. ***Deletion anomaly*** Suppose that the data for employee number 140 are deleted from the table. This will result in losing the information that this employee

FIGURE 4-7 EMP COURSE

EmpID	CourseTitle	DateCompleted
100	SPSS	6/19/201X
100	Surveys	10/7/201X
140	Tax Acc	12/8/201X
110	Visual Basic	1/12/201X
110	C++	4/22/201X
150	SPSS	6/19/201X
150	Java	8/12/201X

completed a course (Tax Acc) on 12/8/201X. In fact, it results in losing the information that this course had an offering that completed on that date.

3. ***Modification anomaly*** Suppose that employee number 100 gets a salary increase. We must record the increase in each of the rows for that employee (two occurrences in Figure 4-2); otherwise, the data will be inconsistent.

These anomalies indicate that EMPLOYEE2 is not a well-structured relation. The problem with this relation is that it contains data about two entities: EMPLOYEE and COURSE. We will use normalization theory (described later in this chapter) to divide EMPLOYEE2 into two relations. One of the resulting relations is EMPLOYEE1 (Figure 4-1). The other we will call EMP COURSE, which appears with sample data in Figure 4-7. The primary key of this relation is the combination of EmpID and CourseTitle, and we underline these attribute names in Figure 4-7 to highlight this fact. Examine Figure 4-7 to verify that EMP COURSE is free of the types of anomalies described previously and is therefore well structured.

TRANSFORMING EER DIAGRAMS INTO RELATIONS

During logical design, you transform the E-R (and EER) diagrams that were developed during conceptual design into relational database schemas. The inputs to this process are the entity-relationship (and enhanced E-R) diagrams that you studied in Chapters 2 and 3. The outputs are the relational schemas described in the first two sections of this chapter.

Transforming (or mapping) EER diagrams into relations is a relatively straightforward process with a well-defined set of rules. In fact, many CASE tools can automatically perform many of the conversion steps. However, it is important that you understand the steps in this process for four reasons:

1. CASE tools often cannot model more complex data relationships such as ternary relationships and supertype/subtype relationships. In these situations, you may have to perform the steps manually.
2. There are sometimes legitimate alternatives where you will need to choose a particular solution.
3. You must be prepared to perform a quality check on the results obtained with a CASE tool.
4. Understanding the transformation process helps you understand why conceptual data modeling (modeling the real-world domain) is truly a different activity from representing the results of the conceptual data modeling process in a form that can be implemented using a DBMS.

In the following discussion, we illustrate the steps in the transformation with examples taken from Chapters 2 and 3. It will help for you to recall that we discussed three types of entities in those chapters:

1. ***Regular entities*** are entities that have an independent existence and generally represent real-world objects, such as persons and products. Regular entity types are represented by rectangles with a single line.

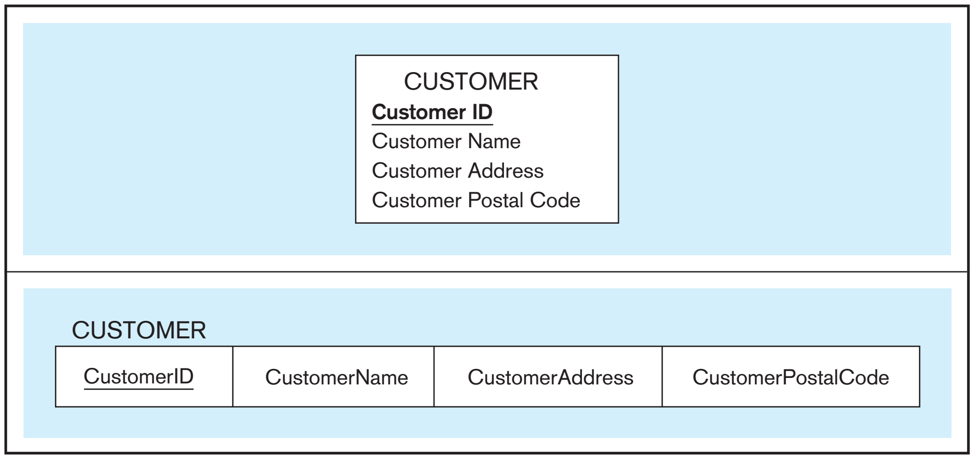


FIGURE 4-8 Mapping the regular entity CUSTOMER
(a) CUSTOMER entity type

(b) CUSTOMER relation

2. ***Weak entities*** are entities that cannot exist except with an identifying relationship with an owner (regular) entity type. Weak entities are identified by a rectangle with a double line.
3. ***Associative entities*** (also called gerunds) are formed from many-to-many relationships between other entity types. Associative entities are represented by a rectangle with rounded corners.

Step 1: Map Regular Entities

Each regular entity type in an E-R diagram is transformed into a relation. The name given to the relation is generally the same as the entity type. Each simple attribute of the entity type becomes an attribute of the relation. The identifier of the entity type becomes the primary key of the corresponding relation. You should check to make sure that this primary key satisfies the desirable properties of identifiers outlined in Chapter 2.

Figure 4-8a shows a representation of the CUSTOMER entity type for Pine Valley Furniture Company from Chapter 2 (see Figure 2-22). The corresponding CUSTOMER relation is shown in graphical form in Figure 4-8b. In this figure and those that follow in this section, we show only a few key attributes for each relation to simplify the figures.

COMPOSITE ATTRIBUTES When a regular entity type has a composite attribute, only the simple components of the composite attribute are included in the new relation as its attributes. Figure 4-9 shows a variation on the example in Figure 4-8, where Customer Address is represented as a composite attribute with components Street, City, and State (see Figure 4-9a). This entity is mapped to the CUSTOMER relation, which contains the simple address attributes, as shown in Figure 4-9b. Although Customer Name is modeled

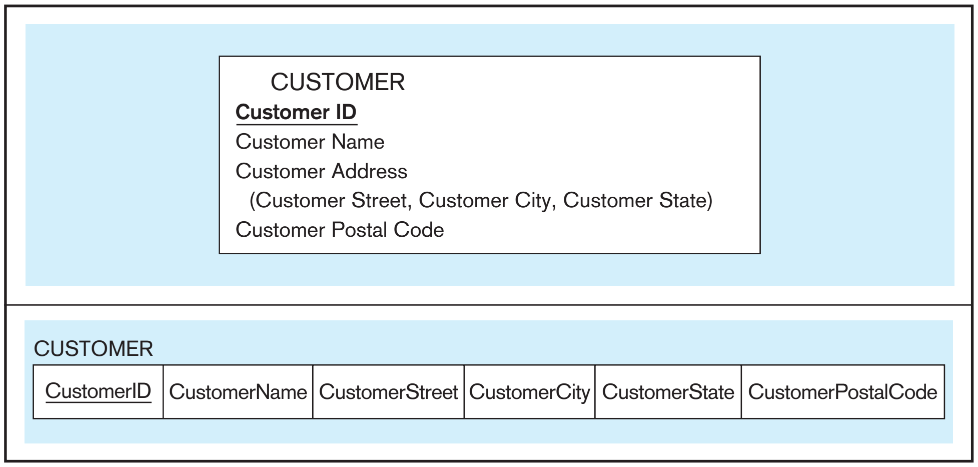


FIGURE 4-9 Mapping a composite attribute
(a) CUSTOMER entity type with composite attribute

(b) CUSTOMER relation with address detail

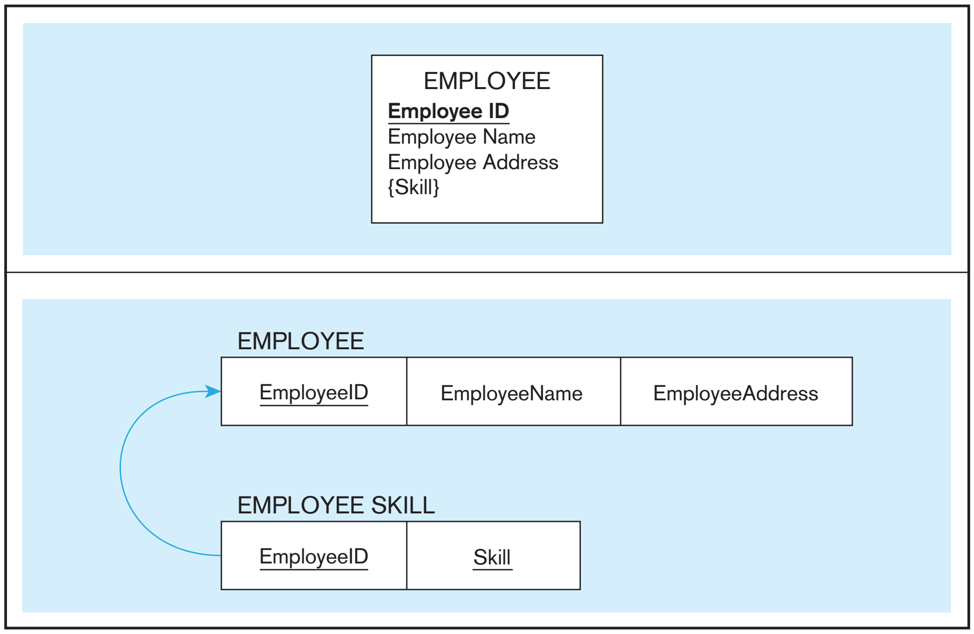


FIGURE 4-10 Mapping an entity with a multivalued attribute
(a) EMPLOYEE entity type with multivalued attribute

(b) Mapping a multivalued attribute

as a simple attribute in Figure 4-9a, you are aware that it instead could have been modeled as a composite attribute with components Last Name, First Name, and Middle Initial. In designing the CUSTOMER relation (Figure 4-9b), you may choose to use these simple attributes instead of CustomerName. Compared to composite attributes, simple attributes improve data accessibility and facilitate maintaining data quality.

MULTIVALUED ATTRIBUTES When the regular entity type contains a multivalued attribute, two new relations (rather than one) are created. The first relation contains all of the attributes of the entity type except the multivalued attribute. The second relation contains two attributes that form the primary key of the second relation. The first of these attributes is the primary key from the first relation, which becomes a foreign key in the second relation. The second is the multivalued attribute. The name of the second relation should capture the meaning of the multivalued attribute.

An example of this procedure is shown in Figure 4-10. This is the EMPLOYEE entity type for Pine Valley Furniture Company. As shown in Figure 4-10a, EMPLOYEE has Skill as a multivalued attribute. Figure 4-10b shows the two relations that are created. The first (called EMPLOYEE) has the primary key EmployeeID. The second relation (called EMPLOYEE SKILL) has the two attributes, EmployeeID and Skill, which form the primary key. The relationship between foreign and primary keys is indicated by the arrow in the figure.

The relation EMPLOYEE SKILL contains no nonkey attributes (also called *descriptors*). Each row simply records the fact that a particular employee possesses a particular skill. This provides an opportunity for you to suggest to users that new attributes can be added to this relation. For example, the attributes YearsExperience and/or CertificationDate might be appropriate new values to add to this relation. See Figure 2-15b for another variation on employee skills.

If an entity type contains multiple multivalued attributes, each of them will be converted to a separate relation.

Step 2: Map Weak Entities

Recall that a weak entity type does not have an independent existence but exists only through an identifying relationship with another entity type called the *owner*. A weak entity type does not have a complete identifier but must have an attribute called a partial identifier that permits distinguishing the various occurrences of the weak entity for each owner entity instance.

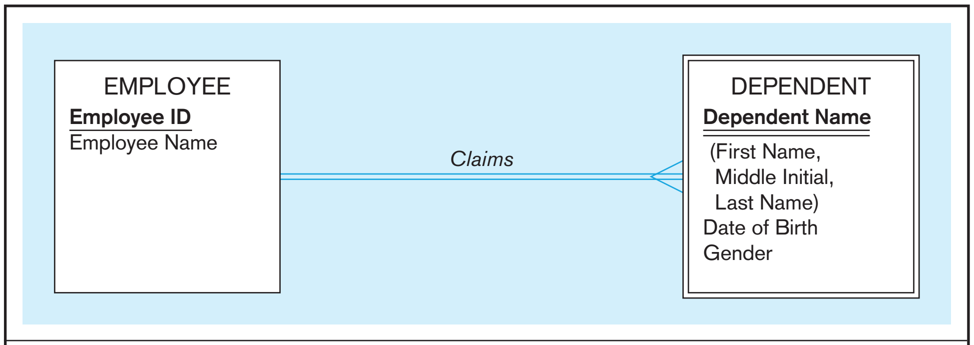
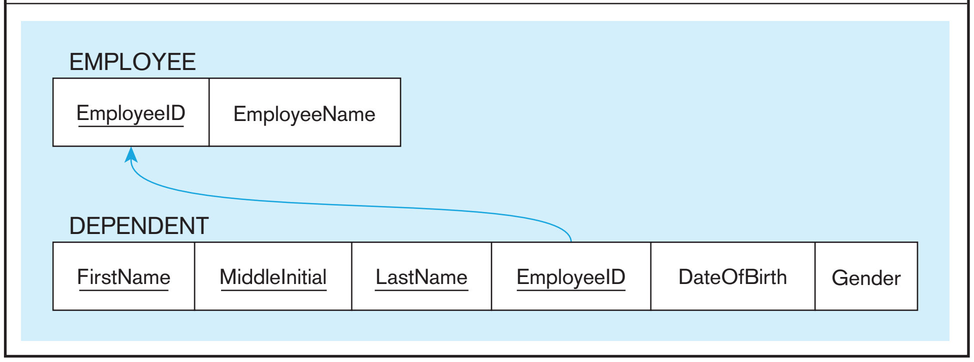


FIGURE 4-11 Example of mapping a weak entity
(a) Weak entity DEPENDENT



(b) Relations resulting from weak entity

The following procedure assumes that you have already created a relation corresponding to the identifying entity type during Step 1. If you have not, you should create that relation now, using the process described in Step 1.

For each weak entity type, create a new relation and include all of the simple attributes (or simple components of composite attributes) as attributes of this relation. Then include the primary key of the *identifying* relation as a foreign key attribute in this new relation. The primary key of the new relation is the combination of this primary key of the identifying and the partial identifier of the weak entity type.

An example of this process is shown in Figure 4-11. Figure 4-11a shows the weak entity type DEPENDENT and its identifying entity type EMPLOYEE, linked by the identifying relationship Claims (see Figure 2-5). Notice that the attribute Dependent Name, which is the partial identifier for this relation, is a composite attribute with components First Name, Middle Initial, and Last Name. Thus, we assume that, *for a given employee*, these items will uniquely identify a dependent (a notable exception being the case of prizefighter George Foreman, who has named all his sons after himself).

Figure 4-11b shows the two relations that result from mapping this E-R segment. The primary key of DEPENDENT consists of four attributes: EmployeeID, FirstName, MiddleInitial, and LastName. DateOfBirth and Gender are the nonkey attributes. The foreign key relationship with its primary key is indicated by the arrow in the figure.

In practice, an alternative approach is often used to simplify the primary key of the DEPENDENT relation: Create a new attribute (called Dependent#), which will be used as a **surrogate primary key** in Figure 4-11b. With this approach, the relation DEPENDENT has the following attributes:

Surrogate primary key
A serial number or other system-assigned primary key for a relation.

DEPENDENT(Dependent#, EmployeeID, FirstName, MiddleInitial, LastName, DateOfBirth, Gender)

Dependent# is simply a serial number that is assigned to each dependent of an employee. Notice that this solution will ensure unique identification for each dependent (even for those of George Foreman!).

WHEN TO CREATE A SURROGATE KEY A surrogate key is usually created to simplify the key structures. According to Hoberman (2006), a surrogate key should be created when any of the following conditions hold:

- There is a composite primary key, as in the case of the DEPENDENT relation shown previously with the four component primary key.
- The natural primary key (i.e., the key used in the organization and identified in conceptual data modeling as the identifier) is inefficient (e.g., it may be very long and hence costly for database software to handle if it is used as a foreign key that references other tables).
- The natural primary key is recycled (i.e., the key is reused or repeated periodically, so it may not actually be unique over time); a more general statement of this condition is when the natural primary key cannot, in fact, be guaranteed to be unique over time (e.g., there could be duplicates, such as with names or titles).

Whenever a surrogate key is created, the natural key is always kept as nonkey data in the same relation because the natural key has organizational meaning. In fact, surrogate keys mean nothing to users, so they are usually never displayed; rather, the natural keys are shown to the user as the primary keys and used as identifiers in searches.

Step 3: Map Binary Relationships

The procedure for representing relationships depends on both the degree of the relationships (unary, binary, or ternary) and the cardinalities of the relationships. We describe and illustrate the important cases in the following discussion.

MAP BINARY ONE-TO-MANY RELATIONSHIPS For each binary 1:*M* relationship, first create a relation for each of the two entity types participating in the relationship, using the procedure described in Step 1. Next, include the primary key attribute (or attributes) of the entity on the one-side of the relationship as a foreign key in the relation that is on the many-side of the relationship. (A mnemonic you can use to remember this rule is this: The primary key migrates to the many side.)

To illustrate this simple process, we use the Submits relationship between customers and orders for Pine Valley Furniture Company (see Figure 2-22). This 1:*M* relationship is illustrated in Figure 4-12a. (Again, we show only a few attributes for simplicity.) Figure 4-12b shows the result of applying this rule to map the entity types with the 1:*M* relationship. The primary key CustomerID of CUSTOMER (the one side) is included as a foreign key in ORDER (the many side). The foreign key relationship is indicated with an arrow.

PINE VALLEY FURNITURE

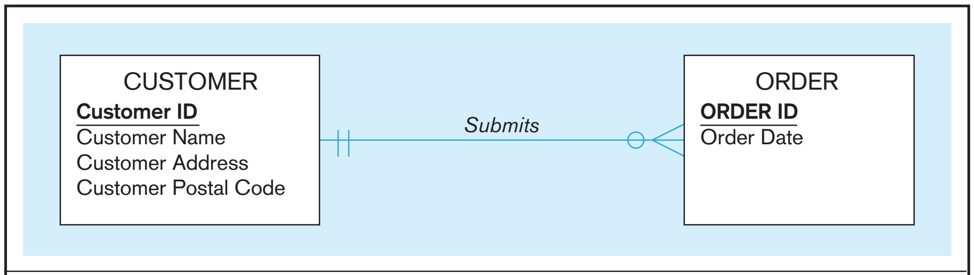
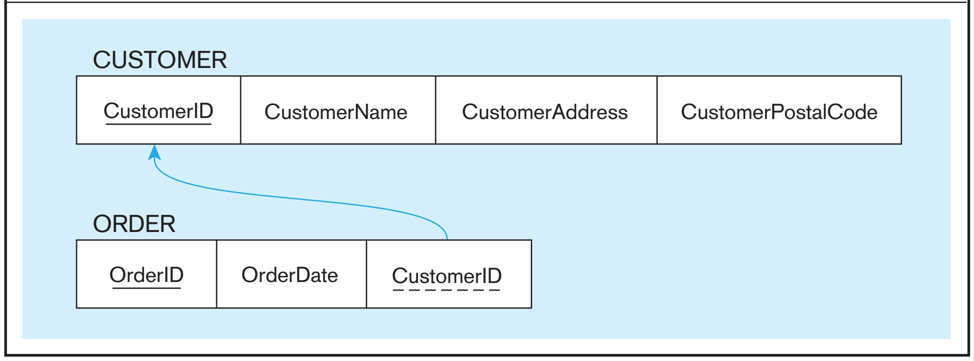


FIGURE 4-12 Example of mapping a 1:*M* relationship (a) Relationship between customers and orders



(b) Mapping the relationship

MAP BINARY MANY-TO-MANY RELATIONSHIPS Suppose that there is a binary many-to-many (*M:N*) relationship between two entity types, A and B. For such a relationship, create a new relation, C. Include as foreign key attributes in C the primary key for each of the two participating entity types. These attributes together become the primary key of C. Any nonkey attributes that are associated with the *M:N* relationship are included with the relation C.

Figure 4-13 shows an example of applying this rule. Figure 4-13a shows the Completes relationship between the entity types EMPLOYEE and COURSE from Figure 2-11a. Figure 4-13b shows the three relations (EMPLOYEE, COURSE, and CERTIFICATE) that are formed from the entity types and the Completes relationship. If Completes had been represented as an associative entity, as is done in Figure 2-11b, a similar result would occur, but we will deal with associative entities in a subsequent section. In the case of an *M:N* relationship, first, a relation is created for each of the two regular entity types EMPLOYEE and COURSE. Then a new relation (named CERTIFICATE in Figure 4-13b) is created for the Completes relationship. The primary key of CERTIFICATE is the combination of EmployeeID and CourseID, which are the respective primary keys of EMPLOYEE and COURSE. As indicated in the diagram, these attributes are foreign keys that "point to" the respective primary keys. The nonkey attribute DateCompleted also appears in CERTIFICATE. Although not shown here, it might be wise to create a surrogate primary key for the CERTIFICATE relation.

MAP BINARY ONE-TO-ONE RELATIONSHIPS Binary one-to-one relationships can be viewed as a special case of one-to-many relationships. The process of mapping such a relationship to relations requires two steps. First, two relations are created, one for each of the participating entity types. Second, the primary key of one of the relations is included as a foreign key in the other relation.

In a 1:1 relationship, the association in one direction is nearly always an optional one, whereas the association in the other direction is mandatory one. (You can review the notation for these terms in Figure 2-1.) You should include in the relation on the

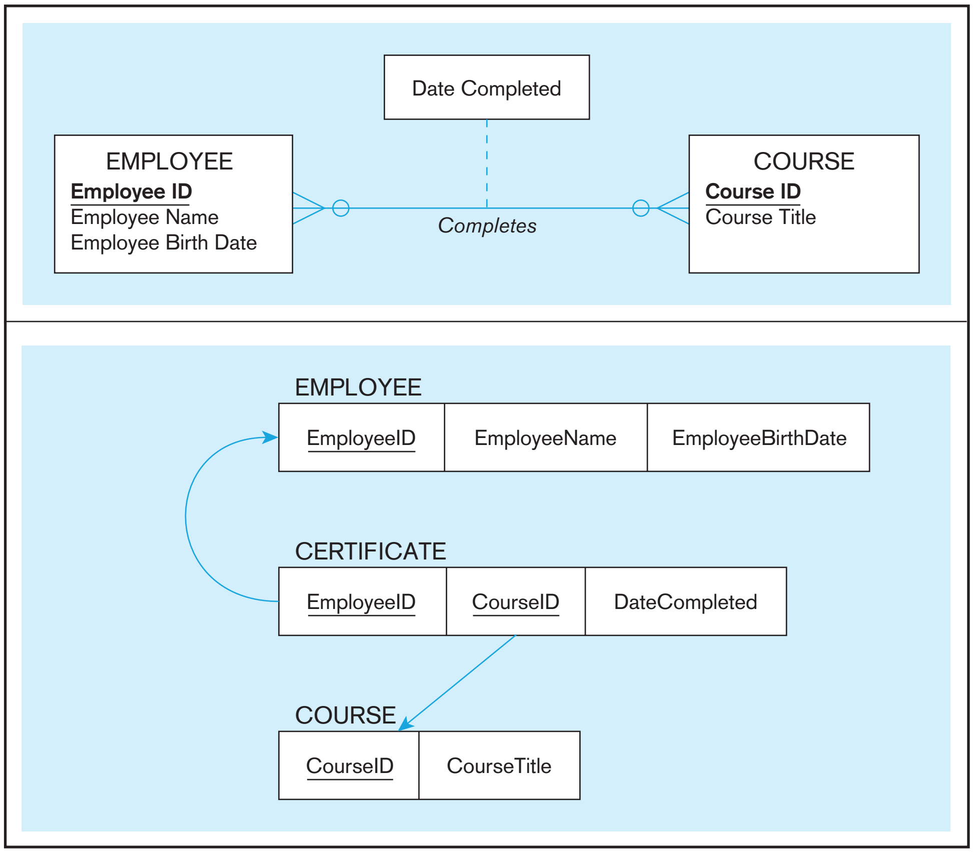


FIGURE 4-13 **Example of mapping a *M:N* relationship (a) Completes relationship (*M:N*)**

(b) Three resulting relations

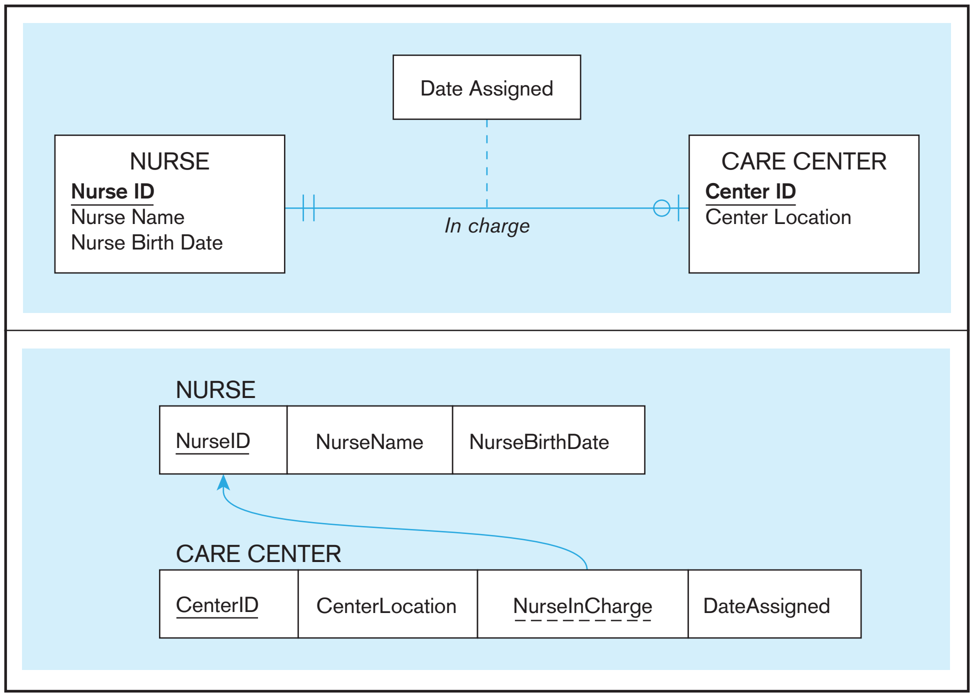


FIGURE 4-14 Example of mapping a binary 1:1 relationship
(a) In Charge relationship (binary 1:1)

(b) Resulting relations

optional side of the relationship the foreign key of the entity type that has the mandatory participation in the 1:1 relationship. This approach will prevent the need to store null values in the foreign key attribute. Any attributes associated with the relationship itself are also included in the same relation as the foreign key.

An example of applying this procedure is shown in Figure 4-14. Figure 4-14a shows a binary 1:1 relationship between the entity types NURSE and CARE CENTER. Each care center must have a nurse who is in charge of that center. Thus, the association from CARE CENTER to NURSE is a mandatory one, whereas the association from NURSE to CARE CENTER is an optional one (since any nurse may or may not be in charge of a care center). The attribute Date Assigned is attached to the In Charge relationship.

The result of mapping this relationship to a set of relations is shown in Figure 4-14b. The two relations NURSE and CARE CENTER are created from the two entity types. Because CARE CENTER is the optional participant, the foreign key is placed in this relation. In this case, the foreign key is NurseInCharge. It has the same domain as NurseID, and the relationship with the primary key is shown in the figure. The attribute DateAssigned is also located in CARE CENTER and would not be allowed to be null.

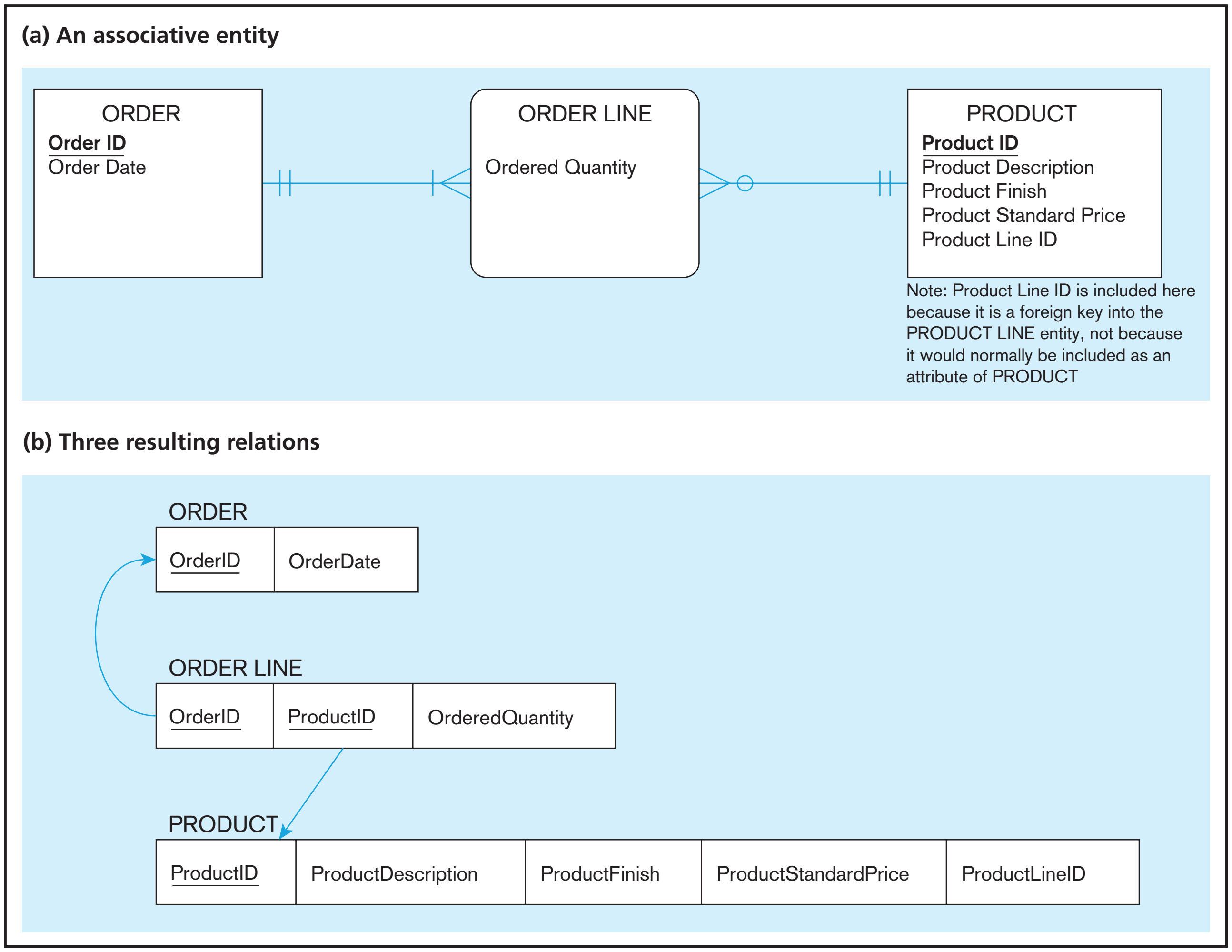
Step 4: Map Associative Entities

As explained in Chapter 2, when a data modeler encounters a many-to-many relationship, he or she may choose to model that relationship as an associative entity in the E-R diagram. This approach is most appropriate when the end user can best visualize the relationship as an entity type rather than as an *M:N* relationship. Mapping the associative entity involves essentially the same steps as mapping an *M:N* relationship, as described in Step 3.

The first step is to create three relations: one for each of the two participating entity types and a third for the associative entity. We refer to the relation formed from the associative entity as the *associative relation*. The second step then depends on whether on the E-R diagram an identifier was assigned to the associative entity.

IDENTIFIER NOT ASSIGNED If an identifier was not assigned, the default primary key for the associative relation consists of the two primary key attributes from the other two relations. These attributes are then foreign keys that reference the other two relations.

FIGURE 4-15 Example of mapping an associative entity



PINE VALLEY FURNITURE

An example of this case is shown in Figure 4-15. Figure 4-15a shows the associative entity ORDER LINE that links the ORDER and PRODUCT entity types at Pine Valley Furniture Company (see Figure 2-22). Figure 4-15b shows the three relations that result from this mapping. Note the similarity of this example to that of an *M:N* relationship shown in Figure 4-13.

IDENTIFIER ASSIGNED Sometimes a data modeler will assign a single-attribute identifier to the associative entity type on the E-R diagram. There are two reasons that may have motivated the data modeler to assign a single-attribute key during conceptual data modeling:

1. The associative entity type has a natural single-attribute identifier that is familiar to end users.
2. The default identifier (consisting of the identifiers for each of the participating entity types) may not uniquely identify instances of the associative entity.

These motivations are in addition to the reasons mentioned earlier in this chapter to create a surrogate primary key.

The process for mapping the associative entity in this case is now modified as follows. As before, a new (associative) relation is created to represent the associative entity. However, the primary key for this relation is the identifier assigned on the E-R diagram (rather than the default key). The primary keys for the two participating entity types are then included as foreign keys in the associative relation.

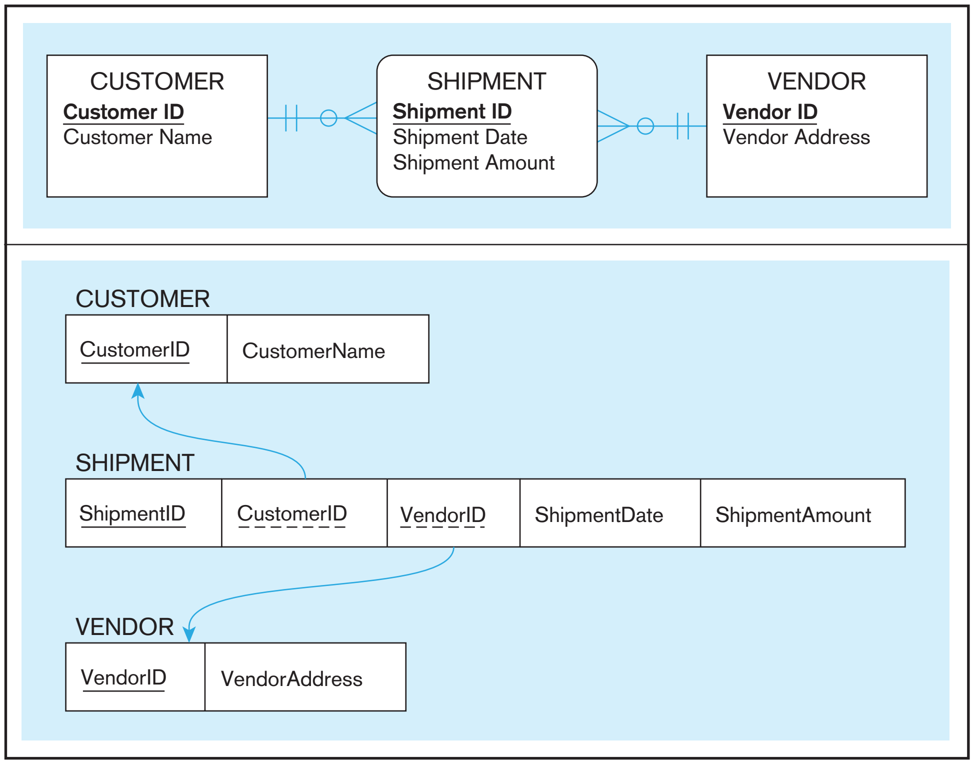


FIGURE 4-16 Example of mapping an associative entity with an identifier
(a) SHIPMENT associative entity

(b) Three resulting relations

An example of this process is shown in Figure 4-16. Figure 4-16a shows the associative entity type SHIPMENT that links the CUSTOMER and VENDOR entity types. Shipment ID has been chosen as the identifier for SHIPMENT for two reasons:

1. Shipment ID is a natural identifier for this entity that is very familiar to end users.
2. The default identifier consisting of the combination of Customer ID and Vendor ID does not uniquely identify the instances of SHIPMENT. In fact, a given vendor typically makes many shipments to a given customer. Even including the attribute Date does not guarantee uniqueness, since there may be more than one shipment by a particular vendor on a given date. The surrogate key ShipmentID will, however, uniquely identify each shipment.

Two nonkey attributes associated with SHIPMENT are Shipment Date and Shipment Amount.

The result of mapping this entity to a set of relations is shown in Figure 4-16b. The new associative relation is named SHIPMENT. The primary key is ShipmentID. CustomerID and VendorID are included as foreign keys in this relation, and ShipmentDate and ShipmentAmount are nonkey attributes.

Step 5: Map Unary Relationships

In Chapter 2, we defined a unary relationship as a relationship between the instances of a single entity type. Unary relationships are also called *recursive relationships*. The two most important cases of unary relationships are one-to-many and many-to-many relationships. We discuss these two cases separately because the approach to mapping is somewhat different for the two types.

UNARY ONE-TO-MANY RELATIONSHIPS The entity type in the unary relationship is mapped to a relation using the procedure described in Step 1. Then a foreign key attribute is added to the same relation; this attribute references the primary key values in the same relation. (This foreign key must have the same domain as the primary key.) This type of a foreign key is called a **recursive foreign key**.

Recursive foreign key
A foreign key in a relation that references the primary key values of the same relation.

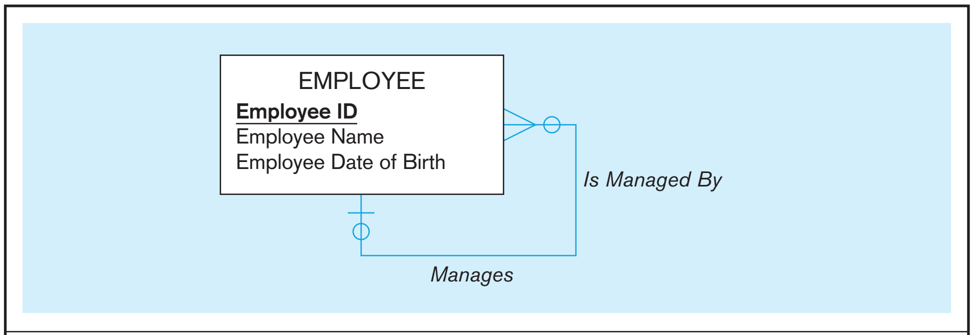
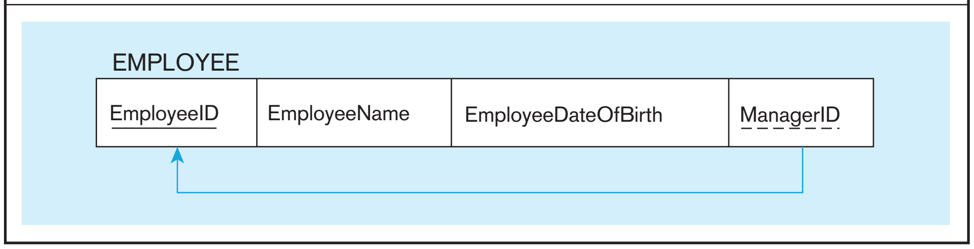


FIGURE 4-17 Mapping a unary 1:*N* relationship
(a) EMPLOYEE entity with unary relationship



(b) EMPLOYEE relation with recursive foreign key

Figure 4-17a shows a unary one-to-many relationship named Manages that associates each employee of an organization with another employee who is his or her manager. Each employee may have one manager; a given employee may manage zero to many employees.

The EMPLOYEE relation that results from mapping this entity and relationship is shown in Figure 4-17b. The (recursive) foreign key in the relation is named ManagerID. This attribute has the same domain as the primary key EmployeeID. Each row of this relation stores the following data for a given employee: EmployeeID, EmployeeName, EmployeeDateOfBirth, and ManagerID (i.e., EmployeeID for this employee's manager). Notice that because it is a foreign key, ManagerID references EmployeeID.

UNARY MANY-TO-MANY RELATIONSHIPS With this type of relationship, two relations are created: one to represent the entity type in the relationship and an associative relation to represent the *M:N* relationship itself. The primary key of the associative relation consists of two attributes. These attributes (which need not have the same name) both take their values from the primary key of the other relation. Any nonkey attribute of the relationship is included in the associative relation.

An example of mapping a unary *M:N* relationship is shown in Figure 4-18. Figure 4-18a shows a bill-of-materials relationship among items that are assembled from other items or components. (This structure was described in Chapter 2, and an example appears in Figure 2-13.) The relationship (called Contains) is *M:N* because a given item can contain numerous component items, and, conversely, an item can be used as a component in numerous other items.

The relations that result from mapping this entity and its relationship are shown in Figure 4-18b. The ITEM relation is mapped directly from the same entity type. COMPONENT is an associative relation whose primary key consists of two attributes that are arbitrarily named ItemNo and ComponentNo. The attribute Quantity is a nonkey attribute of this relation that, for a given item, records the quantity of a particular component item used in that item. Notice that both ItemNo and ComponentNo reference the primary key (ItemNo) of the ITEM relation.

We can easily query these relations to determine, for example, the components of a given item. The following SQL query will list the immediate components (and their quantity) for item number 100:

```
SELECT ComponentNo, Quantity
FROM Component_T
WHERE ItemNo = 100;
```

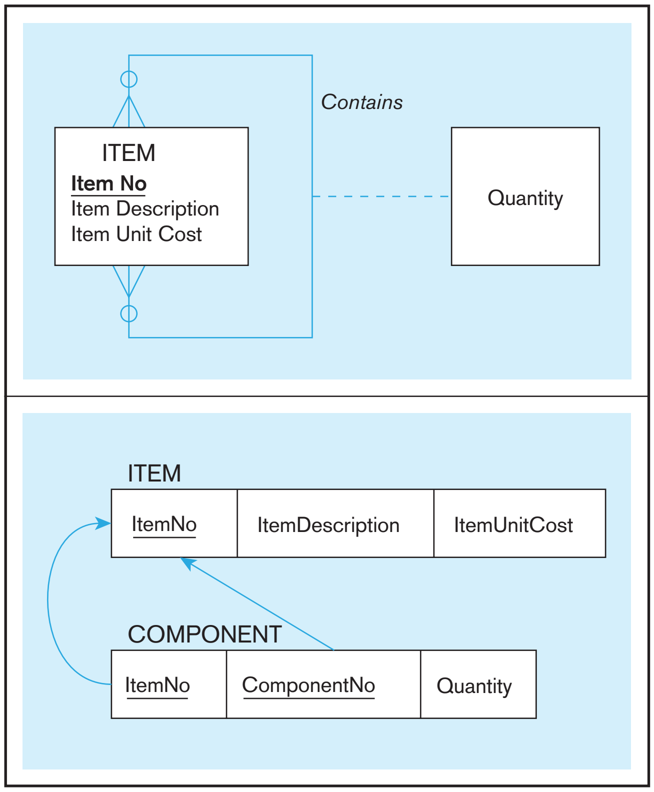


FIGURE 4-18 Mapping a unary *M:N* relationship
(a) Bill-of-materials relationship Contains (*M:N*)
(b) ITEM and COMPONENT relations

Step 6: Map Ternary (and *n*-ary) Relationships

Recall from Chapter 2 that a ternary relationship is a relationship among three entity types. In that chapter, we recommended that you convert a ternary relationship to an associative entity to represent participation constraints more accurately.

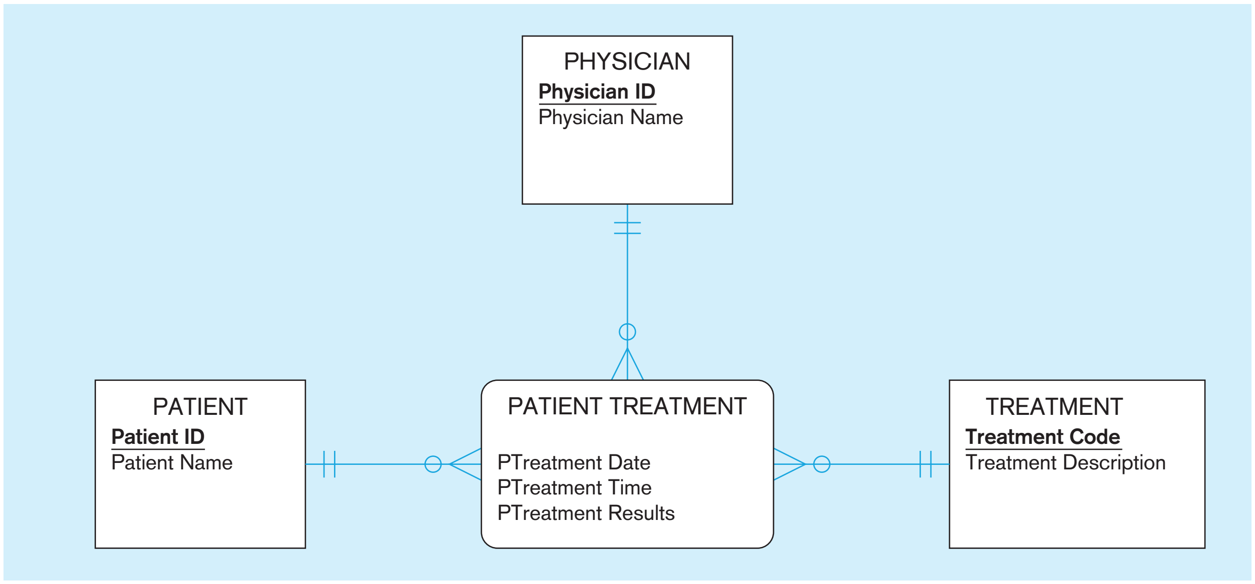
To map an associative entity type that links three regular entity types, we create a new associative relation. The default primary key of this relation consists of the three primary key attributes for the participating entity types. (In some cases, additional attributes are required to form a unique primary key.) These attributes then act in the role of foreign keys that reference the individual primary keys of the participating entity types. Any attributes of the associative entity type become attributes of the new relation.

An example of mapping a ternary relationship (represented as an associative entity type) is shown in Figure 4-19. Figure 4-19a is an E-R segment (or view) that represents a *patient* receiving a *treatment* from a *physician*. The associative entity type PATIENT TREATMENT has the attributes PTreatment Date, PTreatment Time, and PTreatment Results; values are recorded for these attributes for each instance of PATIENT TREATMENT.

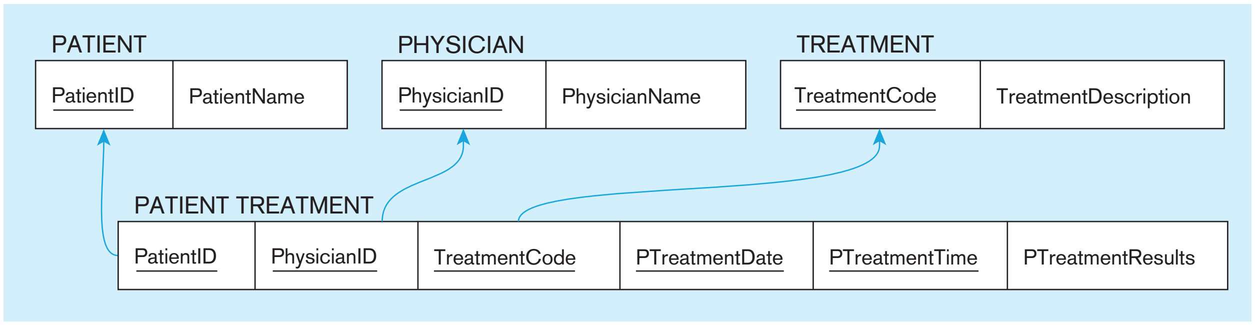
The result of mapping this view is shown in Figure 4-19b. The primary key attributes PatientID, PhysicianID, and TreatmentCode become foreign keys in PATIENT TREATMENT. The foreign key into TREATMENT is called PTreatmentCode in PATIENT TREATMENT. We are using this column name to illustrate that the foreign key name does not have to be the same as the name of the primary key to which it refers, as long as the values come from the same domain. These three attributes are components of the primary key of PATIENT TREATMENT. However, they do not uniquely identify a given treatment, because a patient may receive the same treatment from the same physician on more than one occasion. Does including the attribute Date as part of the primary key (along with the other three attributes) result in a primary key? This would be so if a given patient receives only one treatment from a particular physician on a given date. However, this is not likely to be the case. For example, a patient may receive a treatment in the morning, then the same treatment again in the afternoon. To resolve this issue, we include PTreatmentDate and PTreatmentTime as part of the primary key. Therefore, the primary key of PATIENT TREATMENT consists of the five attributes

FIGURE 4-19 **Mapping a ternary relationship**

(a) PATIENT TREATMENT ternary relationship with associative entity



(b) Mapping the ternary relationship PATIENT TREATMENT



shown in Figure 4-19b: PatientID, PhysicianID, TreatmentCode, PTreatmentDate, and PTreatmentTime. The only nonkey attribute in the relation is PTreatmentResults.

Although this primary key is technically correct, it is complex and therefore difficult to manage and prone to errors. A better approach is to introduce a surrogate key, such as Treatment#, that is a serial number that uniquely identifies each treatment. In this case, each of the former primary key attributes except for PTreatmentDate and PTreatmentTime becomes a foreign key in the PATIENT TREATMENT relation. Another similar approach is to use an enterprise key, as described at the end of this chapter.

Step 7: Map Supertype/Subtype Relationships

The relational data model does not yet directly support supertype/subtype relationships. Fortunately, there are various strategies that database designers can use to represent these relationships with the relational data model (Chouinard, 1989). For our purposes, we use the following strategy, which is the one most commonly employed:

1. Create a separate relation for the supertype and for each of its subtypes.
2. Assign to the relation created for the supertype the attributes that are common to all members of the supertype, including the primary key.
3. Assign to the relation for each subtype the primary key of the supertype and only those attributes that are unique to that subtype.
4. Assign one (or more) attributes of the supertype to function as the subtype discriminator. (The role of the subtype discriminator was discussed in Chapter 3.)

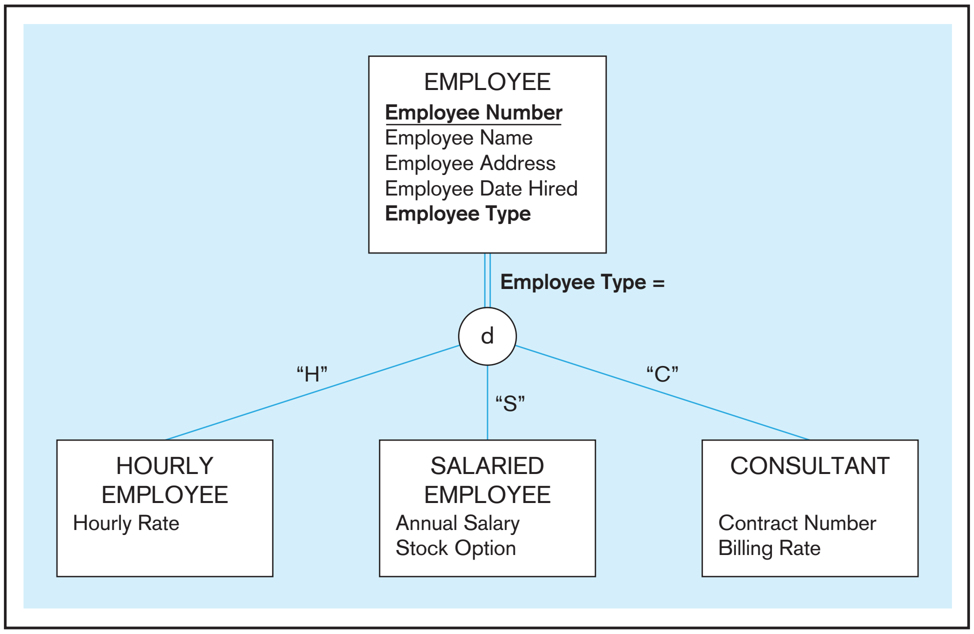


FIGURE 4-20 Supertype/subtype relationships

An example of applying this procedure is shown in Figures 4-20 and 4-21. Figure 4-20 shows the supertype EMPLOYEE with subtypes HOURLY EMPLOYEE, SALARIED EMPLOYEE, and CONSULTANT. (This example is described in Chapter 3, and Figure 4-20 is a repeat of Figure 3-8.) The primary key of EMPLOYEE is Employee Number, and the attribute Employee Type is the subtype discriminator.

The result of mapping this diagram to relations using these rules is shown in Figure 4-21. There is one relation for the supertype (EMPLOYEE) and one for each of the three subtypes. The primary key for each of the four relations is EmployeeNumber. A prefix is used to distinguish the name of the primary key for each subtype. For example, SEmployeeNumber is the name for the primary key of the relation SALARIED EMPLOYEE. Each of these attributes is a foreign key that references the supertype primary key, as indicated by the arrows in the diagram. Each subtype relation contains only those attributes unique to the subtype.

For each subtype, a relation can be produced that contains all of the attributes of that subtype (both specific and inherited) by using an SQL command that joins the subtype with its supertype. For example, suppose that we want to display a table that

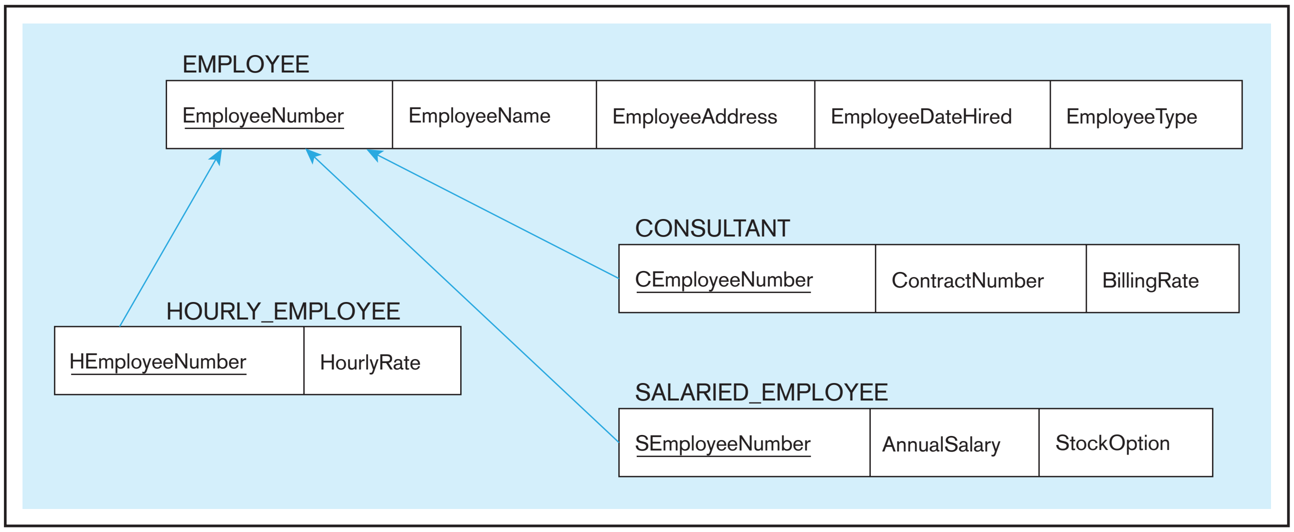


FIGURE 4-21 Mapping supertype/subtype relationships to relations

contains all of the attributes for SALARIED EMPLOYEE. The following command is used:

```
SELECT *
FROM Employee_T, SalariedEmployee_T
WHERE EmployeeNumber = SEmployeeNumber;
```

Summary of EER-to-Relational Transformations

The steps provide a comprehensive explanation of how each element of an EER diagram is transformed into parts of a relational data model. Table 4-2 is a quick reference to these steps and the associated figures that illustrate each type of transformation.

INTRODUCTION TO NORMALIZATION

Following the steps outlined previously for transforming EER diagrams into relations often results in well-structured relations. However, there is no guarantee that all anomalies are removed by following these steps. Normalization is a formal process for deciding which attributes should be grouped together in a relation so that all anomalies are removed. For example, we used the principles of normalization to convert the EMPLOYEE2 table (with its redundancy) to EMPLOYEE1 (Figure 4-1) and EMP COURSE

TABLE 4-2 Summary of EER to Relational Transformations

EER Structure	Relational Representation (Sample Figure)
Regular entity	Create a relation with primary key and nonkey attributes (Figure 4-8)
Composite attribute	Each component of a composite attribute becomes a separate attribute in the target relation (Figure 4-9)
Multivalued attribute	Create a separate relation for multivalued attribute with composite primary key, including the primary key of the entity (Figure 4-10)
Weak entity	Create a relation with a composite primary key (which includes the primary key of the entity on which this entity depends) and nonkey attributes (Figure 4-11)
Binary or unary 1:*N* relationship	Place the primary key of the entity on the one side of the relationship as a foreign key in the relation for the entity on the many side (Figure 4-12; Figure 4-17 for unary relationship)
Binary or unary *M:N* relationship or associative entity without its own key	Create a relation with a composite primary key using the primary keys of the related entities plus any nonkey attributes of the relationship or associative entity (Figure 4-13; Figure 4-15 for associative entity; Figure 4-18 for unary relationship)
Binary or unary 1:1 relationship	Place the primary key of either entity in the relation for the other entity; if one side of the relationship is optional, place the foreign key of the entity on the mandatory side in the relation for the entity on the optional side (Figure 4-14)
Binary or unary *M:N* relationship or associative entity with its own key	Create a relation with the primary key associated with the associative entity plus any nonkey attributes of the associative entity and the primary keys of the related entities as foreign keys (Figure 4-16)
Ternary and *n*-ary relationships	Same as binary *M:N* relationships above; without its own key, include as part of primary key of relation for the relationship or associative entity the primary keys from all related entities; with its own surrogate key, the primary keys of the associated entities are included as foreign keys in the relation for the relationship or associative entity (Figure 4-19)
Supertype/subtype relationship	Create a relation for the superclass, which contains the primary and all nonkey attributes in common with all subclasses, plus create a separate relation for each subclass with the same primary key (with the same or local name) but with only the nonkey attributes related to that subclass (Figure 4-20 and 4-21)

(Figure 4-7). There are two major occasions during the overall database development process when you can usually benefit from using normalization:

1. ***During logical database design (described in this chapter)*** You should use normalization concepts as a quality check for the relations that are obtained from mapping E-R diagrams.
2. ***When reverse-engineering older systems*** Many of the tables and user views for older systems are redundant and subject to the anomalies we describe in this chapter.

So far we have presented an intuitive discussion of well-structured relations; however, we need formal definitions of such relations, together with a process for designing them. **Normalization** is the process of successively reducing relations with anomalies to produce smaller, well-structured relations. Following are some of the main goals of normalization:

Normalization
The process of decomposing relations with anomalies to produce smaller, well-structured relations.

1. Minimize data redundancy, thereby avoiding anomalies and conserving storage space
2. Simplify the enforcement of referential integrity constraints
3. Make it easier to maintain data (insert, update, and delete)
4. Provide a better design that is an improved representation of the real world and a stronger basis for future growth

Normalization makes no assumptions about how data will be used in displays, queries, or reports. Normalization, based on what we will call *normal forms* and *functional dependencies*, defines rules of the business, not data usage. Further, remember that data are normalized by the end of logical database design. Thus, normalization, as we will see in Chapter 5, places no constraints on how data can or should be physically stored or, therefore, on processing performance. Normalization is a logical data modeling technique used to ensure that data are well structured from an organization-wide view.

Steps in Normalization

Normalization can be accomplished and understood in stages, each of which corresponds to a normal form (see Figure 4-22). A **normal form** is a state of a relation that requires that certain rules regarding relationships between attributes (or functional dependencies) are satisfied. We describe these rules briefly in this section and illustrate them in detail in the following sections:

Normal form
A state of a relation that requires that certain rules regarding relationships between attributes (or functional dependencies) are satisfied.

1. ***First normal form*** Any multivalued attributes (also called *repeating groups*) have been removed, so there is a single value (possibly null) at the intersection of each row and column of the table (as in Figure 4-2b).
2. ***Second normal form*** Any partial functional dependencies have been removed (i.e., nonkey attributes are identified by the whole primary key).
3. ***Third normal form*** Any transitive dependencies have been removed (i.e., nonkey attributes are identified by only the primary key).
4. ***Boyce-Codd normal form*** Any remaining anomalies that result from functional dependencies have been removed (because there was more than one possible primary key for the same nonkeys).
5. ***Fourth normal form*** Any multivalued dependencies have been removed.
6. ***Fifth normal form*** Any remaining anomalies have been removed.

We describe and illustrate the first through the third normal forms in this chapter. The remaining normal forms are described in Appendix B. These other normal forms are in an appendix only to save space in this chapter, not because they are less important. In fact, you can easily continue with Appendix B immediately after the section on the third normal form.

Functional Dependencies and Keys

Up to the Boyce-Codd normal form, normalization is based on the analysis of functional dependencies. A **functional dependency** is a constraint between two attributes or two sets of attributes. For any relation R, attribute B is functionally

Functional dependency
A constraint between two attributes in which the value of one attribute is determined by the value of another attribute.

FIGURE 4-22 **Steps in normalization**

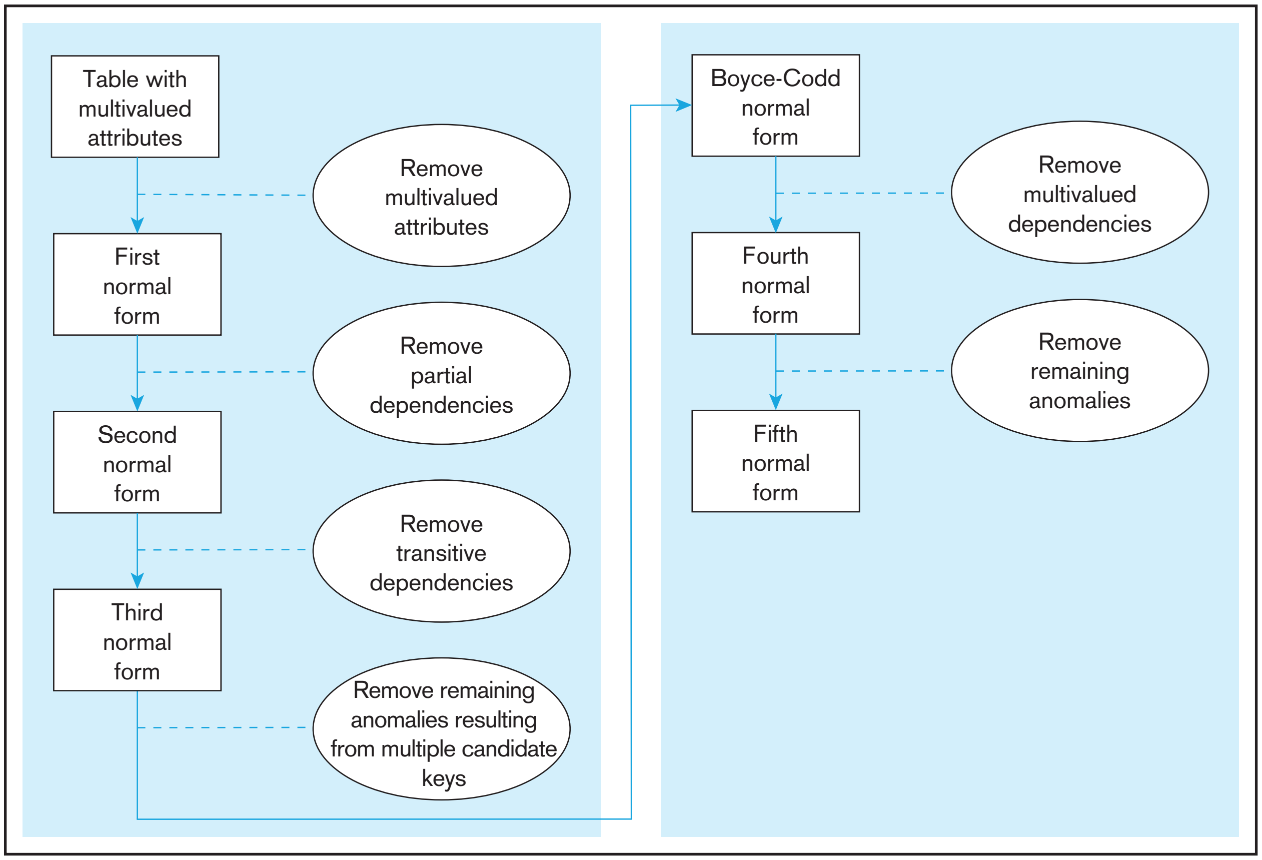


Table with multivalued attributes
Remove multivalued attributes
First normal form
Remove partial dependencies
Second normal form
Remove transitive dependencies
Third normal form
Remove remaining anomalies resulting from multiple candidate keys
Boyce-Codd normal form
Remove multivalued dependencies
Fourth normal form
Remove remaining anomalies
Fifth normal form

dependent on attribute A if, for every valid instance of A, that value of A uniquely determines the value of B (Dutka and Hanson, 1989). The functional dependency of B on A is represented by an arrow, as follows: A → B. A functional dependency is not a mathematical dependency: B cannot be computed from A. Rather, if you know the value of A, there can be only one value for B. An attribute may be functionally dependent on a combination of two (or more) attributes rather than on a single attribute. For example, consider the relation EMP COURSE (EmpID, CourseTitle, DateCompleted) shown in Figure 4-7. We represent the functional dependency in this relation as follows:

EmpID, CourseTitle → DateCompleted

The comma between EmpID and CourseTitle stands for the logical AND operator, because DateCompleted is functionally dependent on EmpID and CourseTitle in combination.

The functional dependency in this statement implies that the date a course is completed is determined by the identity of the employee and the title of the course. Typical examples of functional dependencies are the following:

1. ***SSN → Name, Address, Birthdate*** A person's name, address, and birth date are functionally dependent on that person's Social Security number (in other words, there can be only one Name, one Address, and one Birthdate for each SSN).
2. ***VIN → Make, Model, Color*** The make, model, and color of a vehicle are functionally dependent on the vehicle identification number (as above, there can be only one value of Make, Model, and Color associated with each VIN).
3. ***ISBN → Title, FirstAuthorName, Publisher*** The title of a book, the name of the first author, and the publisher are functionally dependent on the book's international standard book number (ISBN).

DETERMINANTS The attribute on the left side of the arrow in a functional dependency is called a **determinant**. SSN, VIN, and ISBN are determinants (respectively) in the preceding three examples. In the EMP COURSE relation (Figure 4-7), the combination of EmpID and CourseTitle is a determinant.

Determinant
The attribute on the left side of the arrow in a functional dependency.

CANDIDATE KEYS A **candidate key** is an attribute, or combination of attributes, that uniquely identifies a row in a relation. A candidate key must satisfy the following properties (Dutka and Hanson, 1989), which are a subset of the six properties of a relation previously listed:

Candidate key
An attribute, or combination of attributes, that uniquely identifies a row in a relation.

1. ***Unique identification*** For every row, the value of the key must uniquely identify that row. This property implies that each nonkey attribute is functionally dependent on that key.
2. ***Nonredundancy*** No attribute in the key can be deleted without destroying the property of unique identification.

Let's apply the preceding definition to identify candidate keys in two of the relations described in this chapter. The EMPLOYEE1 relation (Figure 4-1) has the following schema: EMPLOYEE1(EmpID, Name, DeptName, Salary). EmpID is the only determinant in this relation. All of the other attributes are functionally dependent on EmpID. Therefore, EmpID is a candidate key and (because there are no other candidate keys) also is the primary key.

We represent the functional dependencies for a relation using the notation shown in Figure 4-23. Figure 4-23a shows the representation for EMPLOYEE1. The horizontal line in the figure portrays the functional dependencies. A vertical line drops from the primary key (EmpID) and connects to this line. Vertical arrows then point to each of the nonkey attributes that are functionally dependent on the primary key.

For the relation EMPLOYEE2 (Figure 4-2b), notice that (unlike EMPLOYEE1) EmpID does not uniquely identify a row in the relation. For example, there are two rows in the table for EmpID number 100. There are two types of functional dependencies in this relation:

1. EmpID → Name, DeptName, Salary
2. EmpID, CourseTitle → DateCompleted

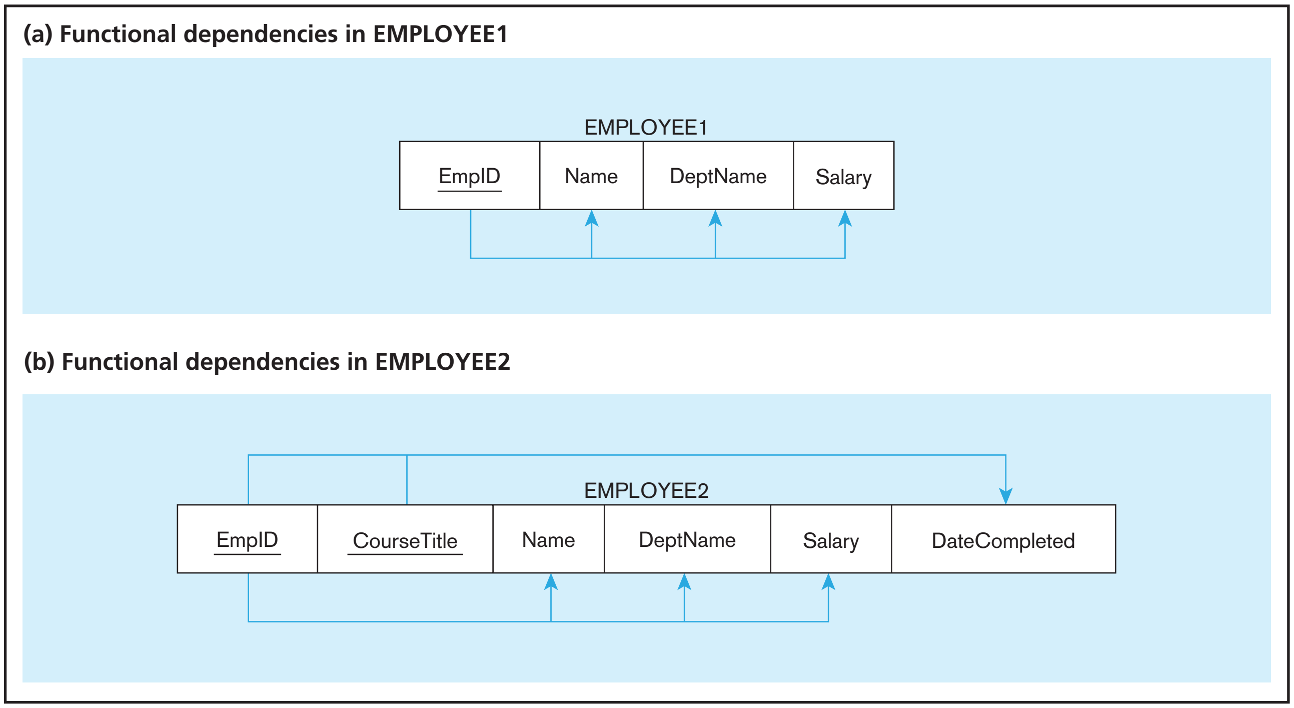


FIGURE 4-23 **Representing functional dependencies**

The functional dependencies indicate that the combination of EmpID and CourseTitle is the only candidate key (and therefore the primary key) for EMPLOYEE2. In other words, the primary key of EMPLOYEE2 is a composite key. Neither EmpID nor CourseTitle uniquely identifies a row in this relation and therefore (according to property 1) cannot by itself be a candidate key. Examine the data in Figure 4-2b to verify that the combination of EmpID and CourseTitle does uniquely identify each row of EMPLOYEE2. We represent the functional dependencies in this relation in Figure 4-23b. Notice that DateCompleted is the only attribute that is functionally dependent on the full primary key consisting of the attributes EmpID and CourseTitle.

We can summarize the relationship between determinants and candidate keys as follows: A candidate key is always a determinant, whereas a determinant may or may not be a candidate key. For example, in EMPLOYEE2, EmpID is a determinant but not a candidate key. A candidate key is a determinant that uniquely identifies the remaining (nonkey) attributes in a relation. A determinant may be a candidate key (such as EmpID in EMPLOYEE1), part of a composite candidate key (such as EmpID in EMPLOYEE2), or a nonkey attribute. We will describe examples of this shortly.

As a preview to the following illustration of what normalization accomplishes, normalized relations have as their primary key the determinant for each of the nonkeys, and within that relation there are no other functional dependencies.

PINE VALLEY FURNITURE

NORMALIZATION EXAMPLE: PINE VALLEY FURNITURE COMPANY

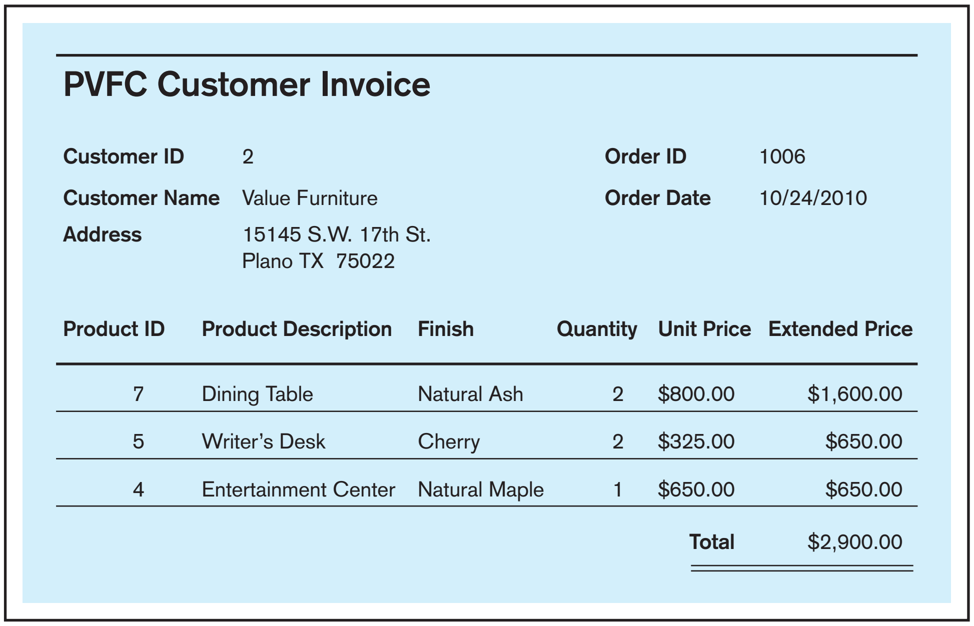
Now that we have examined functional dependencies and keys, we are ready to describe and illustrate the steps of normalization. If an EER data model has been transformed into a comprehensive set of relations for the database, then each of these relations needs to be normalized. In other cases in which the logical data model is being derived from user interfaces, such as screens, forms, and reports, you will want to create relations for each user interface and normalize those relations.

For a simple illustration, we use a customer invoice from Pine Valley Furniture Company (see Figure 4-24.)

Step 0: Represent the View in Tabular Form

The first step (preliminary to normalization) is to represent the user view (in this case, an invoice) as a single table, or relation, with the attributes recorded as column headings. Sample data should be recorded in the rows of the table, including any repeating

FIGURE 4-24 **Invoice (Pine Valley Furniture Company)**



PVFC Customer Invoice

Customer ID	2	**Order ID**	1006
Customer Name	Value Furniture	**Order Date**	10/24/2010
Address	15145 S.W. 17th St. Plano TX 75022		

Product ID	Product Description	Finish	Quantity	Unit Price	Extended Price
7	Dining Table	Natural Ash	2	$800.00	$1,600.00
5	Writer's Desk	Cherry	2	$325.00	$650.00
4	Entertainment Center	Natural Maple	1	$650.00	$650.00
				Total	$2,900.00

FIGURE 4-25 INVOICE data (Pine Valley Furniture Company)

OrderID	Order Date	Customer ID	Customer Name	Customer Address	ProductID	Product Description	Product Finish	Product StandardPrice	Ordered Quantity
1006	10/24/2010	2	Value Furniture	Plano, TX	7	Dining Table	Natural Ash	800.00	2
					5	Writer's Desk	Cherry	325.00	2
					4	Entertainment Center	Natural Maple	650.00	1
1007	10/25/2010	6	Furniture Gallery	Boulder, CO	11	4–Dr Dresser	Oak	500.00	4
					4	Entertainment Center	Natural Maple	650.00	3

groups that are present in the data. The table representing the invoice is shown in Figure 4-25. Notice that data for a second order (OrderID 1007) are included in Figure 4-25 to clarify further the structure of this data.

Step 1: Convert to First Normal Form

A relation is in **first normal form (1NF)** if the following two constraints both apply:

First normal form (1NF)
A relation that has a primary key and in which there are no repeating groups.

1. There are no repeating groups in the relation (thus, there is a single fact at the intersection of each row and column of the table).
2. A primary key has been defined, which uniquely identifies each row in the relation.

REMOVE REPEATING GROUPS As you can see, the invoice data in Figure 4-25 contain a repeating group for each product that appears on a particular order. Thus, OrderID 1006 contains three repeating groups, corresponding to the three products on that order.

In a previous section, we showed how to remove repeating groups from a table by filling relevant data values into previously vacant cells of the table (see Figures 4-2a and 4-2b). Applying this procedure to the invoice table yields the new table (named INVOICE) shown in Figure 4-26.

OrderID	Order Date	Customer ID	Customer Name	Customer Address	ProductID	Product Description	Product Finish	Product StandardPrice	Ordered Quantity
1006	10/24/2010	2	Value Furniture	Plano, TX	7	Dining Table	Natural Ash	800.00	2
1006	10/24/2010	2	Value Furniture	Plano, TX	5	Writer's Desk	Cherry	325.00	2
1006	10/24/2010	2	Value Furniture	Plano, TX	4	Entertainment Center	Natural Maple	650.00	1
1007	10/25/2010	6	Furniture Gallery	Boulder, CO	11	4–Dr Dresser	Oak	500.00	4
1007	10/25/2010	6	Furniture Gallery	Boulder, CO	4	Entertainment Center	Natural Maple	650.00	3

FIGURE 4-26 INVOICE relation (1NF) (Pine Valley Furniture Company)

SELECT THE PRIMARY KEY There are four determinants in INVOICE, and their functional dependencies are the following:

OrderID → OrderDate, CustomerID, CustomerName, CustomerAddress
CustomerID → CustomerName, CustomerAddress
ProductID → ProductDescription, ProductFinish, ProductStandardPrice
OrderID, ProductID → OrderedQuantity

Why do we know these are the functional dependencies? These business rules come from the organization. We know these from studying the nature of the Pine Valley Furniture Company business. We can also see that no data in Figure 4-26 violates any of these functional dependencies. But, because we don't see all possible rows of this table, we cannot be sure that there wouldn't be some invoice that would violate one of these functional dependencies. Thus, we must depend on our understanding of the rules of the organization.

As you can see, the only candidate key for INVOICE is the composite key consisting of the attributes OrderID and ProductID (because there is only one row in the table for any combination of values for these attributes). Therefore, OrderID and ProductID are underlined in Figure 4-26, indicating that they comprise the primary key.

When forming a primary key, you must be careful not to include redundant (and therefore unnecessary) attributes. Thus, although CustomerID is a determinant in INVOICE, it is not included as part of the primary key because all of the nonkey attributes are identified by the combination of OrderID and ProductID. We will see the role of CustomerID in the normalization process that follows.

A diagram that shows these functional dependencies for the INVOICE relation is shown in Figure 4-27. This diagram is a horizontal list of all the attributes in INVOICE, with the primary key attributes (OrderID and ProductID) underlined. Notice that the only attribute that depends on the full key is OrderedQuantity. All of the other functional dependencies are either partial dependencies or transitive dependencies (both are defined next).

ANOMALIES IN 1NF Although repeating groups have been removed, the data in Figure 4-26 still contain considerable redundancy. For example, CustomerID, CustomerName, and CustomerAddress for Value Furniture are recorded in three rows (at least) in the table. As a result of these redundancies, manipulating the data in the table can lead to anomalies such as the following:

1. ***Insertion anomaly*** With this table structure, the company is not able to introduce a new product (say, Breakfast Table with ProductID 8) and add it to the database before it is ordered the first time: No entries can be added to the table without both ProductID *and* OrderID. As another example, if a customer calls and requests another product be added to his OrderID 1007, a new row must be inserted in which the order date and all of the customer information must be repeated. This leads to data replication and potential data entry errors (e.g., the customer name may be entered as "Valley Furniture").

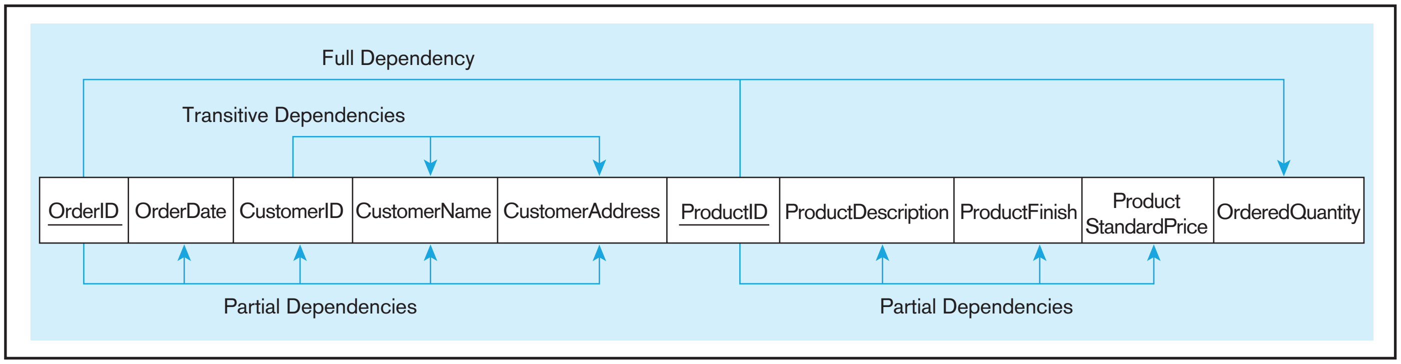


FIGURE 4-27 Functional dependency diagram for INVOICE

2. ***Deletion anomaly*** If a customer calls and requests that the Dining Table be deleted from her OrderID 1006, this row must be deleted from the relation, and we lose the information concerning this item's finish (Natural Ash) and price ($800.00).
3. ***Update anomaly*** If Pine Valley Furniture (as part of a price adjustment) increases the price of the Entertainment Center (ProductID 4) to $750.00, this change must be recorded in all rows containing that item. (There are two such rows in Figure 4-26.)

Step 2: Convert to Second Normal Form

We can remove many of the redundancies (and resulting anomalies) in the INVOICE relation by converting it to second normal form. A relation is in **second normal form (2NF)** if it is in first normal form and contains no partial functional dependencies. A **partial functional dependency** exists when a nonkey attribute is functionally dependent on part (but not all) of the primary key. As you can see, the following partial dependencies exist in Figure 4-27:

Second normal form (2NF)
A relation in first normal form in which every nonkey attribute is fully functionally dependent on the primary key.

Partial functional dependency
A functional dependency in which one or more nonkey attributes are functionally dependent on part (but not all) of the primary key.

OrderID → OrderDate, CustomerID, CustomerName, CustomerAddress
ProductID → ProductDescription, ProductFinish, ProductStandardPrice

The first of these partial dependencies (for example) states that the date on an order is uniquely determined by the order number and has nothing to do with the ProductID.

To convert a relation with partial dependencies to second normal form, the following steps are required:

1. Create a new relation for each primary key attribute (or combination of attributes) that is a determinant in a partial dependency. That attribute is the primary key in the new relation.
2. Move the nonkey attributes that are dependent on this primary key attribute (or attributes) from the old relation to the new relation.

The results of performing these steps for the INVOICE relation are shown in Figure 4-28. Removal of the partial dependencies results in the formation of two new relations: PRODUCT and CUSTOMER ORDER. The INVOICE relation is now left with just the primary key attributes (OrderID and ProductID) and OrderedQuantity, which is functionally dependent on the whole key. We rename this relation ORDER LINE, because each row in this table represents one line item on an order.

As indicated in Figure 4-28, the relations ORDER LINE and PRODUCT are in third normal form. However, CUSTOMER ORDER contains transitive dependencies and therefore (although in second normal form) is not yet in third normal form.

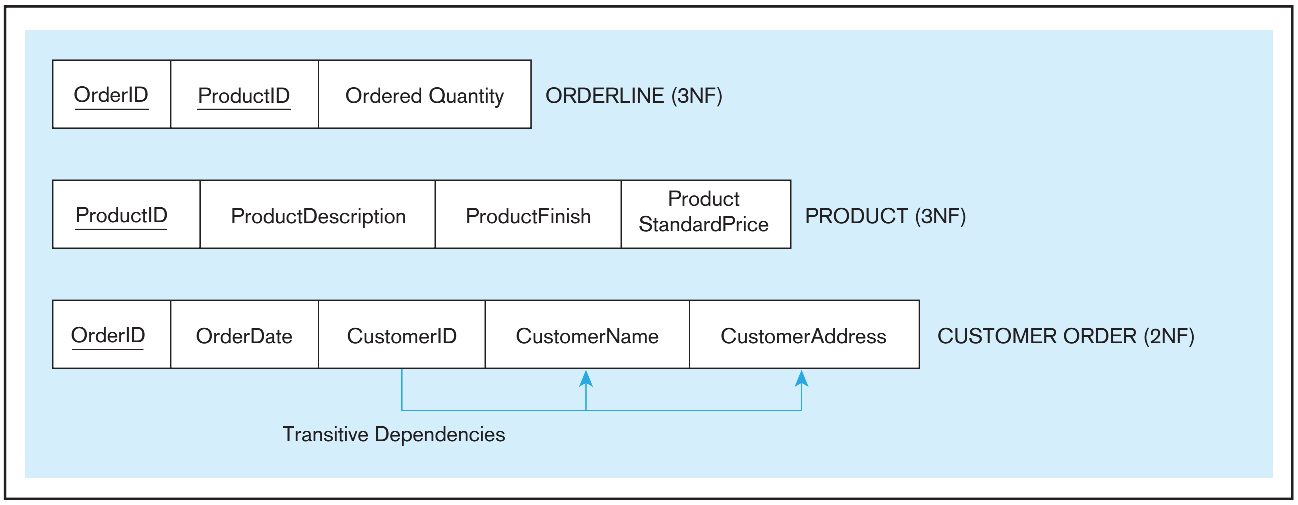


FIGURE 4-28 Removing partial dependencies

A relation that is in first normal form will be in second normal form if any one of the following conditions applies:

1. The primary key consists of only one attribute (e.g., the attribute ProductID in the PRODUCT relation in Figure 4-28). By definition, there cannot be a partial dependency in such a relation.
2. No nonkey attributes exist in the relation (thus all of the attributes in the relation are components of the primary key). There are no functional dependencies in such a relation.
3. Every nonkey attribute is functionally dependent on the full set of primary key attributes (e.g., the attribute OrderedQuantity in the ORDER LINE relation in Figure 4-28).

Step 3: Convert to Third Normal Form

Third normal form (3NF)
A relation that is in second normal form and has no transitive dependencies.

Transitive dependency
A functional dependency between the primary key and one or more nonkey attributes that are dependent on the primary key via another nonkey attribute.

A relation is in **third normal form (3NF)** if it is in second normal form and no transitive dependencies exist. A **transitive dependency** in a relation is a functional dependency between the primary key and one or more nonkey attributes that are dependent on the primary key via another nonkey attribute . For example, there are two transitive dependencies in the CUSTOMER ORDER relation shown in Figure 4-28:

OrderID → CustomerID → CustomerName
OrderID → CustomerID → CustomerAddress

In other words, both customer name and address are uniquely identified by CustomerID, but CustomerID is not part of the primary key (as we noted earlier).

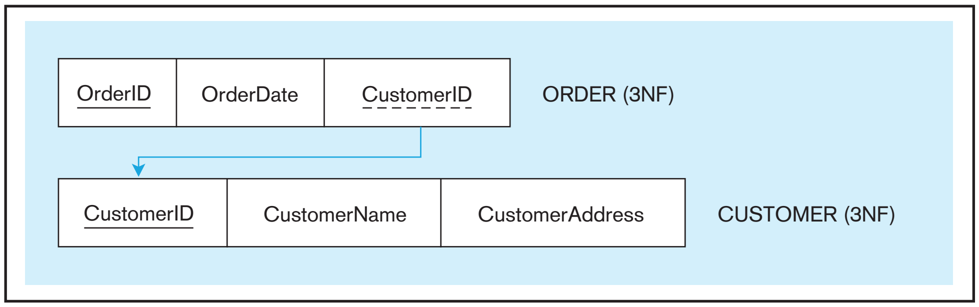
Transitive dependencies create unnecessary redundancy that may lead to the type of anomalies discussed earlier. For example, the transitive dependency in CUSTOMER ORDER (Figure 4-28) requires that a customer's name and address be reentered every time a customer submits a new order, regardless of how many times they have been entered previously. You have no doubt experienced this type of annoying requirement when ordering merchandise online, visiting a doctor's office, or any number of similar activities.

REMOVING TRANSITIVE DEPENDENCIES You can easily remove transitive dependencies from a relation by means of a three-step procedure:

1. For each nonkey attribute (or set of attributes) that is a determinant in a relation, create a new relation. That attribute (or set of attributes) becomes the primary key of the new relation.
2. Move all of the attributes that are functionally dependent on the primary key of the new relation from the old to the new relation.
3. Leave the attribute that serves as a primary key in the new relation in the old relation to serve as a foreign key that allows you to associate the two relations.

The results of applying these steps to the relation CUSTOMER ORDER are shown in Figure 4-29. A new relation named CUSTOMER has been created to receive the components of the transitive dependency. The determinant CustomerID becomes the

FIGURE 4-29 Removing transitive dependencies



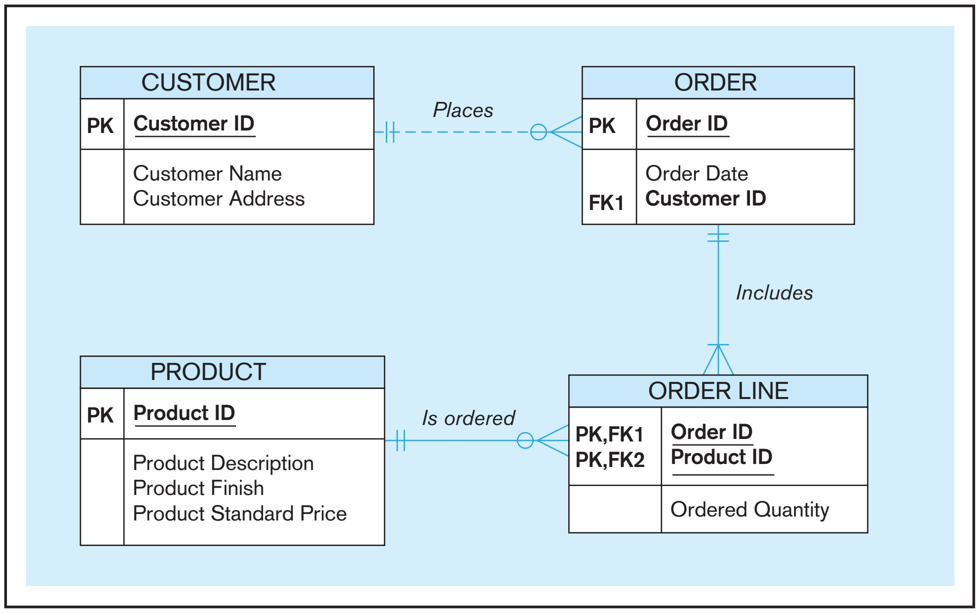


FIGURE 4-30 Relational schema for INVOICE data (Microsoft Visio notation)

primary key of this relation, and the attributes CustomerName and CustomerAddress are moved to the relation. CUSTOMER ORDER is renamed ORDER, and the attribute CustomerID remains as a foreign key in that relation. This allows us to associate an order with the customer who submitted the order. As indicated in Figure 4-29, these relations are now in third normal form.

Normalizing the data in the INVOICE view has resulted in the creation of four relations in third normal form: CUSTOMER, PRODUCT, ORDER, and ORDER LINE. A relational schema showing these four relations and their associations (developed using Microsoft Visio) is shown in Figure 4-30. Note that CustomerID is a foreign key in ORDER, and OrderID and ProductID are foreign keys in ORDER LINE. (Foreign keys are shown in Visio for logical, but not conceptual, data models.) Also note that minimum cardinalities are shown on the relationships even though the normalized relations provide no evidence of what the minimum cardinalities should be. Sample data for the relations might include, for example, a customer with no orders, thus providing evidence of the optional cardinality for the relationship Places. However, even if there were an order for every customer in a sample data set, this would not prove mandatory cardinality. Minimum cardinalities must be determined from business rules, not illustrations of reports, screens, and transactions. The same statement is true for specific maximum cardinalities (e.g., a business rule that no order may contain more than 10 line items).

Determinants and Normalization

We demonstrated normalization through 3NF in steps. There is an easy shortcut, however. If you look back at the original set of four determinants and the associated functional dependencies for the invoice user view, each of these corresponds to one of the relations in Figure 4-30, with each determinant being the primary key of a relation, and the nonkeys of each relation are those attributes functionally dependent on each determinant. There is a subtle but important difference: Because OrderID determines CustomerID, CustomerName, and CustomerAddress and CustomerID determines its dependent attributes, CustomerID becomes a foreign key in the ORDER relation, which is where CustomerName and CustomerAddress are represented. The point is, if you can determine determinants that have no overlapping dependent attributes, then you have defined the relations. Thus, you can do normalization step by step as illustrated for the Pine Valley Furniture invoice, or you can create relations in 3NF straight from determinants' functional dependencies.

Step 4: Further Normalization

After completing steps 0 through 3, all nonkeys will be dependent on the primary key, the whole primary key, and nothing but the primary key ("so help you Codd!"). Actually, normal forms are rules about functional dependencies and, hence, are the result of finding determinants and their associated nonkeys. The steps we outlined above are an aid in creating a relation for each determinant and its associated nonkeys.

You will recall from the beginning of our discussion of normalization that we identified additional normal forms beyond 3NF. The most commonly enforced of these additional normal forms are explained in Appendix B, which you might want to read or scan now.

MERGING RELATIONS

In a previous section, we described how to transform EER diagrams into relations. This transformation occurs when we take the results of a top-down analysis of data requirements and begin to structure them for implementation in a database. We then described how to check the resulting relations to determine whether they are in third (or higher) normal form and perform normalization steps if necessary.

As part of the logical design process, normalized relations may have been created from a number of separate EER diagrams and (possibly) other user views (i.e., there may be bottom-up or parallel database development activities for different areas of the organization as well as top-down ones). For example, besides the invoice used in the prior section to illustrate normalization, there may be an order form, an account balance report, production routing, and other user views, each of which have been normalized separately. The three-schema architecture for databases (see Chapter 1) encourages the simultaneous use of both top-down and bottom-up database development processes. In reality, most medium to large organizations have many reasonably independent systems development activities that at some point need to come together to create a shared database. The result is that some of the relations generated from these various processes may be redundant; that is, they may refer to the same entities. In such cases, we should merge those relations to remove the redundancy. This section describes merging relations (also called *view integration*). An understanding of how to merge relations is important for three reasons:

1. On large projects, the work of several subteams comes together during logical design, so there is often a need to merge relations.
2. Integrating existing databases with new information requirements often leads to the need to integrate different views.
3. New data requirements may arise during the life cycle, so there is a need to merge any new relations with what has already been developed.

An Example

Suppose that modeling a user view results in the following 3NF relation:

EMPLOYEE1(<u>EmployeeID</u>, Name, Address, Phone)

Modeling a second user view might result in the following relation:

EMPLOYEE2(<u>EmployeeID</u>, Name, Address, Jobcode, NoYears)

Because these two relations have the same primary key (EmployeeID), they likely describe the same entity and may be merged into one relation. The result of merging the relations is the following relation:

EMPLOYEE(<u>EmployeeID</u>, Name, Address, Phone, Jobcode, NoYears)

Notice that an attribute that appears in both relations (e.g., Name in this example) appears only once in the merged relation.

View Integration Problems

When integrating relations as in the preceding example, a database analyst must understand the meaning of the data and must be prepared to resolve any problems that may arise in that process. In this section, we describe and briefly illustrate four problems that arise in view integration: *synonyms*, *homonyms*, *transitive dependencies*, and *supertype/subtype relationships*.

SYNONYMS In some situations, two (or more) attributes may have different names but the same meaning (e.g., when they describe the same characteristic of an entity). Such attributes are called **synonyms**. For example, EmployeeID and EmployeeNo may be synonyms. When merging the relations that contain synonyms, you should obtain agreement (if possible) from users on a single, standardized name for the attribute and eliminate any other synonyms. (Another alternative is to choose a third name to replace the synonyms.) For example, consider the following relations:

Synonyms
Two (or more) attributes that have different names but the same meaning.

STUDENT1(<u>StudentID</u>, Name)
STUDENT2(<u>MatriculationNo</u>, Name, Address)

In this case, the analyst recognizes that both StudentID and MatriculationNo are synonyms for a person's student identity number and are identical attributes. (Another possibility is that these are both candidate keys, and only one of them should be selected as the primary key.) One possible resolution would be to standardize on one of the two attribute names, such as StudentID. Another option is to use a new attribute name, such as StudentNo, to replace both synonyms. Assuming the latter approach, merging the two relations would produce the following result:

STUDENT(<u>StudentNo</u>, Name, Address)

Often when there are synonyms, there is a need to allow some database users to refer to the same data by different names. Users may need to use familiar names that are consistent with terminology in their part of the organization. An **alias** is an alternative name used for an attribute. Many database management systems allow the definition of an alias that may be used interchangeably with the primary attribute label.

Alias
An alternative name used for an attribute.

HOMONYMS An attribute name that may have more than one meaning is called a **homonym**. For example, the term *account* might refer to a bank's checking account, savings account, loan account, or other type of account (and therefore *account* refers to different data, depending on how it is used).

Homonym
An attribute that may have more than one meaning.

You should be on the lookout for homonyms when merging relations. Consider the following example:

STUDENT1(<u>StudentID</u>, Name, Address)
STUDENT2(<u>StudentID</u>, Name, PhoneNo, Address)

In discussions with users, an analyst may discover that the attribute Address in STUDENT1 refers to a student's campus address, whereas in STUDENT2 the same attribute refers to a student's permanent (or home) address. To resolve this conflict, we would probably need to create new attribute names, so that the merged relation would become

STUDENT(<u>StudentID</u>, Name, PhoneNo, CampusAddress, PermanentAddress)

TRANSITIVE DEPENDENCIES When two 3NF relations are merged to form a single relation, transitive dependencies (described earlier in this chapter) may result. For example, consider the following two relations:

STUDENT1(StudentID, Major)
STUDENT2(StudentID, Advisor)

Because STUDENT1 and STUDENT2 have the same primary key, the two relations can be merged:

STUDENT(StudentID, Major, Advisor)

However, suppose that each major has exactly one advisor. In this case, Advisor is functionally dependent on Major:

Major $\rightarrow$ Advisor

If the preceding functional dependency exists, then STUDENT is in 2NF but not in 3NF, because it contains a transitive dependency. The analyst can create 3NF relations by removing the transitive dependency. Major becomes a foreign key in STUDENT:

STUDENT(StudentID, Major)
MAJOR ADVISOR(Major, Advisor)

SUPERTYPE/SUBTYPE RELATIONSHIPS These relationships may be hidden in user views or relations. Suppose that we have the following two hospital relations:

PATIENT1(PatientID, Name, Address)
PATIENT2(PatientID, RoomNo)

Initially, it appears that these two relations can be merged into a single PATIENT relation. However, the analyst correctly suspects that there are two different types of patients: resident patients and outpatients. PATIENT1 actually contains attributes common to all patients. PATIENT2 contains an attribute (RoomNo) that is a characteristic only of resident patients. In this situation, the analyst should create supertype/subtype relationships for these entities:

PATIENT(PatientID, Name, Address)
RESIDENT PATIENT(PatientID, RoomNo)
OUTPATIENT(PatientID, DateTreated)

We have created the OUTPATIENT relation to show what it might look like if it were needed, but it is not necessary, given only PATIENT1 and PATIENT2 user views. For an extended discussion of view integration in database design, see Navathe, Elmasri, and Larson (1986).

A FINAL STEP FOR DEFINING RELATIONAL KEYS

In Chapter 2, we provided some criteria for selecting identifiers: They do not change values over time and must be unique and known, nonintelligent, and use a single attribute surrogate for composite identifier. Actually, none of these criteria must apply until the database is implemented (i.e., when the identifier becomes a primary key and is defined as a field in the physical database). Before the relations are defined as tables, the primary keys of relations should, if necessary, be changed to conform to these criteria.

Recently database experts (e.g., Johnston, 2000) have strengthened the criteria for primary key specification. Experts now also recommend that a primary key be unique across *the whole database* (a so-called **enterprise key**), not just unique within the relational table to which it applies. This criterion makes a primary key more like what in object-oriented databases is called an *object identifier* (see Chapters 13 and 14). With this recommendation, the primary key of a relation becomes a value internal to the database system and has no business meaning.

Enterprise key
A primary key whose value is unique across all relations.

A candidate primary key, such as EmpID in the EMPLOYEE1 relation of Figure 4-1 or CustomerID in the CUSTOMER relation (Figure 4-29), if ever used in the organization, is called a *business key* or *natural key* and would be included in the relation as a nonkey attribute. The EMPLOYEE1 and CUSTOMER relations (and every other relation in the database) then have a new enterprise key attribute (called, say, ObjectID), which has no business meaning.

Why create this extra attribute? One of the main motivations for using an enterprise key is database evolvability—merging new relations into a database after the database is created. For example, consider the following two relations:

EMPLOYEE(<u>EmpID</u>, EmpName, DeptName, Salary)
CUSTOMER(<u>CustID</u>, CustName, Address)

In this example, without an enterprise key, EmpID and CustID may or may not have the same format, length, and data type, whether they are intelligent or nonintelligent. Suppose the organization evolves its information processing needs and recognizes that employees can also be customers, so employee and customer are simply two subtypes of the same PERSON supertype. (You saw this in Chapter 3, when studying universal data modeling.) Thus, the organization would then like to have three relations:

PERSON(<u>PersonID</u>, PersonName)
EMPLOYEE(<u>PersonID</u>, DeptName, Salary)
CUSTOMER(<u>PersonID</u>, Address)

In this case, PersonID is supposed to be the same value for the same person throughout all roles. But if values for EmpID and CustID were selected before relation PERSON was created, the values for EmpID and CustID probably will not match. Moreover, if we change the values of EmpID and CustID to match the new PersonID, how do we ensure that all EmpIDs and CustIDs are unique if another employee or customer already has the associated PersonID value? Even worse, if there are other tables that relate to, say, EMPLOYEE, then foreign keys in these other tables have to change, creating a ripple effect of foreign key changes. The only way to guarantee that each primary key of a relation is unique across the database is to create an enterprise key from the very beginning so primary keys never have to change.

In our example, the original database (without PERSON) with an enterprise key is shown in Figures 4-31a (the relations) and 4-31b (sample data). In this figure, EmpID and CustID are now business keys, and OBJECT is the supertype of all other relations. OBJECT can have attributes such as the name of the type of object (included in this example as attribute ObjectType), date created, date last changed, or any other internal system attributes for an object instance. Then, when PERSON is needed, the database evolves to the design shown in Figures 4-31c (the relations) and 4-31d (sample data). Evolution to the database with PERSON still requires some alterations to existing tables, but not to primary key values. The name attribute is moved to PERSON because it is common to both subtypes, and a foreign key is added to EMPLOYEE and CUSTOMER to point to the common person instance. As you will see in Chapter 6, it is easy to add and delete nonkey columns, even foreign keys, to table definitions. In contrast, changing the primary key of a relation is not allowed by most database management systems because of the extensive cost of the foreign key ripple effect.

FIGURE 4-31 Enterprise key

(a) Relations with enterprise key

OBJECT (OID, ObjectType)
EMPLOYEE (OID, EmpID, EmpName, DeptName, Salary)
CUSTOMER (OID, CustID, CustName, Address)

(b) Sample data with enterprise key

OBJECT

OID	ObjectType
1	EMPLOYEE
2	CUSTOMER
3	CUSTOMER
4	EMPLOYEE
5	EMPLOYEE
6	CUSTOMER
7	CUSTOMER

EMPLOYEE

OID	EmpID	EmpName	DeptName	Salary
1	100	Jennings, Fred	Marketing	50000
4	101	Hopkins, Dan	Purchasing	45000
5	102	Huber, Ike	Accounting	45000

CUSTOMER

OID	CustID	CustName	Address
2	100	Fred's Warehouse	Greensboro, NC
3	101	Bargain Bonanza	Moscow, ID
6	102	Jasper's	Tallahassee, FL
7	103	Desks 'R Us	Kettering, OH

(c) Relations after adding the PERSON relation

OBJECT (OID, ObjectType)
EMPLOYEE (OID, EmpID, DeptName, Salary, PersonID)
CUSTOMER (OID, CustID, Address, PersonID)
PERSON (OID, Name)

(continued)

Summary

Logical database design is the process of transforming the conceptual data model into a logical data model. The emphasis in this chapter has been on the relational data model, because of its importance in contemporary database systems. The relational data model represents data in the form of tables called relations. A relation is a named, two-dimensional table of data. A key property of relations is that they cannot contain multivalued attributes.

In this chapter, we described the major steps in the logical database design process. This process is based on transforming EER diagrams into normalized relations. This process has three steps: Transform EER diagrams into relations, normalize the relations, and merge the relations. The result of this process is a set of relations in third normal form that can be implemented using any contemporary relational database management system.

FIGURE 4-31 (*continued*)

(d) Sample data after adding the PERSON relation

OBJECT

OID	ObjectType
1	EMPLOYEE
2	CUSTOMER
3	CUSTOMER
4	EMPLOYEE
5	EMPLOYEE
6	CUSTOMER
7	CUSTOMER
8	PERSON
9	PERSON
10	PERSON
11	PERSON
12	PERSON
13	PERSON
14	PERSON

PERSON

OID	Name
8	Jennings, Fred
9	Fred's Warehouse
10	Bargain Bonanza
11	Hopkins, Dan
12	Huber, Ike
13	Jasper's
14	Desks 'R Us

EMPLOYEE

OID	EmpID	DeptName	Salary	PersonID
1	100	Marketing	50000	8
4	101	Purchasing	45000	11
5	102	Accounting	45000	12

CUSTOMER

OID	CustID	Address	PersonID
2	100	Greensboro, NC	9
3	101	Moscow, ID	10
6	102	Tallahassee, FL	13
7	103	Kettering, OH	14

Each entity type in the EER diagram is transformed into a relation that has the same primary key as the entity type. A one-to-many relationship is represented by adding a foreign key to the relation that represents the entity on the many side of the relationship. (This foreign key is the primary key of the entity on the one side of the relationship.) A many-to-many relationship is represented by creating a separate relation. The primary key of this relation is a composite key, consisting of the primary key of each of the entities that participate in the relationship.

The relational model does not directly support supertype/subtype relationships, but we can model these relationships by creating a separate table (or relation) for the supertype and for each subtype. The primary key of each subtype is the same (or at least from the same domain) as for the supertype. The supertype must have an attribute called the subtype discriminator that indicates to which subtype (or subtypes) each instance of the supertype belongs.

The purpose of normalization is to derive well-structured relations that are free of anomalies (inconsistencies or errors) that would otherwise result when the relations are updated or modified. Normalization is based on the analysis of functional dependencies, which are constraints between two attributes (or two sets of attributes). It may be accomplished in several stages. Relations in first normal form (1NF) contain no multivalued attributes or repeating groups. Relations in second normal form (2NF) contain no partial dependencies, and relations in third normal form (3NF) contain no transitive dependencies. We can use diagrams that show the functional dependencies in a relation to help decompose that relation (if necessary) to obtain relations in third normal form. Higher normal forms (beyond 3NF) have also been defined; we discuss these normal forms in Appendix B.

We must be careful when combining relations to deal with problems such as synonyms, homonyms, transitive dependencies, and supertype/subtype relationships. In addition, before relations are defined to the database management system, all primary keys should be described as single-attribute nonintelligent keys and, preferably, as enterprise keys.

Chapter Review

Key Terms

Alias *189*
Anomaly *164*
Candidate key *181*
Composite key *157*
Determinant *181*
Enterprise key *191*
Entity integrity rule *161*
First normal form (1NF) *183*
Foreign key *158*
Functional dependency *179*
Homonym *189*
Normal form *179*
Normalization *179*
Null *161*
Partial functional dependency *185*
Primary key *157*
Recursive foreign key *173*
Referential integrity constraint *162*
Relation *157*
Second normal form (2NF) *185*
Surrogate primary key *168*
Synonyms *189*
Third normal form (3NF) *186*
Transitive dependency *186*
Well-structured relation *164*

Review Questions

1. Define each of the following terms:
  a. determinant
  b. functional dependency
  c. transitive dependency
  d. recursive foreign key
  e. normalization
  f. composite key
  g. relation
  h. normal form
  i. partial functional dependency
  j. enterprise key
  k. surrogate primary key

2. Match the following terms to the appropriate definitions:

____ well-structured relation
____ anomaly
____ functional dependency
____ determinant
____ composite key
____ 1NF
____ 2NF
____ 3NF
____ recursive foreign key
____ relation
____ transitive dependency

  a. constraint between two attributes
  b. functional dependency between the primary key and a nonkey attribute via another nonkey attribute
  c. references the primary key in the same relation
  d. multivalued attributes removed
  e. inconsistency or error
  f. contains little redundancy
  g. contains two (or more) attributes
  h. contains no partial functional dependencies
  i. transitive dependencies eliminated
  j. attribute on left side of functional dependency
  k. named two-dimensional table of data

3. Contrast the following terms:
  a. normal form; normalization
  b. candidate key; primary key
  c. partial dependency; transitive dependency
  d. composite key; recursive foreign key
  e. determinant; candidate key
  f. foreign key; primary key
  g. enterprise key; surrogate key

4. Describe the primary differences between the conceptual and logical data models.

5. Summarize six important properties of relations.

6. Describe two properties that each candidate key must satisfy.

7. Describe three types of anomalies that can arise in a table and the negative consequences of each.

8. Fill in the blanks in each of the following statements:
  a. A relation that has no partial functional dependencies is in ____ normal form.
  b. A relation that has no transitive dependencies is in ____ normal form.
  c. A relation that has no multivalued attributes is in ____ normal form.

9. What is a well-structured relation? Why are well-structured relations important in logical database design?

10. Describe the primary way in which relationships in an E-R diagram are expressed in a corresponding relational data model.

11. Describe how the following components of an E-R diagram are transformed into relations:
  a. regular entity type
  b. relationship (1:*M*)
  c. relationship (*M*:*N*)
  d. relationship (supertype/subtype)
  e. multivalued attribute
  f. weak entity
  g. composite attribute

12. What is the primary purpose of normalization?

13. Briefly describe four typical problems that often arise in merging relations and common techniques for addressing those problems.

14. List three conditions that you can apply to determine whether a relation that is in first normal form is also in second normal form.

15. Explain how each of the following types of integrity constraints is enforced in the SQL CREATE TABLE commands:
  a. entity integrity
  b. referential integrity

16. What are the benefits of enforcing the integrity constraints as part of the database design and implementation process (instead of doing it in application design)?

17. How are relationships between entities represented in the relational data model?

18. How do you represent a 1:*M* unary relationship in a relational data model?

19. How do you represent an *M:N* ternary relationship in a relational data model?
20. How do you represent an associative entity in a relational data model?
21. What is the relationship between the primary key of a relation and the functional dependencies among all attributes within that relation?
22. Under what conditions must a foreign key not be null?
23. Explain what can be done with primary keys to eliminate key ripple effects as a database evolves.
24. Describe the difference between how a 1:*M* unary relationship and an *M:N* unary relationship are implemented in a relational data model.
25. Explain three conditions that suggest a surrogate key should be created for the primary key of a relation.

Problems and Exercises

1. For each of the following E-R diagrams from Chapter 2:
I. Transform the diagram to a relational schema that shows referential integrity constraints (see Figure 4-5 for an example of such a schema).
II. For each relation, diagram the functional dependencies (see Figure 4-23 for an example).
III. If any of the relations are not in 3NF, transform them to 3NF.
a. Figure 2-8
b. Figure 2-9b
c. Figure 2-11a
d. Figure 2-11b
e. Figure 2-15a (relationship version)
f. Figure 2-15b (attribute version)
g. Figure 2-16b
h. Figure 2-19

2. For each of the following EER diagrams from Chapter 3:
I. Transform the diagram into a relational schema that shows referential integrity constraints (see Figure 4-5 for an example of such a schema).
II. For each relation, diagram the functional dependencies (see Figure 4-23 for an example).
III. If any of the relations are not in 3NF, transform them to 3NF.
a. Figure 3-6b
b. Figure 3-7a
c. Figure 3-9
d. Figure 3-10
e. Figure 3-11

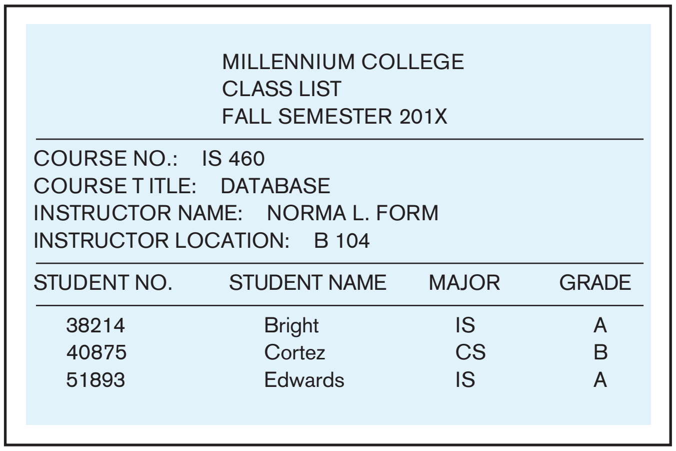
3. For each of the following relations, indicate the normal form for that relation. If the relation is not in third normal form, decompose it into 3NF relations. Functional dependencies (other than those implied by the primary key) are shown where appropriate.
a. CLASS(CourseNo, SectionNo)
b. CLASS(CourseNo, SectionNo, Room)
c. CLASS(CourseNo, SectionNo, Room, Capacity) [FD: Room → Capacity]
d. CLASS(CourseNo, SectionNo, CourseName, Room, Capacity) [FD: CourseNo → CourseName; FD: Room → Capacity]

4. For your answers to the following Problems and Exercises from prior chapters, transform the EER diagrams into a set of relational schemas, diagram the functional dependencies, and convert all the relations to third normal form:
a. Chapter 2, Problem and Exercise 15b
b. Chapter 2, Problem and Exercise 15g
c. Chapter 2, Problem and Exercise 15h
d. Chapter 2, Problem and Exercise 15i
e. Chapter 2, Problem and Exercise 21
f. Chapter 2, Problem and Exercise 24

5. Figure 4-32 shows a class list for Millennium College. Convert this user view to a set of 3NF relations using an enterprise key. Assume the following:
- An instructor has a unique location.
- A student has a unique major.
- A course has a unique title.

6. Figure 4-33 (page 196) shows an EER diagram for a simplified credit card environment. There are two types of card accounts: debit cards and credit cards. Credit card accounts accumulate charges with merchants. Each charge is identified by the date and time of the charge as well as the primary keys of merchant and credit card.
a. Develop a relational schema.
b. Show the functional dependencies.
c. Develop a set of 3NF relations using an enterprise key.

7. Table 4-3 (page 196) contains sample data for parts and for vendors who supply those parts. In discussing these data with users, we find that part numbers (but not descriptions) uniquely identify parts and that vendor names uniquely identify vendors.
a. Convert this table to a relation (named PART SUPPLIER) in first normal form. Illustrate the relation with the sample data in the table.
b. List the functional dependencies in PART SUPPLIER and identify a candidate key.
c. For the relation PART SUPPLIER, identify each of the following: an insert anomaly, a delete anomaly, and a modification anomaly.



MILLENNIUM COLLEGE
CLASS LIST
FALL SEMESTER 201X

COURSE NO.: IS 460
COURSE T ITLE: DATABASE
INSTRUCTOR NAME: NORMA L. FORM
INSTRUCTOR LOCATION: B 104

STUDENT NO.	STUDENT NAME	MAJOR	GRADE
38214	Bright	IS	A
40875	Cortez	CS	B
51893	Edwards	IS	A

FIGURE 4-32 Class list (Millennium College)

FIGURE 4-33 EER diagram for bank cards

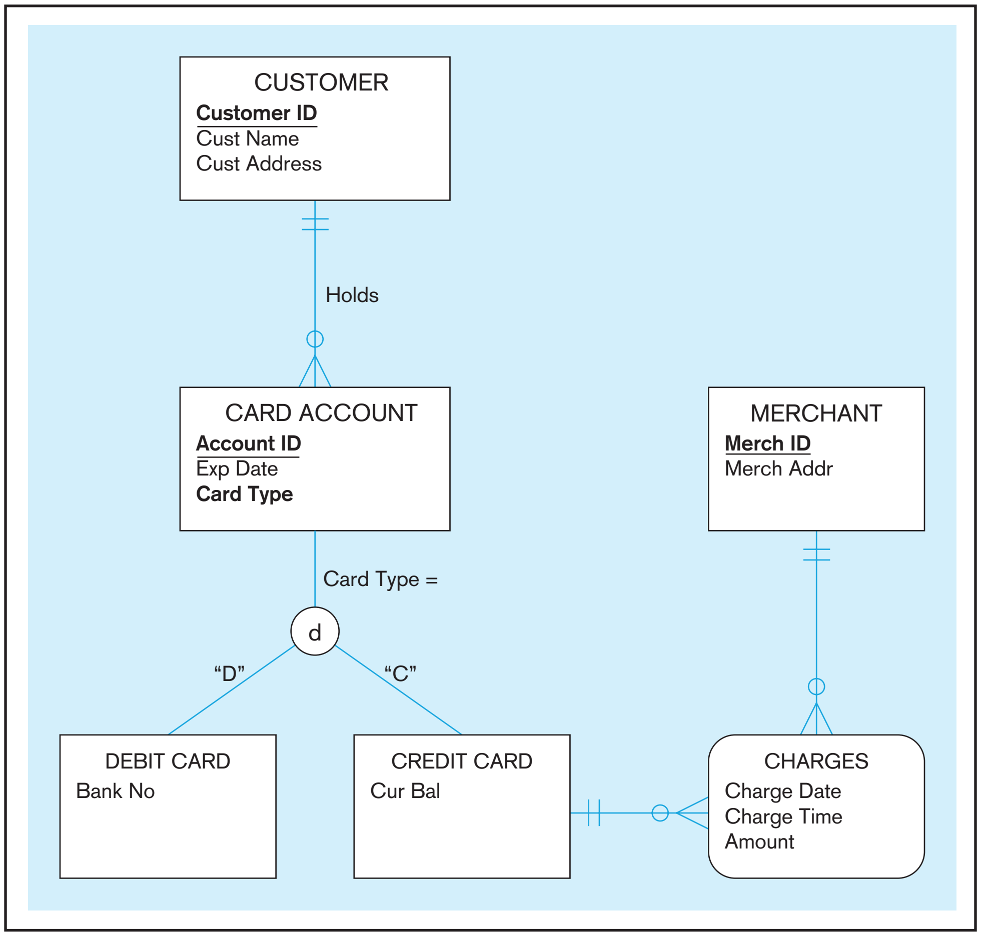


TABLE 4-3 Sample Data for Parts and Vendors

Part No	Description	Vendor Name	Address	Unit Cost
1234	Logic chip	Fast Chips	Cupertino	10.00
		Smart Chips	Phoenix	8.00
5678	Memory chip	Fast Chips	Cupertino	3.00
		Quality Chips	Austin	2.00
		Smart Chips	Phoenix	5.00

d. Draw a relational schema for PART SUPPLIER and show the functional dependencies.
e. In what normal form is this relation?
f. Develop a set of 3NF relations from PART SUPPLIER.
g. Show the 3NF relations using Microsoft Visio (or any other tool specified by your instructor).

8. Table 4-4 shows a relation called GRADE REPORT for a university. Your assignment is as follows:
a. Draw a relational schema and diagram the functional dependencies in the relation.
b. In what normal form is this relation?
c. Decompose GRADE REPORT into a set of 3NF relations.
d. Draw a relational schema for your 3NF relations and show the referential integrity constraints.
e. Draw your answer to part d using Microsoft Visio (or any other tool specified by your instructor).

9. Table 4-5 shows a shipping manifest. Your assignment is as follows:
a. Draw a relational schema and diagram the functional dependencies in the relation.
b. In what normal form is this relation?
c. Decompose MANIFEST into a set of 3NF relations.
d. Draw a relational schema for your 3NF relations and show the referential integrity constraints.
e. Draw your answer to part d using Microsoft Visio (or any other tool specified by your instructor).

10. Transform the relational schema developed in Problem and Exercise 9 into an EER diagram. State any assumptions that you have made.

11. For your answers to the following Problems and Exercises from prior chapters, transform the EER diagrams into a set of relational schemas, diagram the functional

TABLE 4-4 Grade Report Relation

Grade Report

StudentID	StudentName	CampusAddress	Major	CourseID	CourseTitle	Instructor Name	Instructor Location	Grade
168300458	Williams	208 Brooks	IS	IS 350	Database Mgt	Codd	B 104	A
168300458	Williams	208 Brooks	IS	IS 465	Systems Analysis	Parsons	B 317	B
543291073	Baker	104 Phillips	Acctg	IS 350	Database Mgt	Codd	B 104	C
543291073	Baker	104 Phillips	Acctg	Acct 201	Fund Acctg	Miller	H 310	B
543291073	Baker	104 Phillips	Acctg	Mkgt 300	Intro Mktg	Bennett	B 212	A

TABLE 4-5 Shipping Manifest

Shipment ID:	**00-0001**	**Shipment Date:**	**01/10/2010**
Origin:	Boston	Expected Arrival:	01/14/2010
Destination:	Brazil		
Ship Number:	39	Captain:	002-15
			Henry Moore

Item Number	Type	Description	Weight	Quantity	TOTALWEIGHT
3223	BM	Concrete Form	500	100	50,000
3297	BM	Steel Beam	87	2,000	174,000
				Shipment Total:	224,000

TABLE 4-6 Parking Tickets at Millennium College

Parking Ticket Table

St ID	L Name	F Name	Phone No	St Lic	Lic No	Ticket #	Date	Code	Fine
38249	Brown	Thomas	111-7804	FL	BRY 123	15634	10/17/10	2	$25
						16017	11/13/10	1	$15
82453	Green	Sally	391-1689	AL	TRE 141	14987	10/05/10	3	$100
						16293	11/18/10	1	$15
						17892	12/13/10	2	$25

dependencies, and convert all the relations to third normal form.

a. Chapter 3, Problem and Exercise 15
b. Chapter 3, Problem and Exercise 20

12. Transform Figure 2-15b, attribute version, to 3NF relations. Transform Figure 2-15b, relationship version, to 3NF relations. Compare these two sets of 3NF relations with those in Figure 4-10. What observations and conclusions do you reach by comparing these different sets of 3NF relations?

13. The Public Safety office at Millennium College maintains a list of parking tickets issued to vehicles parked illegally on the campus. Table 4-6 shows a portion of this list for the fall semester. (Attribute names are abbreviated to conserve space.)
    a. Convert this table to a relation in first normal form by entering appropriate data in the table. What are the determinants in this relation?
    b. Draw a dependency diagram that shows all functional dependencies in the relation, based on the sample data shown.

c. Give an example of one or more anomalies that can result in using this relation.

d. Develop a set of relations in third normal form. Include a new column with the heading Violation in the appropriate table to explain the reason for each ticket. Values in this column are: expired parking meter (ticket code 1), no parking permit (ticket code 2), and handicap violation (ticket code 3).

e. Develop an E-R diagram with the appropriate cardinality notations.

14. The materials manager at Pine Valley Furniture Company maintains a list of suppliers for each of the material items purchased by the company from outside vendors. Table 4-7 shows the essential data required for this application.

a. Draw a dependency diagram for this data. You may assume the following:

- Each material item has one or more suppliers. Each supplier may supply one or more items or may not supply any items.
- The unit price for a material item may vary from one vendor to another.
- The terms code for a supplier uniquely identifies the terms of the sale (e.g., code 2 means 10 percent net 30 days, etc.). The terms for a supplier are the same for all material items ordered from that supplier.

b. Decompose this diagram into a set of diagrams in 3NF.

c. Draw an E-R diagram for this situation.

15. Table 4-8 shows a portion of a shipment table for a large manufacturing company. Each shipment (identified by Shipment#) uniquely identifies the shipment Origin, Destination, and Distance. The shipment Origin and Destination pair also uniquely identifies the Distance.

a. Develop a diagram that shows the functional dependencies in the SHIPMENT relation.

b. In what normal form is SHIPMENT? Why?

c. Convert SHIPMENT to third normal form if necessary. Show the resulting table(s) with the sample data presented in SHIPMENT.

16. Figure 4-34 shows an EER diagram for Vacation Property Rentals. This organization rents preferred properties in

TABLE 4-7 Pine Valley Furniture Company Purchasing Data

Attribute Name	Sample Value
Material ID	3792
Material Name	Hinges 3″ locking
Unit of Measure	each
Standard Cost	$5.00
Vendor ID	V300
Vendor Name	Apex Hardware
Unit Price	$4.75
Terms Code	1
Terms	COD

TABLE 4-8 Shipment Relation

Shipment#	Origin	Destination	Distance
409	Seattle	Denver	1,537
618	Chicago	Dallas	1,058
723	Boston	Atlanta	1,214
824	Denver	Los Angeles	975
629	Seattle	Denver	1,537

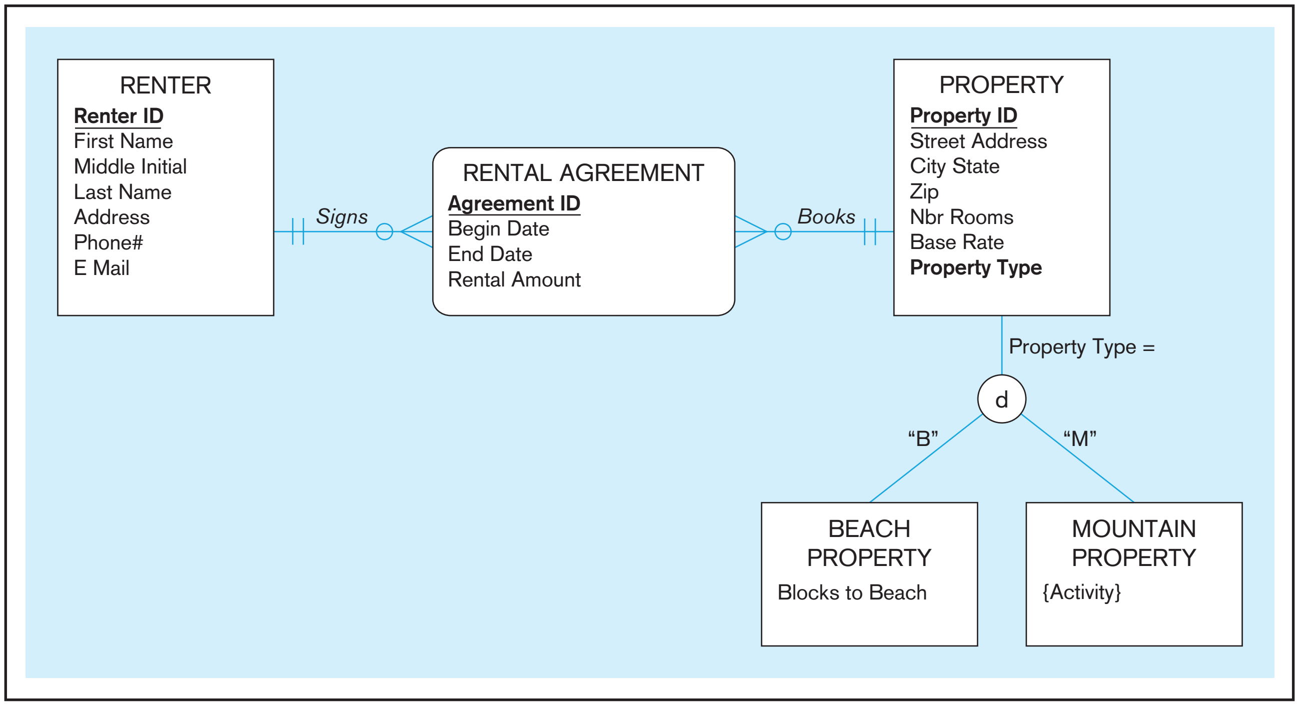


FIGURE 4-34 EER diagram for Vacation Property Rentals

several states. As shown in the figure, there are two basic types of properties: beach properties and mountain properties.

a. Transform the EER diagram to a set of relations and develop a relational schema.
b. Diagram the functional dependencies and determine the normal form for each relation.
c. Convert all relations to third normal form, if necessary, and draw a revised relational schema.
d. Suggest an integrity constraint that would ensure that no property is rented twice during the same time interval.

17. For your answers to Problem and Exercise 16 from Chapter 3, transform the EER diagrams into a set of relational schemas, diagram the functional dependencies, and convert all the relations to third normal form.

18. Figure 4-35 includes an EER diagram describing a car racing league. Transform the diagram into a relational schema that shows referential integrity constraints (see Figure 4-5 for an example of such a schema). In addition, verify that the resulting relations are in 3NF.

19. Figure 4-36 includes an EER diagram for a medium-size software vendor. Transform the diagram into a relational schema that shows referential integrity constraints (see Figure 4-5 for an example of such a schema). In addition, verify that the resulting relations are in 3NF.

20. Examine the set of relations in Figure 4-37. What normal form are these in? How do you know this? If they are in 3NF, convert the relations into an EER diagram. What assumptions did you have to make to answer these questions?

21. A pet store currently uses a legacy flat file system to store all of its information. The owner of the store, Peter Corona, wants to implement a Web-enabled database application. This would enable branch stores to enter data regarding inventory levels, ordering, and so on. Presently, the data for inventory and sales tracking are stored in one file that has the following format:

StoreName, PetName, Pet Description, Price, Cost, SupplierName, ShippingTime, QuantityOnHand, DateOfLastDelivery, DateOfLastPurchase, DeliveryDate1, DeliveryDate2, DeliveryDate3, DeliveryDate4, PurchaseDate1, PurchaseDate2, PurchaseDate3, PurchaseDate4, LastCustomerName, CustomerName1, CustomerName2, CustomerName3, CustomerName4

Assume that you want to track all purchase and inventory data, such as who bought the fish, the date that it was purchased, the date that it was delivered, and so on. The present file format allows only the tracking of the last purchase and delivery as well as four prior purchases and deliveries. You can assume that a type of fish is supplied by one supplier.

a. Show all functional dependencies.
b. What normal form is this table in?
c. Design a normalized data model for these data. Show that it is in 3NF.

22. For Problem and Exercise 21, draw the ER diagram based on the normalized relations.

23. How would Problems and Exercises 21 and 22 change if a type of fish could be supplied by multiple suppliers?

24. Figure 4-38 shows an EER diagram for a university dining service organization that provides dining services to a major university.

a. Transform the EER diagram to a set of relations and develop a relational schema.
b. Diagram the functional dependencies and determine the normal form for each relation.
c. Convert all relations to third normal form, if necessary, and draw a revised relational schema.

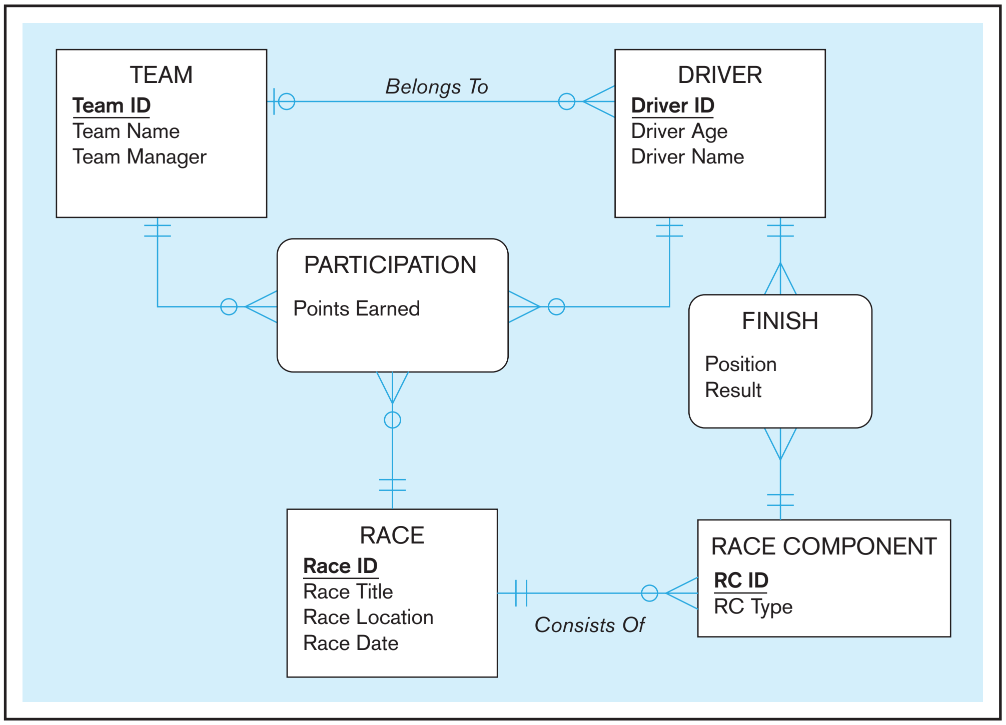
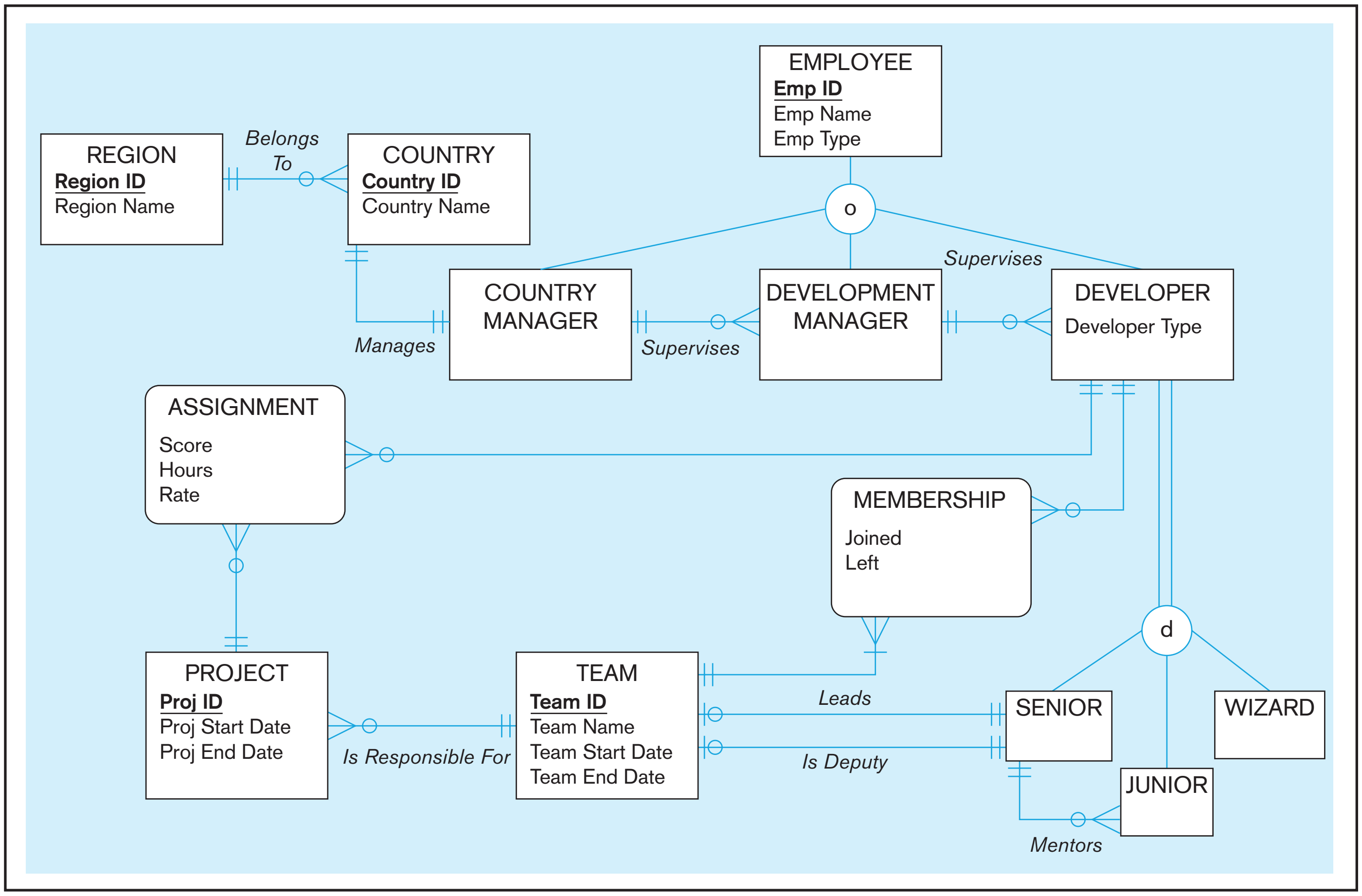
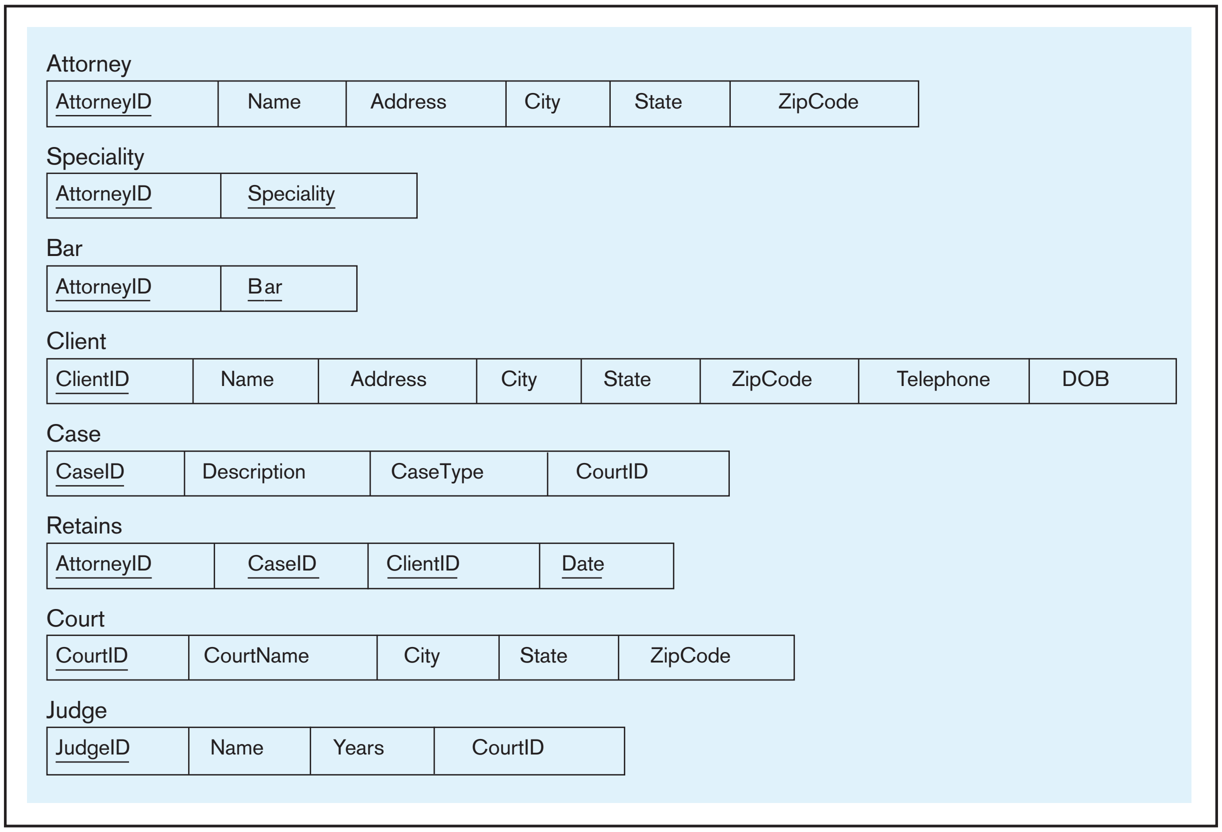


FIGURE 4-35 **EER diagram for a car racing league**

FIGURE 4-36 **EER diagram for a middle-size software vendor**





Attorney

AttorneyID	Name	Address	City	State	ZipCode

Speciality

AttorneyID	Speciality

Bar

AttorneyID	Bar

Client

ClientID	Name	Address	City	State	ZipCode	Telephone	DOB

Case

CaseID	Description	CaseType	CourtID

Retains

AttorneyID	CaseID	ClientID	Date

Court

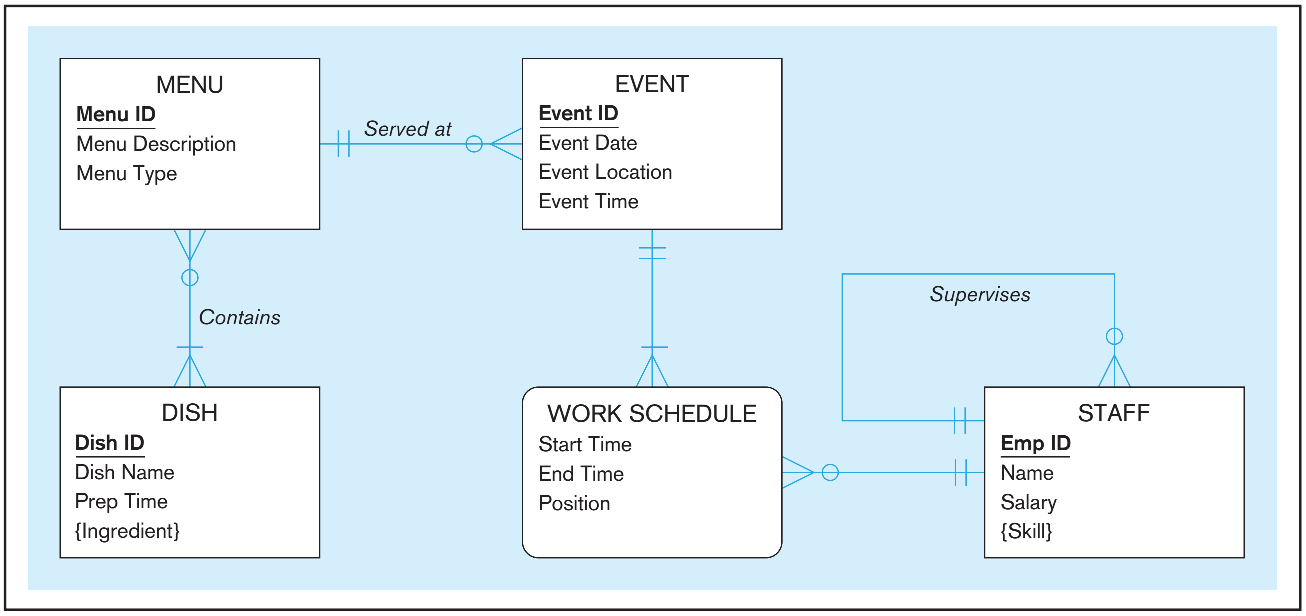
CourtID	CourtName	City	State	ZipCode

Judge

JudgeID	Name	Years	CourtID

FIGURE 4-37 **Relations for Problem and Exercise 20**

FIGURE 4-38 EER diagram for university dining services



25. The following attributes form a relation that includes information about individual computers, their vendors, software packages running on the computers, computer users, and user authorizations. Users are authorized to use a specific software package on a specific computer during a specific timeframe (characterized with attributes UserAuthorization Starts and UserAuthorizationEnds and secured with UserAuthorizationPassword). Software is licensed to be used on specific computers (potentially multiple software packages at the same time) until an expiration time (SoftwareLicenceExpires) at a specific price. Computers are sold by vendors, and each vendor has a support person with an ID, name, and phone extension. Each individual computer has a specific purchase price. The attributes are as follows:

ComputerSerialNbr, VendorID, VendorName, VendorPhone, VendorSupportID, VendorSupportName, VendorSupportExtension, SoftwareID, SoftwareName, SoftwareVendor, SoftwareLicenceExpires, SoftwareLicencePrice, UserID, UserName, UserAuthorizationStarts, UserAuthorizationEnds, UserAuthorizationPassword, PurchasePrice

Based on this information,

a. Identify the functional dependencies between the attributes.
b. Identify the reasons why this relation is not in 3NF.
c. Present the attributes organized so that the resulting relations are in 3NF.

26. The following attributes represent data about a movie copy at a video rental store. Each movie is identified by a movie number and has a title and information about the director and the studio that produced the movie. Each movie has one or several characters, and there is exactly one actor playing the role of each of the characters (but one actor can play multiple roles in each of the movies). A video store has multiple copies of the same movie, and the store differentiates copies with a movie copy number, which is unique within a single movie but not unique between different movies. Each movie copy has a rental status and return date; in addition, each copy has a type (VHS, DVD, or Blu-ray). The rental price depends on the movie and the copy type, but the price is the same for all copies of the same type. The attributes are as follows:

Movie Nbr, Title, Director ID, Director Name, Studio ID, Studio Name, Studio Location, Studio CEO, Character, Actor ID, Name, Movie Copy Nbr, Movie Copy Type, Movie Rental Price, Copy Rental Status, Copy Return Date

A sample data set regarding a movie would be as follows (the data in the curly brackets are character/actor data, in this case for four different characters):

567, "It's a Wonderful Life", 25, "Frank Capra", 234, "Liberty Films", "Hollywood, CA", "Orson Wells", {"George Bailey", 245, "James Stewart" | "Mary Bailey", 236, "Donna Reed" | "Clarence Oddbody", 765, "Henry Travers" | "Henry F. Potter", 325, "Lionel Barrymore" }, 5434, "DVD", 2.95, "Rented", "12/15/2010"

Based on this information,

a. Identify the functional dependencies between the attributes.
b. Identify the reasons why this set of data items is not in 3NF and tell what normal form (if any) it is in.
c. Present the attributes organized into 3NF relations that have been named appropriately.

Field Exercises

1. Interview system designers and database designers at several organizations. Ask them to describe the process they use for logical design. How do they transform their conceptual data models (e.g., E-R diagrams) to relational schema? What is the role of CASE tools in this process? Do they use normalization? If they do, how far in the process do they go, and for what purpose?
2. Obtain a common document such as a sales slip, customer invoice from an auto repair shop, credit card statement, etc. Use the normalization steps (steps 0 through 4) described in this chapter to convert this user view to a set of relations in third normal form. Also draw a relational schema. List several integrity rules that you would recommend to ensure the quality of the data in this application.
3. Using Appendix B as a resource, interview a database analyst/designer to determine whether he or she normalizes relations to higher than 3NF. Why or why not does he or she use normal forms beyond 3NF?
4. Find a form or report from a business organization, possibly a statement, bill, or document you have received. Draw an EER diagram of the data in this form or report. Transform the diagram into a set of 3NF relations.

References

Chouinard, P. 1989. "Supertypes, Subtypes, and DB2." *Database Programming & Design* 2,10 (October): 50–57.

Codd, E. F. 1970. "A Relational Model of Data for Large Shared Data Banks." *Communications of the ACM* 13,6 (June): 77–87.

Codd, E. F. 1990. *The Relational Model for Database Management*, Version 2. Reading, MA: Addison-Wesley.

Date, C. J. 2003. *An Introduction to Database Systems*, 8th ed. Reading, MA: Addison-Wesley.

Dutka, A. F., and H. H. Hanson. 1989. *Fundamentals of Data Normalization*. Reading, MA: Addison-Wesley.

Fleming, C. C., and B. von Halle. 1989. *Handbook of Relational Database Design*. Reading, MA: Addison-Wesley.

Hoberman, S. 2006. "To Surrogate Key or Not." *DM Review* 16,8 (August): 29.

Johnston, T. 2000. "Primary Key Reengineering Projects: The Problem" and "Primary Key Reengineering Projects: The Solution." Available at **www.information-management.com**.

Navathe, S., R. Elmasri, and J. Larson. 1986. "Integrating User Views in Database Design." *Computer* 19,1 (January): 50–62.

Further Reading

Elmasri, R., and S. Navathe. 2006. *Fundamentals of Database Systems*, 5th ed. Menlo Park, CA: Benjamin Cummings.

Hoffer, J. A., J. F. George, and J. S. Valacich. 2010. *Modern Systems Analysis and Design*, 6th ed. Upper Saddle River, NJ: Prentice Hall.

Russell, T., and R. Armstrong. 2002. "13 Reasons Why Normalized Tables Help Your Business." *Database Administrator*, April 20, 2002. Available at **http://searchoracle.techtarget.com/tip/13-reasons-why-normalized-tables-help-your-business**

Storey, V. C. 1991. "Relational Database Design Based on the Entity-Relationship Model." *Data and Knowledge Engineering* 7,1 (November): 47–83.

Web Resources

http://en.wikipedia.org/wiki/Database_normalization Wikipedia entry that provides a thorough explanation of first, second, third, fourth, fifth, and Boyce-Codd normal forms.

www.bkent.net/Doc/simple5.htm Web site that presents a summary paper by William Kent titled "A Simple Guide to Five Normal Forms in Relational Database Theory."

www.stevehoberman.com/challenges.htm Web site where Steve Hoberman, a leading consultant and lecturer on database design, periodically creates database design (conceptual and logical) problems and posts them. These are practical (based on real experiences or questions sent to him) situations that make for nice puzzles to solve.

www.troubleshooters.com/codecorn/norm.htm Web page on normalization on Steve Litt's site that contains various troubleshooting tips for avoiding programming and systems development problems.

CASE

Mountain View Community Hospital

Case Description

You have been introduced to the Mountain View Community Hospital (MVCH) case in the preceding chapters. This chapter continues the case, with special emphasis on logical design for the relational data model. Although the hospital will continue to evaluate newer technology (e.g., object-oriented databases, XML, and XML databases), it is expected that relational technology will continue to dominate its systems development over the next few years.

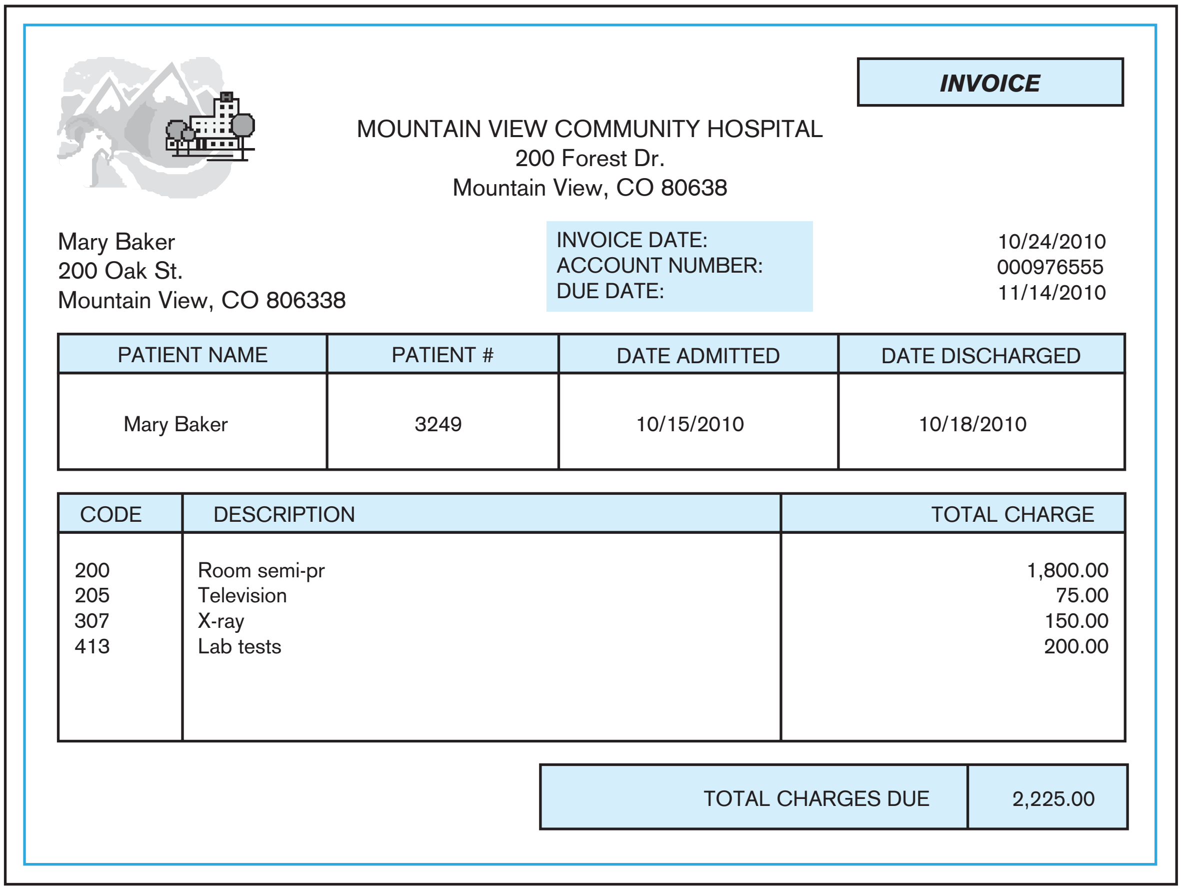
Case Questions

1. Should MVCH continue to use relational technology for its systems development? Why or why not?
2. Should MVCH use normalization in designing its relational databases? Why or why not?
3. Why are entity integrity constraints of importance to the hospital? Based on the case description from previous chapters, which attributes have you encountered that may be null?
4. Why are referential integrity constraints of importance to the hospital?
5. Physicians at MVCH can be uniquely identified by their Social Security number, their license number, their DEA registration number, or hospital-assigned PhysicianID. Which attribute would you suggest using as the primary key for a PHYSICIAN relation? Why? What specific concerns are related to those attributes that you do not recommend be used?
6. The chapter describes the importance of using an *enterprise key*, which is a primary key that is unique across the whole database. Why might this be important in a hospital setting such as MVCH? Explain.
7. Why might you need to revisit and potentially modify the EER model you developed earlier, during the logical design phase?

Case Exercises

1. The assistant administrator at MVCH has asked you to review the data used in the patient billing and accounting systems. Occasional errors have been discovered in patient statements and the patient records maintained by the hospital. As part of this effort, you have selected four user views for analysis. Simplified versions of these views are shown in MVCH Figures 4-1 through 4-4 and described briefly here:

   • ***Patient bill (MVCH Figure 4-1)*** This statement is presented to the patient (or patient representative) when the patient is discharged. Assume that each item on the bill



INVOICE

MOUNTAIN VIEW COMMUNITY HOSPITAL
200 Forest Dr.
Mountain View, CO 80638

Mary Baker
200 Oak St.
Mountain View, CO 806338

INVOICE DATE:	10/24/2010
ACCOUNT NUMBER:	000976555
DUE DATE:	11/14/2010

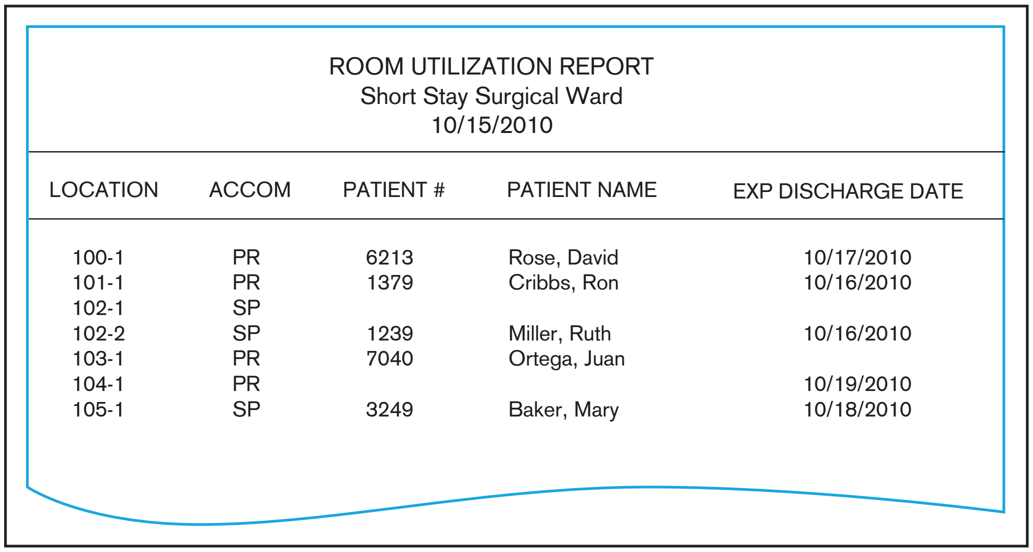
PATIENT NAME	PATIENT #	DATE ADMITTED	DATE DISCHARGED
Mary Baker	3249	10/15/2010	10/18/2010

CODE	DESCRIPTION	TOTAL CHARGE
200	Room semi-pr	1,800.00
205	Television	75.00
307	X-ray	150.00
413	Lab tests	200.00

TOTAL CHARGES DUE	2,225.00

MVCH FIGURE 4-1 Patient bill

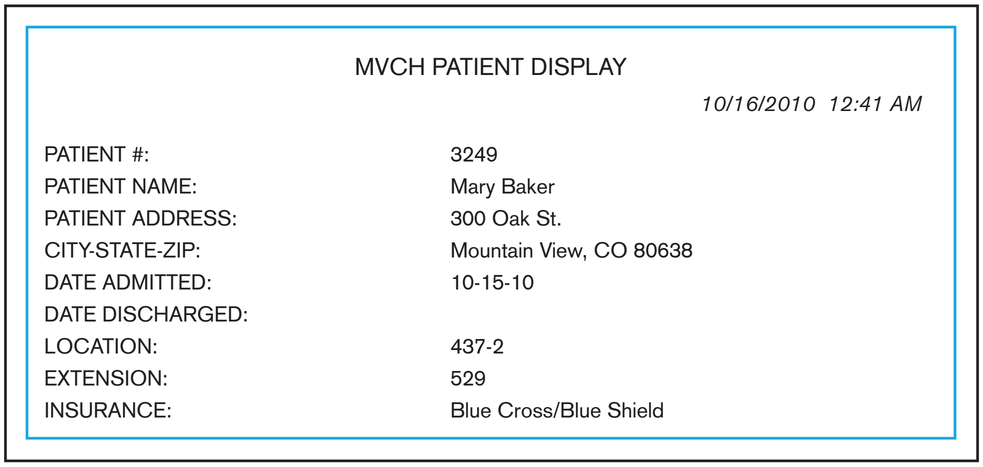
MVCH FIGURE 4-2 Room utilization report (excerpt)



ROOM UTILIZATION REPORT
Short Stay Surgical Ward
10/15/2010

LOCATION	ACCOM	PATIENT #	PATIENT NAME	EXP DISCHARGE DATE
100-1	PR	6213	Rose, David	10/17/2010
101-1	PR	1379	Cribbs, Ron	10/16/2010
102-1	SP			
102-2	SP	1239	Miller, Ruth	10/16/2010
103-1	PR	7040	Ortega, Juan	
104-1	PR			10/19/2010
105-1	SP	3249	Baker, Mary	10/18/2010

MVCH FIGURE 4-3 Patient display



MVCH PATIENT DISPLAY

10/16/2010 12:41 AM

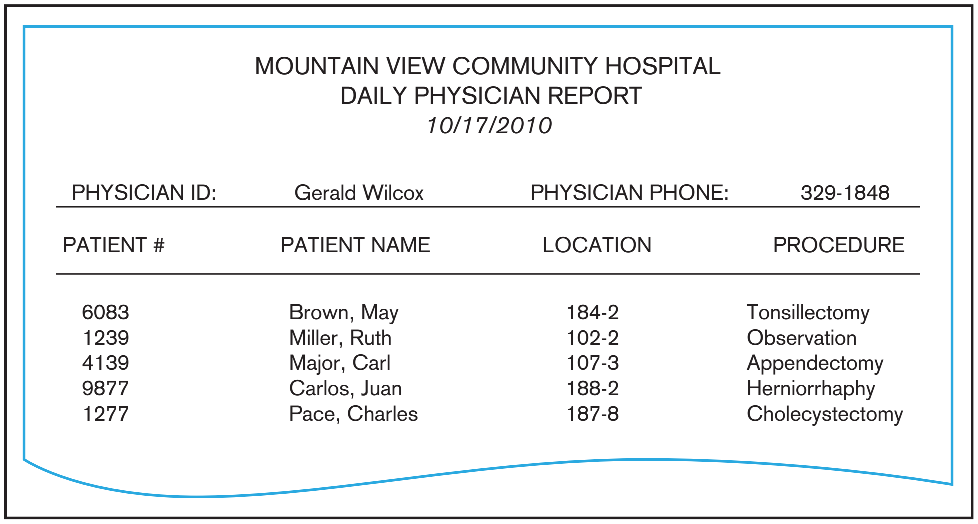
PATIENT #:	3249
PATIENT NAME:	Mary Baker
PATIENT ADDRESS:	300 Oak St.
CITY-STATE-ZIP:	Mountain View, CO 80638
DATE ADMITTED:	10-15-10
DATE DISCHARGED:	
LOCATION:	437-2
EXTENSION:	529
INSURANCE:	Blue Cross/Blue Shield

has a unique description and that the charge for a particular item may vary from one patient to another.

- ***Room utilization report (MVCH Figure 4-2)*** This is a daily report that is distributed to qualified personnel. The information can also be retrieved online by a qualified staff member. It shows the status of each room and bed location in the hospital and is used primarily for scheduling and tracking the utilization of facilities. The Location column in this report records the room number and bed location in the room. The Accom column indicates the type of accommodation (PR = private, SP = semiprivate).
- ***Patient display (MVCH Figure 4-3)*** This display is presented on demand to any qualified doctor, nurse, or other staff member. Assume that for each location there is a unique telephone number.
- ***Daily physician report (MVCH Figure 4-4)*** This report is prepared daily for each physician on the staff of MVCH. It shows the patients who have been treated on that day by the physician and the name of the treatment (or procedure). To simplify the analysis, assume that each patient may receive only one treatment from a given physician on a given day. (We ask you to comment on this assumption later.)

a. Using the normalization steps described in this chapter, develop a set of 3NF relations for each of the four user views.
b. For each user view, draw a relational schema for the 3NF relations you developed in Case Exercise 1a. Be sure to show the functional dependencies and referential integrity constraints for each schema.
c. Merge the relations for the four user views into a single set of 3NF relations, using the guidelines presented in this chapter. Draw a single relational schema for the four user views and show the referential integrity constraints.
d. Suggest any refinements to the design in Case Exercise 1c that would promote data quality and integrity.
e. How would you change your approach to accommodate the rule that a patient may receive multiple treatments from a given physician on a given day?

2. The Multiple Sclerosis (MS) Center, headed by Dr. "Z," has been using a spreadsheet to keep track of information that patients provide upon signing in for a clinic visit. One of the staff members thought it would be better to use a relational database for recording this information and imported the



MOUNTAIN VIEW COMMUNITY HOSPITAL
DAILY PHYSICIAN REPORT
10/17/2010

PHYSICIAN ID: Gerald Wilcox PHYSICIAN PHONE: 329-1848

PATIENT #	PATIENT NAME	LOCATION	PROCEDURE
6083	Brown, May	184-2	Tonsillectomy
1239	Miller, Ruth	102-2	Observation
4139	Major, Carl	107-3	Appendectomy
9877	Carlos, Juan	188-2	Herniorrhaphy
1277	Pace, Charles	187-8	Cholecystectomy

MVCH FIGURE 4-4 Daily physician report (excerpt)

spreadsheet as a table into a Microsoft Access database (MVCH Figure 4-5).

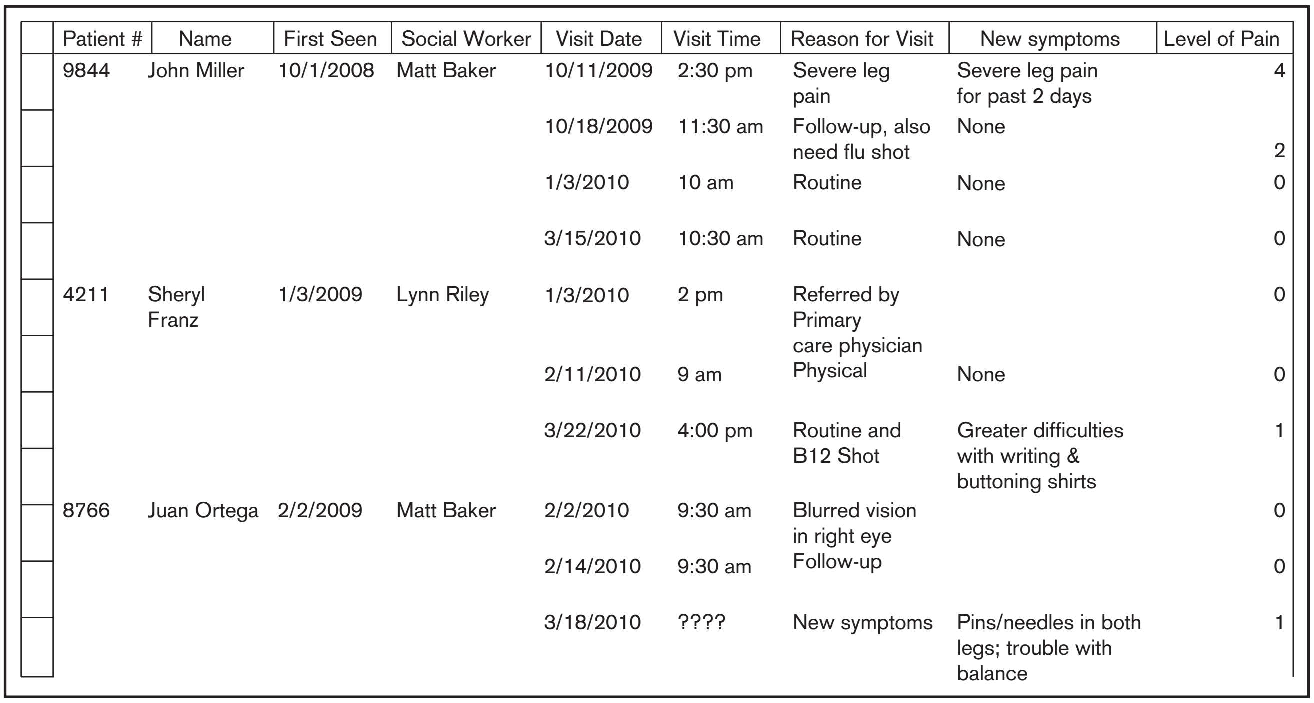
a. What would you suggest as the primary key for this table?
b. Is this table a relation? Why or why not?
c. Can you identify any problems with this table structure? Are there any insertion, deletion, or update anomalies?
d. Diagram the functional dependencies for this table.
e. Using the normalization steps described in this chapter, develop a set of 3NF relations.
f. Using a tool such as Microsoft Visio, draw the relational schema, clearly indicating referential integrity constraints.
g. Write CREATE TABLE commands for all relations in your schema. Make reasonable assumptions concerning the data type for each attribute in each of the relations.

3. Dr. Z in the MS Center is using the MS Clinic Management System from an external vendor to keep track of clinical information regarding his patients. The application uses a relational database. Before seeing a patient, Dr. Z reviews a printout of the worksheet shown in MVCH Figure 4-6.

a. Diagram the functional dependencies for this worksheet and develop a set of 3NF relations for the data on this worksheet.
b. Draw the relational schema and clearly show the referential integrity constraints.
c. Draw your answer to part b using Microsoft Visio (or any other tool specified by your instructor).

Project Assignments

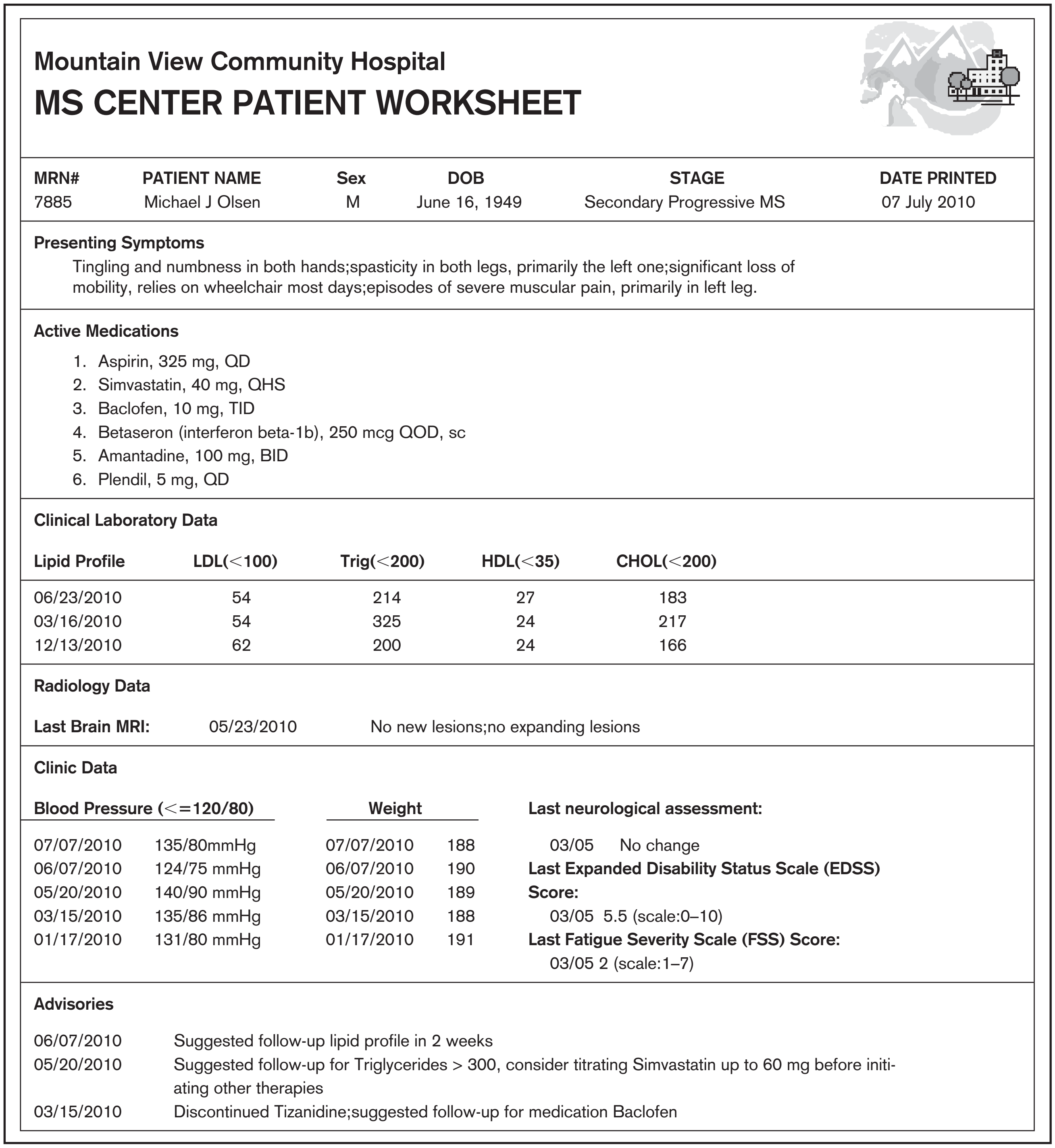
After developing conceptual data models for MVCH's new system and reviewing them with your team and key stakeholders



	Patient #	Name	First Seen	Social Worker	Visit Date	Visit Time	Reason for Visit	New symptoms	Level of Pain
	9844	John Miller	10/1/2008	Matt Baker	10/11/2009	2:30 pm	Severe leg pain	Severe leg pain for past 2 days	4
					10/18/2009	11:30 am	Follow-up, also need flu shot	None	2
					1/3/2010	10 am	Routine	None	0
					3/15/2010	10:30 am	Routine	None	0
	4211	Sheryl Franz	1/3/2009	Lynn Riley	1/3/2010	2 pm	Referred by Primary care physician		0
					2/11/2010	9 am	Physical	None	0
					3/22/2010	4:00 pm	Routine and B12 Shot	Greater difficulties with writing & buttoning shirts	1
	8766	Juan Ortega	2/2/2009	Matt Baker	2/2/2010	9:30 am	Blurred vision in right eye		0
					2/14/2010	9:30 am	Follow-up		0
					3/18/2010	????	New symptoms	Pins/needles in both legs; trouble with balance	1

MVCH FIGURE 4-5 MS Center patient sign-in data

MVCH FIGURE 4-6 **MS Center patient worksheet**



Mountain View Community Hospital

MS CENTER PATIENT WORKSHEET

MRN#	PATIENT NAME	Sex	DOB	STAGE	DATE PRINTED
7885	Michael J Olsen	M	June 16, 1949	Secondary Progressive MS	07 July 2010

Presenting Symptoms

Tingling and numbness in both hands;spasticity in both legs, primarily the left one;significant loss of mobility, relies on wheelchair most days;episodes of severe muscular pain, primarily in left leg.

Active Medications

1. Aspirin, 325 mg, QD
2. Simvastatin, 40 mg, QHS
3. Baclofen, 10 mg, TID
4. Betaseron (interferon beta-1b), 250 mcg QOD, sc
5. Amantadine, 100 mg, BID
6. Plendil, 5 mg, QD

Clinical Laboratory Data

Lipid Profile	LDL(<100)	Trig(<200)	HDL(<35)	CHOL(<200)
06/23/2010	54	214	27	183
03/16/2010	54	325	24	217
12/13/2010	62	200	24	166

Radiology Data

Last Brain MRI: 05/23/2010 No new lesions;no expanding lesions

Clinic Data

Blood Pressure (<=120/80)		Weight	
07/07/2010	135/80mmHg	07/07/2010	188
06/07/2010	124/75 mmHg	06/07/2010	190
05/20/2010	140/90 mmHg	05/20/2010	189
03/15/2010	135/86 mmHg	03/15/2010	188
01/17/2010	131/80 mmHg	01/17/2010	191

Last neurological assessment:
03/05 No change

Last Expanded Disability Status Scale (EDSS) Score:
03/05 5.5 (scale:0–10)

Last Fatigue Severity Scale (FSS) Score:
03/05 2 (scale:1–7)

Advisories

06/07/2010	Suggested follow-up lipid profile in 2 weeks
05/20/2010	Suggested follow-up for Triglycerides > 300, consider titrating Simvastatin up to 60 mg before initiating other therapies
03/15/2010	Discontinued Tizanidine;suggested follow-up for medication Baclofen

at the hospital, you are ready to move on to the logical design for the relational database. Your next deliverable is the relational schema. You may also have to modify the EER model you created in Chapter 3.

P**1.** Map the EER diagram you developed in Chapter 3 to a relational schema, using the techniques described in this chapter. Be sure to underline all primary keys, include all necessary foreign keys, and clearly indicate referential integrity constraints.

P**2.** Analyze and diagram the functional dependencies in each relation. If any relation is not in 3NF, decompose that relation into 3NF relations, using the steps described in this chapter. Revise the relational schema accordingly.

P**3.** Create enterprise keys for all relations and redefine all relations. Revise the relational schema accordingly.

P**4.** If necessary, revisit and modify the EER model you developed in Chapter 3 and explain the changes you made.