any parts used. The actual repair of a device is the performance of the services necessary to address the problems described by the customer. Completing a repair request may require the performance of many services. Each service can be performed many different times during the repair of different devices, but each service will be performed only once during a given repair request.

- All repairs eventually require the performance of at least one service, but which services will be required may not be known at the time the repair request is made. It is possible for services to be available at PEAR but that have never been required in performing any repair.
- Some services involve only labor activities and no parts are required, but most services require the replacement of one or more parts. The quantity of each part required in the performance of each service should also be recorded. For each part, the part number, part description, quantity in stock, and cost is recorded in the system. The cost indicated is the amount that PEAR pays for the part. Some parts may be used in more than one service, but each part is required for at least one service.

10. Luxury-Oriented Scenic Tours (LOST) provides guided tours to groups of visitors to the Washington D.C. area. In recent years, LOST has grown quickly and is having difficulty keeping up with all of the various information needs of the company. The company's operations are as follows:

    - LOST offers many different tours. For each tour, the tour name, approximate length (in hours), and fee charged is needed. Guides are identified by an employee ID, but the system should also record a guide's name, home address, and date of hire. Guides take a test to be qualified to lead specific tours. It is important to know which guides are qualified to lead which tours and the date that they completed the qualification test for each tour. A guide may be qualified to lead many different tours. A tour can have many different qualified guides. New guides may or may not be qualified to lead any tours, just as a new tour may or may not have any qualified guides.
    - Every tour must be designed to visit at least three locations. For each location, a name, type, and official description are kept. Some locations (such as the White House) are visited by more than one tour, while others (such as Arlington Cemetery) are visited by a single tour. All locations are visited by at least one tour. The order in which the tour visits each location should be tracked as well.
    - When a tour is actually given, that is referred to as an "outing." LOST schedules outings well in advance so they can be advertised and so employees can understand their upcoming work schedules. A tour can have many scheduled outings, although newly designed tours may not have any outings scheduled. Each outing is for a single tour and is scheduled for a particular date and time. All outings must be associated with a tour. All tours at LOST are guided tours, so a guide must be assigned to each outing. Each outing has one and only one guide. Guides are occasionally asked to lead an outing of a tour even if they are not officially qualified to lead that tour. Newly hired guides may not have ever been scheduled to lead any outings. Tourists, called "clients" by LOST, pay to join a scheduled outing. For each client, the name and telephone number are recorded. Clients may sign up to join many different outings, and each outing can have many clients. Information is kept only on clients who have signed up for at least one outing, although newly scheduled outings may not have any clients signed up yet.

a. Create a Crow's Foot notation ERD to support LOST operations.
b. The operations provided state that it is possible for a guide to lead an outing of a tour even if the guide is not officially qualified to lead outings of that tour. Imagine that the business rules instead specified that a guide is never, under any circumstance, allowed to lead an outing unless he or she is qualified to lead outings of that tour. How could the data model in Part a. be modified to enforce this new constraint?

Note

You can use the following cases and additional problems from the Instructor Online Companion as the basis for class projects. These problems illustrate the challenge of translating a description of operations into a set of business rules that will define the components for an ERD you can implement successfully. These problems can also be used as the basis for discussions about the components and contents of a proper description of operations. If you want to create databases that can be successfully implemented, you must learn to separate the generic background material from the details that directly affect database design. You must also keep in mind that many constraints cannot be incorporated into the database design; instead, such constraints are handled by the application software.

Cases

11. The administrators of Tiny College are so pleased with your design and implementation of their student registration and tracking system that they want you to expand the design to include the database for their motor vehicle pool. A brief description of operations follows:

    - Faculty members may use the vehicles owned by Tiny College for officially sanctioned travel. For example, the vehicles may be used by faculty members to travel to off-campus learning centers, to travel to locations at which research papers are presented, to transport students to officially sanctioned locations, and to travel for public service purposes. The vehicles used for such purposes are managed by Tiny College's Travel Far But Slowly (TFBS) Center.
    - Using reservation forms, each department can reserve vehicles for its faculty, who are responsible for filling out the appropriate trip completion form at the end of a trip. The reservation form includes the expected departure date, vehicle type required, destination, and name of the authorized faculty member. The faculty member who picks up a vehicle must sign a checkout form to log out the vehicle and pick up a trip completion form. (The TFBS employee who releases the vehicle for use also signs the checkout form.) The faculty member's trip completion form includes the faculty member's identification code, the vehicle's identification, the odometer readings at the start and end of the trip, maintenance complaints (if any), gallons of fuel purchased (if any), and the Tiny College credit card number used to pay for the fuel. If fuel is purchased, the credit card receipt must be stapled to the trip completion form. Upon receipt of the trip completion form, the faculty member's department is billed at a mileage rate based on the vehicle type used: sedan, station wagon, panel truck, minivan, or minibus. (*Hint:* Do *not* use more entities than are necessary. Remember the difference between attributes and entities!)

- All vehicle maintenance is performed by TFBS. Each time a vehicle requires maintenance, a maintenance log entry is completed on a prenumbered maintenance log form. The maintenance log form includes the vehicle identification, brief description of the type of maintenance required, initial log entry date, date the maintenance was completed, and name of the mechanic who released the vehicle back into service. (Only mechanics who have an inspection authorization may release a vehicle back into service.)
- As soon as the log form has been initiated, the log form's number is transferred to a maintenance detail form; the log form's number is also forwarded to the parts department manager, who fills out a parts usage form on which the maintenance log number is recorded. The maintenance detail form contains separate lines for each maintenance item performed, for the parts used, and for identification of the mechanic who performed the maintenance. When all maintenance items have been completed, the maintenance detail form is stapled to the maintenance log form, the maintenance log form's completion date is filled out, and the mechanic who releases the vehicle back into service signs the form. The stapled forms are then filed, to be used later as the source for various maintenance reports.
- TFBS maintains a parts inventory, including oil, oil filters, air filters, and belts of various types. The parts inventory is checked daily to monitor parts usage and to reorder parts that reach the "minimum quantity on hand" level. To track parts usage, the parts manager requires each mechanic to sign out the parts that are used to perform each vehicle's maintenance; the parts manager records the maintenance log number under which the part is used.
- Each month TFBS issues a set of reports. The reports include the mileage driven by vehicle, by department, and by faculty members within a department. In addition, various revenue reports are generated by vehicle and department. A detailed parts usage report is also filed each month. Finally, a vehicle maintenance summary is created each month.

Given that brief summary of operations, draw the appropriate (and fully labeled) ERD. Use the Crow's foot methodology to indicate entities, relationships, connectivities, and participations.

12. During peak periods, Temporary Employment Corporation (TEC) places temporary workers in companies. TEC's manager gives you the following description of the business:
    - TEC has a file of candidates who are willing to work.
    - Any candidate who has worked before has a specific job history. (Naturally, no job history exists if the candidate has never worked.) Each time the candidate works, one additional job history record is created.
    - Each candidate has earned several qualifications. Each qualification may be earned by more than one candidate. (For example, more than one candidate may have earned a Bachelor of Business Administration degree or a Microsoft Network Certification, and clearly a candidate may have earned both a BBA and a Microsoft Network Certification.)
    - TEC offers courses to help candidates improve their qualifications.
    - Every course develops one specific qualification; however, TEC does not offer a course for every qualification. Some qualifications are developed through multiple courses.
    - Some courses cover advanced topics that require specific qualifications as prerequisites. Some courses cover basic topics that do not require any prerequisite

qualifications. A course can have several prerequisites. A qualification can be a prerequisite for more than one course.

- Courses are taught during training sessions. A training session is the presentation of a single course. Over time, TEC will offer many training sessions for each course; however, new courses may not have any training sessions scheduled right away.
- Candidates can pay a fee to attend a training session. A training session can accommodate several candidates, although new training sessions will not have any candidates registered at first.
- TEC also has a list of companies that request temporaries.
- Each time a company requests a temporary employee, TEC makes an entry in the Openings folder. That folder contains an opening number, a company name, required qualifications, a starting date, an anticipated ending date, and hourly pay.
- Each opening requires only one specific or main qualification.
- When a candidate matches the qualification, the job is assigned, and an entry is made in the Placement Record folder. The folder contains such information as an opening number, candidate number, and total hours worked. In addition, an entry is made in the job history for the candidate.
- An opening can be filled by many candidates, and a candidate can fill many openings.
- TEC uses special codes to describe a candidate's qualifications for an opening. The list of codes is shown in Table P4.12.

TABLE P4.12

CODE	DESCRIPTION
SEC-45	Secretarial work; candidate must type at least 45 words per minute
SEC-60	Secretarial work; candidate must type at least 60 words per minute
CLERK	General clerking work
PRG-VB	Programmer, Visual Basic
PRG-C++	Programmer, C++
DBA-ORA	Database Administrator, Oracle
DBA-DB2	Database Administrator, IBM DB2
DBA-SQLSERV	Database Administrator, MS SQL Server
SYS-1	Systems Analyst, level 1
SYS-2	Systems Analyst, level 2
NW-NOV	Network Administrator, Novell experience
WD-CF	Web Developer, ColdFusion

TEC's management wants to keep track of the following entities:

COMPANY, OPENING, QUALIFICATION, CANDIDATE, JOB_HISTORY, PLACEMENT, COURSE, and SESSION. Given that information, do the following:

a. Draw the Crow's Foot ERDs for this enterprise.

b. Identify all necessary relationships.

c. Identify the connectivity for each relationship.

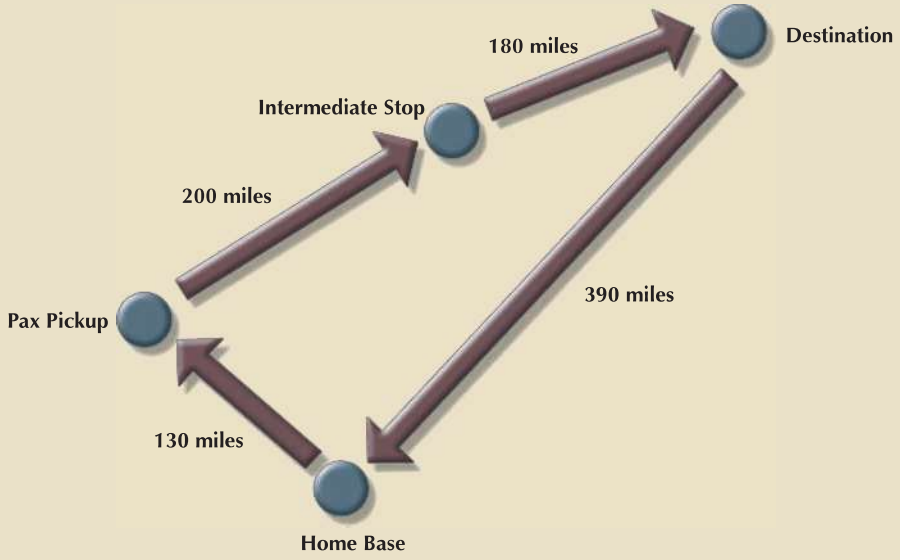
d. Identify the mandatory and optional dependencies for the relationships.

e. Resolve all M:N relationships.

13. Use the following description of the operations of the RC_Charter2 Company to complete this exercise.

- The RC_Charter2 Company operates a fleet of aircraft under the Federal Air Regulations (FAR) Part 135 (air taxi or charter) certificate, enforced by the FAA. The aircraft are available for air taxi (charter) operations within the United States and Canada.
- Charter companies provide so-called unscheduled operations—that is, charter flights take place only after a customer reserves the use of an aircraft at a designated date and time to fly to one or more designated destinations; the aircraft transports passengers, cargo, or some combination of passengers and cargo. Of course, a customer can reserve many different charter trips during any time frame. However, for billing purposes, each charter trip is reserved by one and only one customer. Some of RC_Charter2's customers do not use the company's charter operations; instead, they purchase fuel, use maintenance services, or use other RC_Charter2 services. However, this database design will focus on the charter operations only.
- Each charter trip yields revenue for the RC_Charter2 Company. This revenue is generated by the charges a customer pays upon the completion of a flight. The charter flight charges are a function of aircraft model used, distance flown, waiting time, special customer requirements, and crew expenses. The distance flown charges are computed by multiplying the round-trip miles by the model's charge per mile. Round-trip miles are based on the actual navigational path flown. The sample route traced in Figure P4.13 illustrates the procedure. Note that the number of round-trip miles is calculated to be 130 + 200 + 180 + 390 = 900.

FIGURE P4.13 ROUND-TRIP MILE DETERMINATION



- Depending on whether a customer has RC_Charter2 credit authorization, the customer may do the following:
  a. Pay the entire charter bill upon the completion of the charter flight.
  b. Pay a part of the charter bill and charge the remainder to the account. The charge amount may not exceed the available credit.

c. Charge the entire charter bill to the account. The charge amount may not exceed the available credit.

d. Customers may pay all or part of the existing balance for previous charter trips. Such payments may be made at any time and are not necessarily tied to a specific charter trip. The charter mileage charge includes the expense of the pilot(s) and other crew required by FAR 135. However, if customers request *additional* crew *not* required by FAR 135, those customers are charged for the crew members on an hourly basis. The hourly crew-member charge is based on each crew member's qualifications.

e. The database must be able to handle crew assignments. Each charter trip requires the use of an aircraft, and a crew flies each aircraft. The smaller, piston-engine charter aircraft require a crew consisting of only a single pilot. All jets and other aircraft that have a gross takeoff weight of at least 12,500 pounds require a pilot and a copilot, while some of the larger aircraft used to transport passengers may require flight attendants as part of the crew. Some of the older aircraft require the assignment of a flight engineer, and larger cargo-carrying aircraft require the assignment of a loadmaster. In short, a crew can consist of more than one person, and not all crew members are pilots.

f. The charter flight's aircraft waiting charges are computed by multiplying the hours waited by the model's hourly waiting charge. Crew expenses are limited to meals, lodging, and ground transportation.

The RC_Charter2 database must be designed to generate a monthly summary of all charter trips, expenses, and revenues derived from the charter records. Such records are based on the data that each pilot in command is required to record for each charter trip: trip date(s) and time(s), destination(s), aircraft number, pilot data and other crew data, distance flown, fuel usage, and other data pertinent to the charter flight. Such charter data is then used to generate monthly reports that detail revenue and operating cost information for customers, aircraft, and pilots. All pilots and other crew members are RC_Charter2 Company employees; that is, the company does not use contract pilots and crew.

FAR Part 135 operations are conducted under a strict set of requirements that govern the licensing and training of crew members. For example, pilots must have earned either a commercial license or an Airline Transport Pilot (ATP) license. Both licenses require appropriate ratings, which are specific competency requirements. For example, consider the following:

- To operate a multiengine aircraft designed for takeoffs and landings on land only, the appropriate rating is MEL, or Multiengine Landplane. When a multiengine aircraft can take off and land on water, the appropriate rating is MES, or Multiengine Seaplane.
- The instrument rating is based on a demonstrated ability to conduct all flight operations with sole reference to cockpit instrumentation. The instrument rating is required to operate an aircraft under Instrument Meteorological Conditions (IMC), and all such operations are governed under FAR-specified Instrument Flight Rules (IFR). In contrast, operations conducted under "good weather" or *visual* flight conditions are based on the FAR Visual Flight Rules (VFR).
- The type rating is required for all aircraft with a takeoff weight of more than 12,500 pounds or for aircraft that are purely jet-powered. If an aircraft uses jet engines to drive propellers, that aircraft is said to be turboprop-powered. A turboprop—that

is, a turbo-propeller-powered aircraft—does not require a type rating unless it meets the 12,500-pound weight limitation.

- Although pilot licenses and ratings are not time limited, exercising the privilege of the license and ratings under Part 135 requires both *a current medical certificate and a current Part 135 checkride*. The following distinctions are important:
  a. The medical certificate may be Class I or Class II. The Class I medical is more stringent than the Class II, and it must be renewed every six months. The Class II medical must be renewed yearly. If the Class I medical is not renewed during the six-month period, it automatically reverts to a Class II certificate. If the Class II medical is not renewed within the specified period, it automatically reverts to a Class III medical, which is not valid for commercial flight operations.
  b. A Part 135 checkride is a practical flight examination that must be successfully completed every six months. The checkride includes all flight maneuvers and procedures specified in Part 135.

Nonpilot crew members must also have the proper certificates to meet specific job requirements. For example, loadmasters need an appropriate certificate, as do flight attendants. Crew members such as loadmasters and flight attendants may be required in operations that involve large aircraft with a takeoff weight of more than 12,500 pounds and more than 19 passengers; these crew members are also required to pass a written and practical exam periodically. The RC_Charter2 Company is required to keep a complete record of all test types, dates, and results for each crew member, as well as examination dates for pilot medical certificates.

In addition, all flight crew members are required to submit to periodic drug testing; the results must be tracked as well. Note that nonpilot crew members are not required to take pilot-specific tests such as Part 135 checkrides, nor are pilots required to take crew tests such as loadmaster and flight attendant practical exams. However, many crew members have licenses and certifications in several areas. For example, a pilot may have an ATP and a loadmaster certificate. If that pilot is assigned to be a loadmaster on a given charter flight, the loadmaster certificate is required. Similarly, a flight attendant may have earned a commercial pilot's license. Sample data formats are shown in Table P4.13.

Pilots and other crew members must receive recurrency training appropriate to their work assignments. Recurrency training is based on an FAA-approved curriculum that is job specific. For example, pilot recurrency training includes a review of all applicable Part 135 flight rules and regulations, weather data interpretation, company flight operations requirements, and specified flight procedures. The RC_Charter2 Company is required to keep a complete record of all recurrency training for each crew member subject to the training.

The RC_Charter2 Company is required to maintain a detailed record of all crew credentials and all training mandated by Part 135. The company must keep a complete record of each requirement and of all compliance data.

To conduct a charter flight, the company must have a properly maintained aircraft available. A pilot who meets all of the FAA's licensing and currency requirements must fly the aircraft as Pilot in Command (PIC). For aircraft that are powered by piston engines or turboprops and have a gross takeoff weight under 12,500 pounds, single-pilot operations are permitted under Part 135 as long as a properly maintained autopilot is available. However, even if FAR Part 135 permits single-pilot operations, many customers require the presence of a copilot who is capable of conducting the flight operations under Part 135.

TABLE P4.13

PART A TESTS		
TEST CODE	TEST DESCRIPTION	TEST FREQUENCY
1	Part 135 Flight Check	6 months
2	Medical, Class I	6 months
3	Medical, Class II	12 months
4	Loadmaster Practical	12 months
5	Flight Attendant Practical	12 months
6	Drug test	Random
7	Operations, written exam	6 months

PART B RESULTS			
EMPLOYEE	TEST CODE	TEST DATE	TEST RESULT
101	1	12-Nov-15	Pass-1
103	6	23-Dec-15	Pass-1
112	4	23-Dec-15	Pass-2
103	7	11-Jan-16	Pass-1
112	7	16-Jan-16	Pass-1
101	7	16-Jan-16	Pass-1
101	6	11-Feb-16	Pass-2
125	2	15-Feb-16	Pass-1

PART C LICENSES AND CERTIFICATIONS	
LICENSE OR CERTIFICATE	LICENSE OR CERTIFICATE DESCRIPTION
ATP	Airline Transport Pilot
Comm	Commercial license
Med-1	Medical certificate, Class I
Med-2	Medical certificate, Class II
Instr	Instrument rating
MEL	Multiengine Land aircraft rating
LM	Loadmaster
FA	Flight Attendant

EMPLOYEE	LICENSE OR CERTIFICATE	DATE EARNED
101	Comm	12-Nov-93
101	Instr	28-Jun-94
101	MEL	9-Aug-94
103	Comm	21-Dec-95
112	FA	23-Jun-02
103	Instr	18-Jan-96
112	LM	27-Nov-05

The RC_Charter2 operations manager anticipates the lease of turbojet-powered aircraft, which are required to have a crew consisting of a pilot and copilot. Both the pilot and copilot must meet the same Part 135 licensing, ratings, and training requirements.

The company also leases larger aircraft that exceed the 12,500-pound gross takeoff weight. Those aircraft might carry enough passengers to require the presence of one or more flight attendants. If those aircraft carry cargo that weighs more than 12,500 pounds, a loadmaster must be assigned as a crew member to supervise the loading and securing of the cargo. *The database must be designed to meet the anticipated capability for additional charter crew assignments.*

a. Given this incomplete description of operations, write all applicable business rules to establish entities, relationships, optionalities, connectivities, and cardinalities. (*Hint:* Use the following five business rules as examples, and write the remaining business rules in the same format.) A customer may request many charter trips.
   - Each charter trip is requested by only one customer.
   - Some customers have not yet requested a charter trip.
   - An employee may be assigned to serve as a crew member on many charter trips.
   - Each charter trip may have many employees assigned to serve as crew members.

b. Draw the fully labeled and implementable Crow's Foot ERD based on the business rules you wrote in Part a. of this problem. Include all entities, relationships, optionalities, connectivities, and cardinalities.

Chapter 5

Advanced Data Modeling

In this chapter, you will learn:

- About the extended entity relationship (EER) model
- How entity clusters are used to represent multiple entities and relationships
- The characteristics of good primary keys and how to select them
- How to use flexible solutions for special data-modeling cases

Preview

In the previous two chapters, you learned how to use entity relationship diagrams (ERDs) to properly create a data model. In this chapter, you will learn about the extended entity relationship (EER) model. The EER model builds on ER concepts and adds support for entity supertypes, subtypes, and entity clustering.

Most current database implementations are based on relational databases. Because the relational model uses keys to create associations among tables, it is essential to learn the characteristics of good primary keys and how to select them. Selecting a good primary key is too important to be left to chance, so this chapter covers the critical aspects of primary key identification and placement.

Focusing on practical database design, this chapter also illustrates some special design cases that highlight the importance of flexible designs, which can be adapted to meet the demands of changing data and information requirements. Data modeling is a vital step in the development of databases that in turn provides a good foundation for successful application development. Remember that good database applications cannot be based on bad database designs, and no amount of outstanding coding can overcome the limitations of poor database design.

Data Files and Available Formats

	MS Access	Oracle	MS SQL	My SQL		MS Access	Oracle	MS SQL	My SQL
CH05_AirCo	✓	✓	✓	✓	CH05_GCSdata	✓	✓	✓	✓
CH05_TinyCollege	✓	✓	✓	✓					

Data Files Available on cengagebrain.com

Note

The extended entity relationship model discussed in this chapter includes advanced data modeling constructs such as specialization hierarchies. Although Microsoft Visio 2010 and earlier versions handled these constructs neatly, newer versions of Visio starting with Microsoft Visio 2013 removed support for many database modeling activities, including specialization hierarchies.

5-1 The Extended Entity Relationship Model

As the complexity of the data structures being modeled has increased and as application software requirements have become more stringent, the need to capture more information in the data model has increased. The **extended entity relationship model (EERM)**, sometimes referred to as the enhanced entity relationship model, is the result of adding more semantic constructs to the original entity relationship (ER) model. As you might expect, a diagram that uses the EERM is called an **EER diagram (EERD)**. In the following sections, you will learn about the main EER model constructs—entity supertypes, entity subtypes, and entity clustering—and see how they are represented in ERDs/EERDs.

5-1a Entity Supertypes and Subtypes

Because most employees possess a wide range of skills and special qualifications, data modelers must find a variety of ways to group employees based on their characteristics. For instance, a retail company could group employees as salaried and hourly, while a university could group employees as faculty, staff, and administrators.

The grouping of employees into various *types* provides two important benefits:

- It avoids unnecessary nulls in attributes when some employees have characteristics that are not shared by other employees.
- It enables a particular employee type to participate in relationships that are unique to that employee type.

extended entity relationship model (EERM)
Sometimes referred to as the enhanced entity relationship model; the result of adding more semantic constructs, such as entity supertypes, entity subtypes, and entity clustering, to the original entity relationship (ER) model.

EER diagram (EERD)
The entity relationship diagram resulting from the application of extended entity relationship concepts that provide additional semantic content in the ER model.

To illustrate those benefits, you will explore the case of an aviation business that employs pilots, mechanics, secretaries, accountants, database managers, and many other types of employees. Figure 5.1 illustrates how pilots share certain characteristics with other employees, such as a last name (EMP_LNAME) and hire date (EMP_HIRE_DATE). On the other hand, many pilot characteristics are not shared by other employees. For example, unlike other employees, pilots must meet special requirements such as flight hour restrictions, flight checks, and periodic training. Therefore, if all employee characteristics and special qualifications were stored in a single EMPLOYEE entity, you would have a lot of nulls or you would have to create a lot of needless dummy entries. In this case, special pilot characteristics such as EMP_LICENSE, EMP_RATINGS, and EMP_MED_TYPE will generate nulls for employees who are not pilots. In addition, pilots participate in some relationships that are unique to their qualifications. For example, not all employees can fly airplanes; only employees who are pilots can participate in the "employee flies airplane" relationship.

Based on the preceding discussion, you would correctly deduce that the PILOT entity stores only attributes that are unique to pilots, and that the EMPLOYEE entity stores attributes that are common to all employees. Based on that hierarchy, you can conclude

FIGURE 5.1 NULLS CREATED BY UNIQUE ATTRIBUTES

Database name: Ch05_AirCo

EMP_NUM	EMP_LNAME	EMP_FNAME	EMP_INITIAL	EMP_LICENSE	EMP_RATINGS	EMP_MED_TYPE	EMP_HIRE_DATE
100	Kolmycz	Xavier	T				15-Mar-88
101	Lewis	Marcos		ATP	SEL/MEL/Instr/CFII	1	25-Apr-89
102	Vandam	Jean					20-Dec-93
103	Jones	Victoria	R				28-Aug-03
104	Lange	Edith		ATP	SEL/MEL/Instr	1	20-Oct-97
105	Williams	Gabriel	U	COM	SEL/MEL/Instr/CFI	2	08-Nov-97
106	Duzak	Mario		COM	SEL/MEL/Instr	2	05-Jan-04
107	Diante	Venite	L				02-Jul-97
108	Wiesenbach	Joni					18-Nov-95
109	Travis	Brett	T	COM	SEL/MEL/SES/Instr/CFII	1	14-Apr-01
110	Genkazi	Stan					01-Dec-03

that PILOT is a *subtype* of EMPLOYEE, and that EMPLOYEE is the *supertype* of PILOT. In modeling terms, an **entity supertype** is a generic entity type that is related to one or more **entity subtypes**. The entity supertype contains common characteristics, and the entity subtypes each contain their own unique characteristics.

Two criteria help the designer determine when to use subtypes and supertypes:

- There must be different, identifiable kinds or types of the entity in the user's environment.
- The different kinds or types of instances should each have one or more attributes that are unique to that kind or type of instance.

In the preceding example, because pilots meet both criteria of being an identifiable kind of employee and having unique attributes that other employees do not possess, it is appropriate to create PILOT as a subtype of EMPLOYEE. Assume that mechanics and accountants also each have attributes that are unique to them, respectively, and that clerks do not. In that case, MECHANIC and ACCOUNTANT would also be legitimate subtypes of EMPLOYEE because they are identifiable kinds of employees and have unique attributes. CLERK would *not* be an acceptable subtype of EMPLOYEE because it only satisfies one of the criteria—it is an identifiable kind of employee—but none of the attributes are unique to just clerks. In the next section, you will learn how entity supertypes and subtypes are related in a specialization hierarchy.

entity supertype
In a generalization/specialization hierarchy, a generic entity type that contains the common characteristics of entity subtypes.

entity subtype
In a generalization/specialization hierarchy, a subset of an entity supertype. The entity supertype contains the common characteristics and the subtypes contain the unique characteristics of each entity.

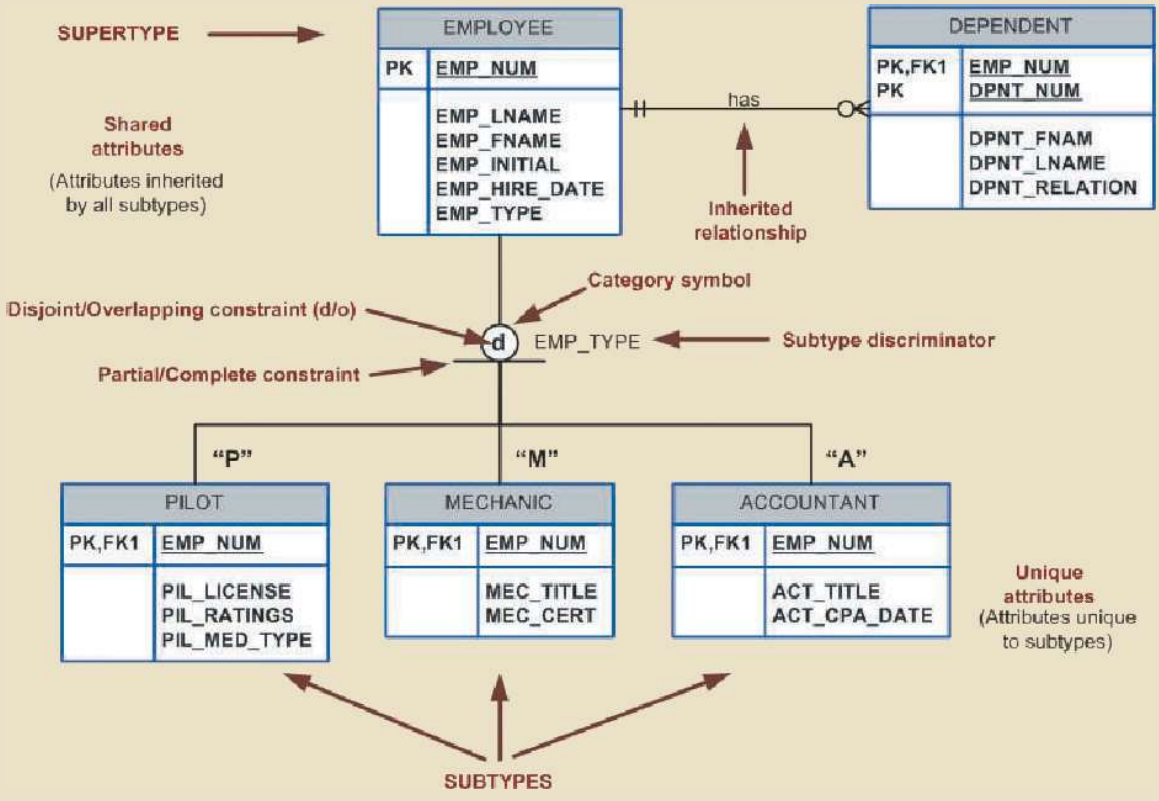
specialization hierarchy
A hierarchy based on the top-down process of identifying lower-level, more specific entity subtypes from a higher-level entity supertype. Specialization is based on grouping unique characteristics and relationships of the subtypes.

5-1b Specialization Hierarchy

Entity supertypes and subtypes are organized in a **specialization hierarchy**, which depicts the arrangement of higher-level entity supertypes (parent entities) and lower-level entity subtypes (child entities). Figure 5.2 shows the specialization hierarchy formed by an EMPLOYEE supertype and three entity subtypes—PILOT, MECHANIC, and ACCOUNTANT. The specialization hierarchy reflects the 1:1 relationship between EMPLOYEE and its subtypes. For example, a PILOT subtype occurrence is related to one instance of the EMPLOYEE supertype, and a MECHANIC subtype occurrence is related to one instance of the EMPLOYEE supertype. The terminology and symbols in Figure 5.2 are explained throughout this chapter.

The relationships depicted within the specialization hierarchy are sometimes described in terms of "is-a" relationships. For example, a pilot *is an* employee, a mechanic *is an* employee, and an accountant *is an* employee. It is important to understand that within a specialization hierarchy, a subtype can exist only within the context of a supertype, and every subtype can have only one supertype to which it is directly related. However, a

FIGURE 5.2 A SPECIALIZATION HIERARCHY



Online Content

This chapter covers only specialization hierarchies. The EER model also supports specialization *lattices*, in which a subtype can have multiple parents (supertypes). However, those concepts are better covered under the object-oriented model in Appendix G, Object-Oriented Databases. The appendix is available at *www.cengagebrain.com*.

specialization hierarchy can have many levels of supertype/subtype relationships—that is, you can have a specialization hierarchy in which a supertype has many subtypes. In turn, one of the subtypes is the supertype to other lower-level subtypes.

As you can see in Figure 5.2, the arrangement of entity supertypes and subtypes in a specialization hierarchy is more than a cosmetic convenience. Specialization hierarchies enable the data model to capture additional semantic content (meaning) into the ERD. A specialization hierarchy provides the means to:

- Support attribute inheritance.
- Define a special supertype attribute known as the subtype discriminator.
- Define disjoint/overlapping constraints and complete/partial constraints.

The following sections cover such characteristics and constraints in more detail.

5-1c Inheritance

The property of **inheritance** enables an entity subtype to inherit the attributes and relationships of the supertype. As discussed earlier, a supertype contains attributes that are common to all of its subtypes. In contrast, subtypes contain only the attributes that are unique to the subtype. For example, Figure 5.2 illustrates that pilots, mechanics, and accountants all inherit the employee number, last name, first name, middle initial, and hire date from the EMPLOYEE entity. However, Figure 5.2 also illustrates that pilots have unique attributes; the same is true for mechanics and accountants. *One important inheritance characteristic is that all entity subtypes inherit their primary key attribute from their supertype.* Note in Figure 5.2 that the EMP_NUM attribute is the primary key for each of the subtypes.

inheritance
In the EERD, the property that enables an entity subtype to inherit the attributes and relationships of the entity supertype.

At the implementation level, the supertype and its subtype(s) depicted in the specialization hierarchy maintain a 1:1 relationship. For example, the specialization hierarchy

lets you replace the undesirable EMPLOYEE table structure in Figure 5.1 with two tables—one representing the supertype EMPLOYEE and the other representing the subtype PILOT. (See Figure 5.3.)

FIGURE 5.3 THE EMPLOYEE-PILOT SUPERTYPE-SUBTYPE RELATIONSHIP

Database name: Ch05_AirCo

Table name: EMPLOYEE

EMP_NUM	EMP_LNAME	EMP_FNAME	EMP_INITIAL	EMP_HIRE_DATE	EMP_TYPE
100	Kolmycz	Xavier	T	15-Mar-88	
101	Lewis	Marcos		25-Apr-89	P
102	Vandam	Jean		20-Dec-93	A
103	Jones	Victoria	R	28-Aug-03	
104	Lange	Edith		20-Oct-97	P
105	Williams	Gabriel	U	08-Nov-97	P
106	Duzak	Mario		05-Jan-04	P
107	Diante	Venite	L	02-Jul-97	M
108	Wiesenbach	Joni		18-Nov-95	M
109	Travis	Brett	T	14-Apr-01	P
110	Genkazi	Stan		01-Dec-03	A

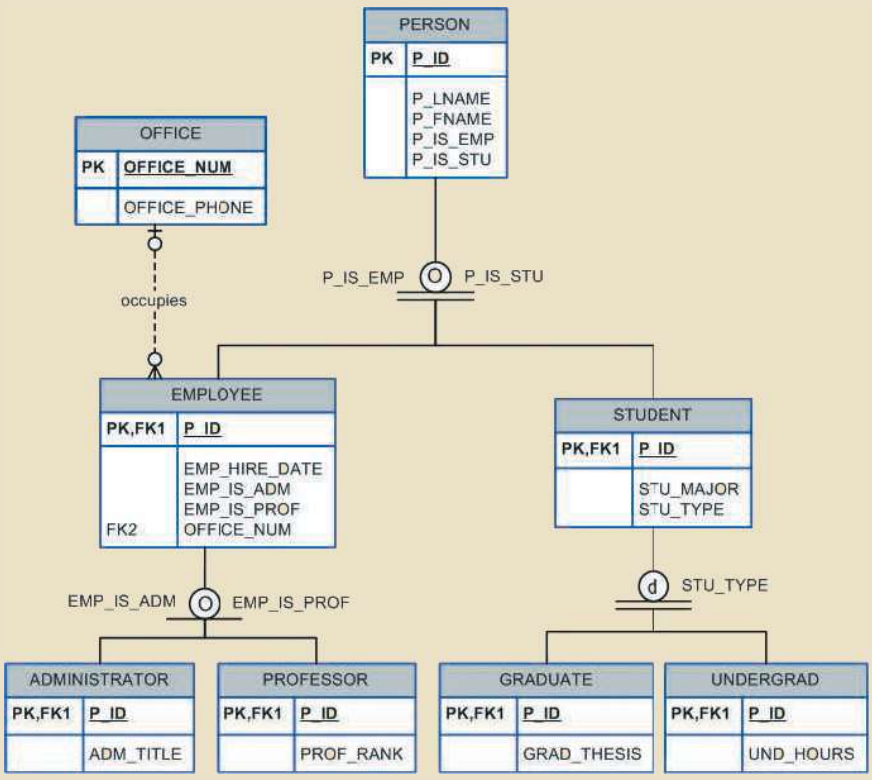
Table name: PILOT

EMP_NUM	PIL_LICENSE	PIL_RATINGS	PIL_MED_TYPE
101	ATP	SEL/MEL/Instr/CFII	1
104	ATP	SEL/MEL/Instr	1
105	COM	SEL/MEL/Instr/CFI	2
106	COM	SEL/MEL/Instr	2
109	COM	SEL/MEL/SES/Instr/CFII	1

Entity subtypes inherit all relationships in which the supertype entity participates. For example, Figure 5.2 shows the EMPLOYEE entity supertype participating in a 1:M relationship with a DEPENDENT entity. Through inheritance, all subtypes also participate in that relationship. In specialization hierarchies with multiple levels of supertype and subtypes, a lower-level subtype inherits all of the attributes and relationships from all of its upper-level supertypes.

Inheriting the relationships of their supertypes does not mean that subtypes cannot have relationships of their own. Figure 5.4 illustrates a 1:M relationship between EMPLOYEE, a subtype of PERSON, and OFFICE. Because only employees and no other type of person will ever have an office within this system, the relationship is modeled with the subtype directly.

FIGURE 5.4 SPECIALIZATION HIERARCHY WITH OVERLAPPING SUBTYPES



Online Content

For a tutorial on using Visio 2010 to create a specialization hierarchy, see Appendix A, Designing Databases with Visio Professional: A Tutorial, at *www.cengagebrain.com*.

5-1d Subtype Discriminator

A **subtype discriminator** is the attribute in the supertype entity that determines to which subtype the supertype occurrence is related. In Figure 5.2, the subtype discriminator is the employee type (EMP_TYPE).

It is common practice to show the subtype discriminator and its value for each subtype in the ER diagram, as shown in Figure 5.2. However, not all ER modeling tools follow that practice. For example, Microsoft Visio shows the subtype discriminator but not its value. In Figure 5.2, a text tool was used to manually add the discriminator value above the entity subtype, close to the connector line. Using Figure 5.2 as your guide, note that the supertype is related to a PILOT subtype if the EMP_TYPE has a value of "P." If the EMP_TYPE value is "M," the supertype is related to a MECHANIC subtype. If the EMP_TYPE value is "A," the supertype is related to the ACCOUNTANT subtype.

Note that the default comparison condition for the subtype discriminator attribute is the equality comparison. However, in some situations the subtype discriminator is not necessarily based on an equality comparison. For example, based on business requirements, you might create two new pilot subtypes: pilot-in-command (PIC)-qualified and copilot-qualified only. A PIC-qualified pilot must have more than 1,500 PIC flight hours. In this case, the subtype discriminator would be "Flight_Hours," and the criteria would be > 1,500 or <= 1,500, respectively.

Note

In Visio 2010, you select the subtype discriminator when creating a category by using the Category shape from the available shapes. The Category shape is a small circle with a horizontal line underneath that connects the supertype to its subtypes. Visio 2013 does not support specialization hierarchy.

5-1e Disjoint and Overlapping Constraints

subtype discriminator
The attribute in the supertype entity that determines to which entity subtype each supertype occurrence is related.

disjoint subtype
In a specialization hierarchy, a unique and nonoverlapping subtype entity set.

nonoverlapping subtype
See *disjoint subtype*.

overlapping subtype
In a specialization hierarchy, a condition in which each entity instance (row) of the supertype can appear in more than one subtype.

An entity supertype can have disjoint or overlapping entity subtypes. In the aviation example, an employee can be a pilot, a mechanic, or an accountant. Assume that one of the business rules dictates that an employee cannot belong to more than one subtype at a time; that is, an employee cannot be a pilot and a mechanic at the same time. **Disjoint subtypes**, also known as **nonoverlapping subtypes**, are subtypes that contain a *unique* subset of the supertype entity set; in other words, each entity instance of the supertype can appear in only one of the subtypes. For example, in Figure 5.2, an employee (supertype) who is a pilot (subtype) can appear only in the PILOT subtype, not in any of the other subtypes. In an ERD, such disjoint subtypes are indicated by the letter *d* inside the category shape.

On the other hand, if the business rule specifies that employees can have multiple classifications, the EMPLOYEE supertype may contain *overlapping* job classification subtypes. **Overlapping subtypes** are subtypes that contain nonunique subsets of the supertype entity set; that is, each entity instance of the supertype may appear in more than one subtype. For example, in a university environment, a person may be an employee, a student, or both. In turn, an employee may be a professor as well as an administrator. Because an employee may also be a student, STUDENT and EMPLOYEE are overlapping subtypes of the supertype PERSON, just as PROFESSOR and ADMINISTRATOR are overlapping subtypes of the supertype EMPLOYEE. Figure 5.4 illustrates overlapping subtypes with the letter *o* inside the category shape.

It is common practice to show disjoint and overlapping symbols in the ERD. (See Figures 5.2 and 5.4.) However, not all ER modeling tools follow that practice. For example, by default, Visio shows only the subtype discriminator (using the Category shape), but not the disjoint and overlapping symbols. The Visio text tool was used to manually add the *d* and *o* symbols in Figures 5.2 and 5.4.

Note

Alternative notations exist for representing disjoint and overlapping subtypes. For example, Toby J. Teorey popularized the use of G and Gs to indicate disjoint and overlapping subtypes.

completeness constraint
A constraint that specifies whether each entity supertype occurrence must also be a member of at least one subtype. The completeness constraint can be partial or total.

As you learned earlier in this section, the implementation of disjoint subtypes is based on the value of the subtype discriminator attribute in the supertype. However, *implementing* overlapping subtypes requires the use of one discriminator attribute for each subtype. For example, in the case of the Tiny College database design in Chapter 4, Entity Relationship (ER) Modeling, a professor can also be an administrator. Therefore, the EMPLOYEE supertype would have the subtype discriminator attributes and values shown in Table 5.1.

TABLE 5.1

DISCRIMINATOR ATTRIBUTES WITH OVERLAPPING SUBTYPES

DISCRIMINATOR ATTRIBUTES		COMMENT
PROFESSOR	ADMINISTRATOR	
Y	N	The Employee is a member of the Professor subtype.
N	Y	The Employee is a member of the Administrator subtype.
Y	Y	The Employee is both a Professor and an Administrator.

5-1f Completeness Constraint

The **completeness constraint** specifies whether each entity supertype occurrence must also be a member of at least one subtype. The completeness constraint can be partial or total. **Partial completeness** means that not every supertype occurrence is a member of a subtype; some supertype occurrences may not be members of any subtype. **Total completeness** means that every supertype occurrence must be a member of at least one subtype.

The ERDs in Figures 5.2 and 5.4 represent the completeness constraint based on the Visio Category shape. A single horizontal line under the circle represents a partial completeness constraint; a double horizontal line under the circle represents a total completeness constraint.

partial completeness
In a generalization/specialization hierarchy, a condition in which some supertype occurrences might not be members of any subtype.

total completeness
In a generalization/specialization hierarchy, a condition in which every supertype occurrence must be a member of at least one subtype.

Note

Alternative notations exist to represent the completeness constraint. For example, some notations use a single line (partial) or double line (total) to connect the supertype to the Category shape.

Given the disjoint and overlapping subtypes and completeness constraints, it is possible to have the specialization hierarchy constraint scenarios shown in Table 5.2.

TABLE 5.2

SPECIALIZATION HIERARCHY CONSTRAINT SCENARIOS

TYPE	DISJOINT CONSTRAINT	OVERLAPPING CONSTRAINT
Partial 	Supertype has optional subtypes. Subtype discriminator can be null. Subtype sets are unique.	Supertype has optional subtypes. Subtype discriminators can be null. Subtype sets are not unique.
Total 	Every supertype occurrence is a member of only one subtype. Subtype discriminator cannot be null. Subtype sets are unique.	Every supertype occurrence is a member of at least one subtype. Subtype discriminators cannot be null. Subtype sets are not unique.

5-1g Specialization and Generalization

You can use various approaches to develop entity supertypes and subtypes. For example, you can first identify a regular entity, and then identify all entity subtypes based on their distinguishing characteristics. You can also start by identifying multiple entity types and then later extract the common characteristics of those entities to create a higher-level supertype entity.

Specialization is the top-down process of identifying lower-level, more specific entity subtypes from a higher-level entity supertype. Specialization is based on grouping the unique characteristics and relationships of the subtypes. In the aviation example, you used specialization to identify multiple entity subtypes from the original employee supertype. **Generalization** is the bottom-up process of identifying a higher-level, more generic entity supertype from lower-level entity subtypes. Generalization is based on grouping the common characteristics and relationships of the subtypes. For example, you might identify multiple types of musical instruments: piano, violin, and guitar. Using the generalization approach, you could identify a "string instrument" entity supertype to hold the common characteristics of the multiple subtypes.

specialization
In a specialization hierarchy, the grouping of unique attributes into a subtype entity.

generalization
In a specialization hierarchy, the grouping of common attributes into a supertype entity.

entity cluster
A "virtual" entity type used to represent multiple entities and relationships in the ERD. An entity cluster is formed by combining multiple interrelated entities into a single abstract entity object. An entity cluster is considered "virtual" or "abstract" because it is not actually an entity in the final ERD.

5-2 Entity Clustering

Developing an ER diagram entails the discovery of possibly hundreds of entity types and their respective relationships. Generally, the data modeler will develop an initial ERD that contains a few entities. As the design approaches completion, the ERD will contain hundreds of entities and relationships that crowd the diagram to the point of making it unreadable and inefficient as a communication tool. In those cases, you can use entity clusters to minimize the number of entities shown in the ERD.

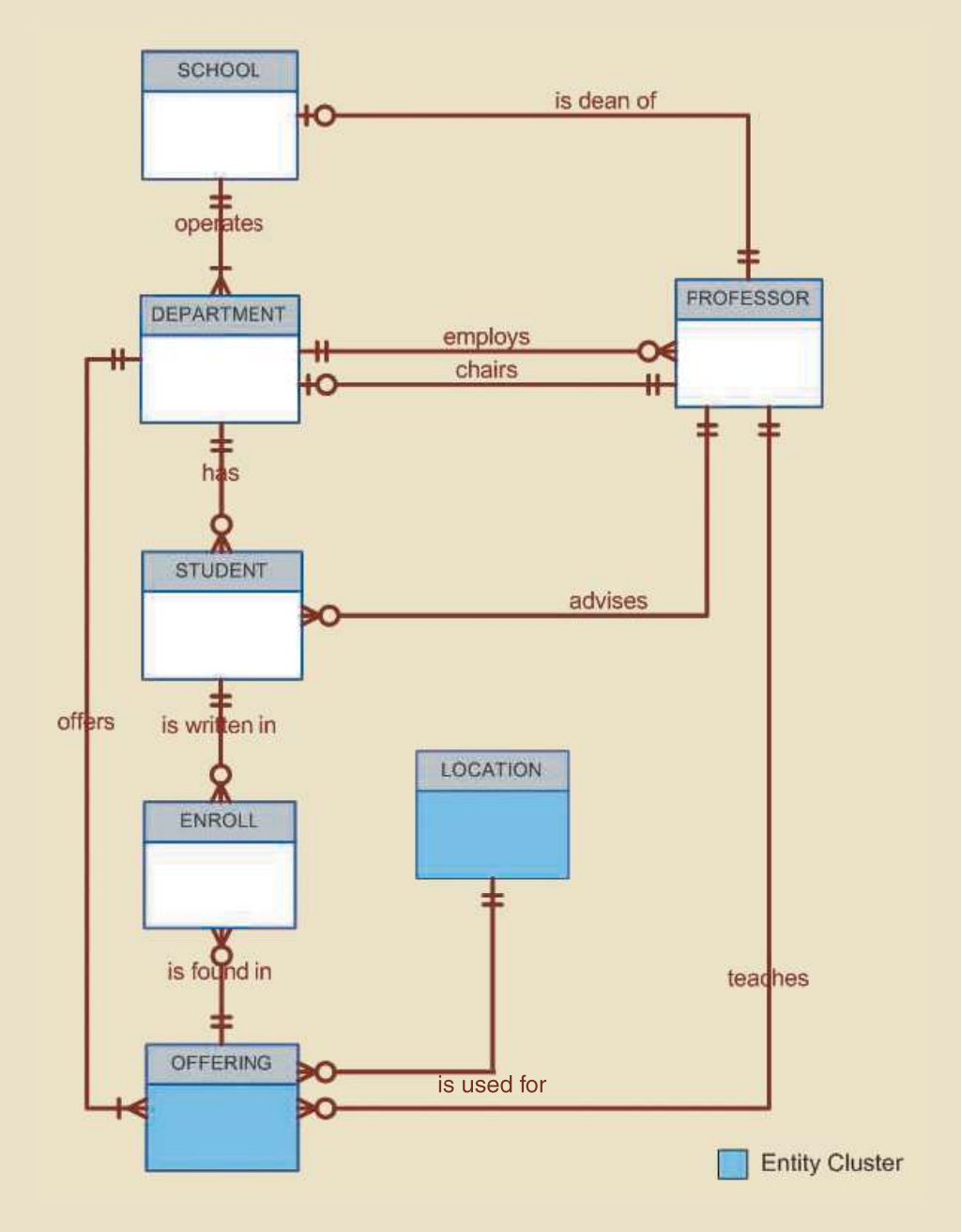
An **entity cluster** is a "virtual" entity type used to represent multiple entities and relationships in the ERD. An entity cluster is formed by combining multiple interrelated entities into a single, abstract entity object. An entity cluster is considered "virtual" or "abstract" in the sense that it is not actually an entity in the final ERD. Instead, it is a temporary entity used to represent multiple entities and relationships, with the purpose of simplifying the ERD and thus enhancing its readability.

Figure 5.5 illustrates the use of entity clusters based on the Tiny College example in Chapter 4. Note that the ERD contains two entity clusters:

- OFFERING, which groups the SEMESTER, COURSE, and CLASS entities and relationships
- LOCATION, which groups the ROOM and BUILDING entities and relationships

Note also that the ERD in Figure 5.5 does not show attributes for the entities. When using entity clusters, the key attributes of the combined entities are no longer available. Without the key attributes, primary key inheritance rules change. In turn, the change in the inheritance rules can have undesirable consequences, such as changes in relationships—from identifying to nonidentifying or vice versa—and the loss of foreign key attributes from some entities. To eliminate those problems, the general rule is to *avoid the display of attributes when entity clusters are used.*

FIGURE 5.5 TINY COLLEGE ERD USING ENTITY CLUSTERS



5-3 Entity Integrity: Selecting Primary Keys

Arguably, the most important characteristic of an entity is its primary key (a single attribute or some combination of attributes), which uniquely identifies each entity instance.

The primary key's function is to guarantee entity integrity. Furthermore, primary keys and foreign keys work together to implement relationships in the relational model. Therefore, the importance of properly selecting the primary key has a direct bearing on the efficiency and effectiveness of database implementation.

5-3a Natural Keys and Primary Keys

The concept of a unique identifier is commonly encountered in the real world. For example, you use class or section numbers to register for classes, invoice numbers to identify specific invoices, and account numbers to identify credit cards. Those examples illustrate natural identifiers or keys. A **natural key** or **natural identifier** is a real-world, generally accepted identifier used to distinguish—that is, uniquely identify—real-world objects. As its name implies, a natural key is familiar to end users and forms part of their day-to-day business vocabulary.

Usually, if an entity *has* a natural identifier, a data modeler uses it as the primary key of the entity being modeled. Generally, most natural keys make acceptable primary key identifiers. The next section presents some basic guidelines for selecting primary keys.

5-3b Primary Key Guidelines

A primary key is the attribute or combination of attributes that uniquely identifies entity instances in an entity set. However, can the primary key be based on, for example, 12 attributes? And just how long can a primary key be? In previous examples, why was EMP_NUM selected as a primary key of EMPLOYEE and not a combination of EMP_LNAME, EMP_FNAME, EMP_INITIAL, and EMP_DOB? Can a single, 256-byte text attribute be a good primary key? There is no single answer to those questions, but database experts have built a body of practice over the years. This section examines that body of documented practices.

First, you should understand the function of a primary key. Its main function is to uniquely identify an entity instance or row within a table. In particular, given a primary key value—that is, the determinant—the relational model can determine the value of all dependent attributes that "describe" the entity. Note that identification and description are separate semantic constructs in the model. *The function of the primary key is to guarantee entity integrity, not to "describe" the entity.*

Second, primary keys and foreign keys are used to implement relationships among entities. However, the implementation of such relationships is done mostly behind the scenes, hidden from end users. In the real world, end users identify objects based on the characteristics they know about the objects. For example, when shopping at a grocery store, you select products by taking them from a display shelf and reading the labels, not by looking at the stock number. It is wise for database applications to mimic the human selection process as much as possible. Therefore, database applications should let the end user choose among multiple descriptive narratives of different objects, while using primary key values behind the scenes. Keeping those concepts in mind, look at Table 5.3, which summarizes desirable primary key characteristics.

natural key (natural identifier)
A generally accepted identifier for real-world objects. As its name implies, a natural key is familiar to end users and forms part of their day-to-day business vocabulary.

5-3c When To Use Composite Primary Keys

In the previous section, you learned about the desirable characteristics of primary keys. For example, you learned that the primary key should use the minimum number of attributes possible. However, that does *not* mean that composite primary keys are not permitted in a model. In fact, composite primary keys are particularly useful in two cases:

TABLE 5.3

DESIRABLE PRIMARY KEY CHARACTERISTICS

PK CHARACTERISTIC	RATIONALE
Unique values	The PK must uniquely identify each entity instance. A primary key must be able to guarantee unique values. It cannot contain nulls.
Nonintelligent	The PK should not have embedded semantic meaning other than to uniquely identify each entity instance. An attribute with embedded semantic meaning is probably better used as a descriptive characteristic of the entity than as an identifier. For example, a student ID of 650973 would be preferred over *Smith, Martha L.* as a primary key identifier.
No change over time	If an attribute has semantic meaning, it might be subject to updates, which is why names do not make good primary keys. If *Vickie Smith* is the primary key, what happens if she changes her name when she gets married? If a primary key is subject to change, the foreign key values must be updated, thus adding to the database work load. Furthermore, changing a primary key value means that you are basically changing the identity of an entity. In short, the PK should be permanent and unchangeable.
Preferably single-attribute	A primary key should have the minimum number of attributes possible (irreducible). Single-attribute primary keys are desirable but not required. Single-attribute primary keys simplify the implementation of foreign keys. Having multiple-attribute primary keys can cause primary keys of related entities to grow through the possible addition of many attributes, thus adding to the database workload and making (application) coding more cumbersome.
Preferably numeric	Unique values can be better managed when they are numeric, because the database can use internal routines to implement a counter-style attribute that automatically increments values with the addition of each new row. In fact, most database systems include the ability to use special constructs, such as Autonumber in Microsoft Access, sequence in Oracle, or uniqueidentifier in MS SQL Server to support self-incrementing primary key attributes.
Security-compliant	The selected primary key must not be composed of any attribute(s) that might be considered a security risk or violation. For example, using a Social Security number as a PK in an EMPLOYEE table is not a good idea.

- As identifiers of composite entities, in which each primary key combination is allowed only once in the M:N relationship
- As identifiers of weak entities, in which the weak entity has a strong identifying relationship with the parent entity

To illustrate the first case, assume that you have a STUDENT entity set and a CLASS entity set. In addition, assume that those two sets are related in an M:N relationship via an ENROLL entity set, in which each student/class combination may appear only once in the composite entity. Figure 5.6 shows the ERD to represent such a relationship.

As shown in Figure 5.6, the composite primary key automatically provides the benefit of ensuring that there cannot be duplicate values—that is, it ensures that the same student cannot enroll more than once in the same class.

In the second case, a weak entity in a strong identifying relationship with a parent entity is normally used to represent one of two situations:

1. *A real-world object that is existence-dependent on another real-world object.* Such objects are distinguishable in the real world. A dependent and an employee are two separate people who exist independently of each other. However, such objects can exist in the model only when they relate to each other in a strong identifying relationship. For example, the relationship between EMPLOYEE and DEPENDENT is one of existence dependency, in which the primary key of the dependent entity is a composite key that contains the key of the parent entity.

FIGURE 5.6 THE M:N RELATIONSHIP BETWEEN STUDENT AND CLASS

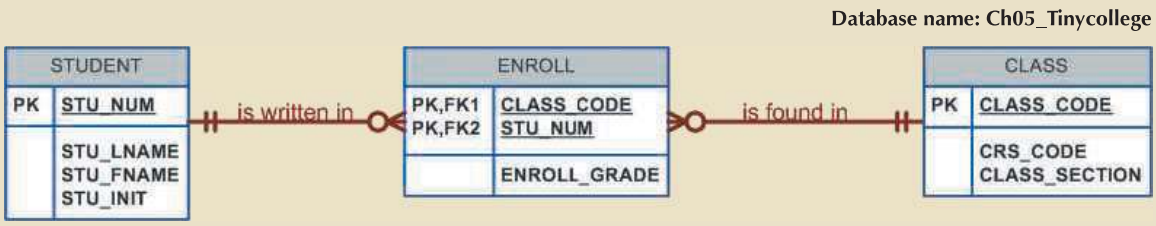


Table name: STUDENT
(first four fields)

STU_NUM	STU_LNAME	STU_FNAME	STU_INIT
321452	Bowser	William	C
324257	Smithson	Anne	K
324258	Brewer	Juliette	
324269	Oblonski	Walter	H
324273	Smith	John	D
324274	Katinga	Raphael	P
324291	Robertson	Gerald	T
324299	Smith	John	B

Table name: ENROLL

CLASS_CODE	STU_NUM	ENROLL_GRADE
10014	321452	C
10014	324257	B
10018	321452	A
10018	324257	B
10021	321452	C
10021	324257	C

Table name: CLASS
(first three fields)

CLASS_CODE	CRS_CODE	CLASS_SECTION
10012	ACCT-211	1
10013	ACCT-211	2
10014	ACCT-211	3
10015	ACCT-212	1
10016	ACCT-212	2
10017	CIS-220	1
10018	CIS-220	2
10019	CIS-220	3
10020	CIS-420	1
10021	QM-261	1
10022	QM-261	2
10023	QM-362	1
10024	QM-362	2
10025	MATH-243	1

2. *A real-world object that is represented in the data model as two separate entities in a strong identifying relationship.* For example, the real-world invoice object is represented by two entities in a data model: INVOICE and LINE. Clearly, the LINE entity does not exist in the real world as an independent object, but as part of an INVOICE.

In both situations, having a strong identifying relationship ensures that the dependent entity can exist only when it is related to the parent entity. In summary, the selection of a composite primary key for composite and weak entity types provides benefits that enhance the integrity and consistency of the model.

5-3d When To Use Surrogate Primary Keys

In some instances a primary key doesn't exist in the real world or the existing natural key might not be a suitable primary key. In these cases, it is standard practice to create a surrogate key. A **surrogate key** is a primary key created by the database designer to simplify the identification of entity instances. The surrogate key has no meaning in the user's environment—it exists only to distinguish one entity instance from another (just like any other primary key). One practical advantage of a surrogate key is that because it has no intrinsic meaning, values for it can be generated by the DBMS to ensure that unique values are always provided.

For example, consider the case of a park recreation facility that rents rooms for small parties. The manager of the facility keeps track of all events, using a folder with the format shown in Table 5.4.

Given the data shown in Table 5.4, you would model the EVENT entity as follows:

EVENT (DATE, TIME_START, TIME_END, ROOM, EVENT_NAME, PARTY_OF)

surrogate key
A system-assigned primary key, generally numeric and auto-incremented.

What primary key would you suggest? In this case, there is no simple natural key that could be used as a primary key in the model. Based on the primary key concepts you learned in previous chapters, you might suggest one of these options:

TABLE 5.4

DATA USED TO KEEP TRACK OF EVENTS

DATE	TIME_START	TIME_END	ROOM	EVENT_NAME	PARTY_OF
6/17/2016	11:00a.m.	2:00p.m.	Allure	Burton Wedding	60
6/17/2016	11:00a.m.	2:00p.m.	Bonanza	Adams Office	12
6/17/2016	3:00p.m.	5:30p.m.	Allure	Smith Family	15
6/17/2016	3:30p.m.	5:30p.m.	Bonanza	Adams Office	12
6/18/2016	1:00p.m.	3:00p.m.	Bonanza	Boy Scouts	33
6/18/2016	11:00a.m.	2:00p.m.	Allure	March of Dimes	25
6/18/2016	11:00a.m.	12:30p.m.	Bonanza	Smith Family	12

(**DATE**, **TIME_START**, **ROOM**) or (**DATE**, **TIME_END**, **ROOM**)

Assume that you select the composite primary key (**DATE**, **TIME_START**, **ROOM**) for the EVENT entity. Next, you determine that one EVENT may use many RESOURCEs (such as tables, projectors, PCs, and stands) and that the same RESOURCE may be used for many EVENTs. The RESOURCE entity would be represented by the following attributes:

RESOURCE (**RSC_ID**, RSC_DESCRIPTION, RSC_TYPE, RSC_QTY, RSC_PRICE)

Given the business rules, the M:N relationship between RESOURCE and EVENT would be represented via the EVNTRSC composite entity with a composite primary key as follows:

EVNTRSC (**DATE**, **TIME_START**, **ROOM**, **RSC_ID**, QTY_USED)

You now have a lengthy, four-attribute composite primary key. What would happen if the EVNTRSC entity's primary key were inherited by another existence-dependent entity? At this point, you can see that the composite primary key could make the database implementation and program coding unnecessarily complex.

As a data modeler, you probably noticed that the EVENT entity's selected primary key might not fare well, given the primary key guidelines in Table 5.3. In this case, the EVENT entity's selected primary key contains embedded semantic information and is formed by a combination of date, time, and text data columns. In addition, the selected primary key would cause lengthy primary keys for existence-dependent entities. The preferred alternative is to use a numeric, single-attribute surrogate primary key.

Surrogate primary keys are accepted practice in today's complex data environments. They are especially helpful when there is no natural key, when the selected candidate key has embedded semantic contents, or when the selected candidate key is too long or cumbersome. However, there is a trade-off: if you use a surrogate key, you must ensure that the candidate key of the entity in question performs properly through the use of "unique index" and "not null" constraints.

Note

This example shows a case in which entity integrity is maintained but semantic correctness of business rules is not. For example, you could have two events that overlap and whose primary keys are perfectly compliant. The only way to ensure adherence to this type of business rule (two events cannot overlap—occur on the same room at the same time) would be via application programming code.

5-4 Design Cases: Learning Flexible Database Design

Data modeling and database design require skills that are acquired through experience. In turn, experience is acquired through practice—regular and frequent repetition, applying the concepts learned to specific and different design problems. This section presents four special design cases that highlight the importance of flexible designs, proper identification of primary keys, and placement of foreign keys.

Note

In describing the various modeling concepts throughout this book, the focus is on relational models. Also, given the focus on the practical nature of database design, all design issues are addressed with the implementation goal in mind. Therefore, there is no sharp line of demarcation between design and implementation.

At the pure conceptual stage of the design, foreign keys are not part of an ER diagram. The ERD displays only entities and relationships. Entity instances are distinguished by identifiers that may become primary keys. During design, the modeler attempts to understand and define the entities and relationships. Foreign keys are the mechanism through which the relationship designed in an ERD is implemented in a relational model.

5-4a Design Case 1: Implementing 1:1 Relationships

Foreign keys work with primary keys to properly implement relationships in the relational model. The basic rule is very simple: put the primary key of the "one" side (the parent entity) on the "many" side (the dependent entity) as a foreign key. However, where do you place the foreign key when you are working with a 1:1 relationship? For example, take the case of a 1:1 relationship between EMPLOYEE and DEPARTMENT based on the business rule "one EMPLOYEE is the manager of one DEPARTMENT, and one DEPARTMENT is managed by one EMPLOYEE." In that case, there are two options for selecting and placing the foreign key:

1. *Place a foreign key in both entities.* This option is derived from the basic rule you learned in Chapter 4. Place EMP_NUM as a foreign key in DEPARTMENT, and place DEPT_ID as a foreign key in EMPLOYEE. However, this solution is not recommended because it duplicates work, and it could conflict with other existing relationships. (Remember that DEPARTMENT and EMPLOYEE also participate in a 1:M relationship—one department employs many employees.)
2. *Place a foreign key in one of the entities.* In that case, the primary key of one of the two entities appears as a foreign key in the other entity. That is the preferred solution, but a question remains: *which* primary key should be used as a foreign key? The answer is found in Table 5.5, which shows the rationale for selecting the foreign key in a 1:1 relationship based on the relationship properties in the ERD.

Figure 5.7 illustrates the "EMPLOYEE manages DEPARTMENT" relationship. Note that in this case, EMPLOYEE is mandatory to DEPARTMENT. Therefore, EMP_NUM is placed as the foreign key in DEPARTMENT. Alternatively, you might also argue that the "manager" role is played by the EMPLOYEE in the DEPARTMENT.

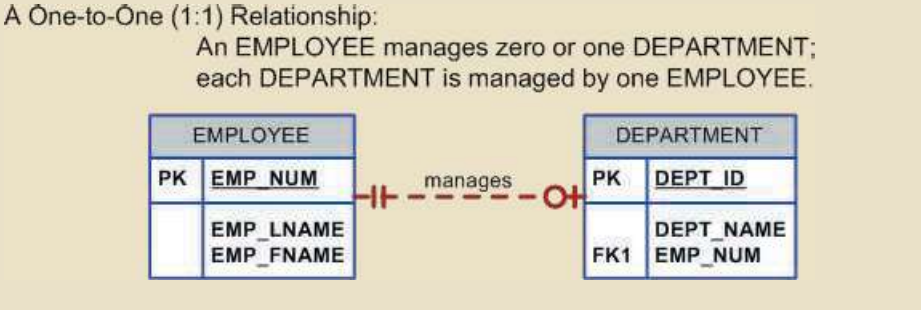
As a designer, you must recognize that 1:1 relationships exist in the real world; therefore, they should be supported in the data model. In fact, a 1:1 relationship is used to

TABLE 5.5

SELECTION OF FOREIGN KEY IN A 1:1 RELATIONSHIP

CASE	ER RELATIONSHIP CONSTRAINTS	ACTION
I	One side is mandatory and the other side is optional.	Place the PK of the entity on the mandatory side in the entity on the optional side as a FK, and make the FK mandatory.
II	Both sides are optional.	Select the FK that causes the fewest nulls, or place the FK in the entity in which the (relationship) role is played.
III	Both sides are mandatory.	See Case II, or consider revising your model to ensure that the two entities do not belong together in a single entity.

FIGURE 5.7 THE 1:1 RELATIONSHIP BETWEEN DEPARTMENT AND EMPLOYEE



ensure that two entity sets are not placed in the same table. In other words, EMPLOYEE and DEPARTMENT are clearly separate and unique entity types that do not belong together in a single entity. If you grouped them together in one entity, what would you name that entity?

5-4b Design Case 2: Maintaining History of Time-Variant Data

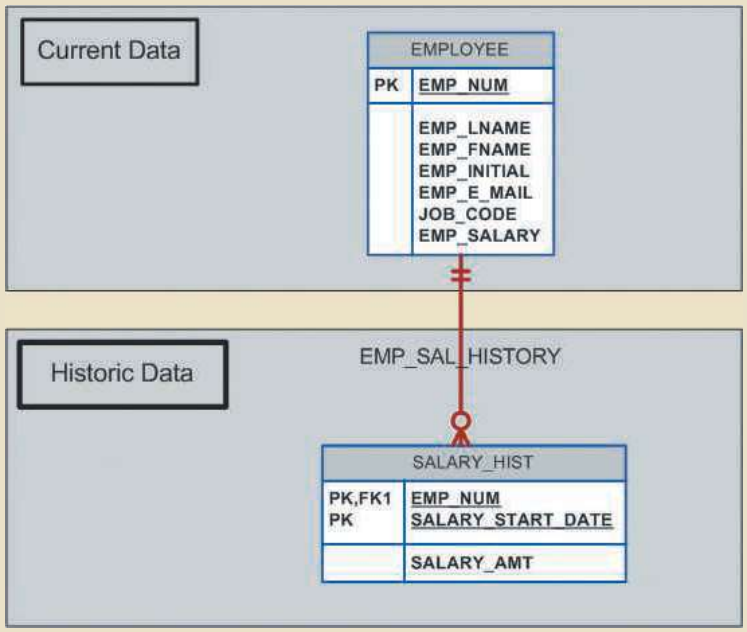
Company managers generally realize that good decision making is based on the information generated through the data stored in databases. Such data reflects both current and past events. Company managers use the data stored in databases to answer questions such as "How do the current company profits compare to those of previous years?" and "What are XYZ product's sales trends?" In other words, the data stored in databases reflects not only current data, but historic data.

Normally, data changes are managed by replacing the existing attribute value with the new value, without regard to the previous value. However, in some situations the history of values for a given attribute must be preserved. From a data-modeling point of view, **time-variant data** refer to data whose values change over time and for which you *must* keep a history of the data changes. You could argue that all data in a database is subject to change over time and is therefore time variant. However, some attribute values, such as your date of birth or your Social Security number, are not time variant. On the other hand, attributes such as your student GPA or your bank account balance are subject to change over time. Sometimes the data changes are externally originated and event driven, such as a product price change. On other occasions, changes are based on well-defined schedules, such as the daily stock quote "open" and "close" values.

time-variant data
Data whose values are a function of time. For example, time-variant data can be seen at work when a company's history of all administrative appointments is tracked.

The storage of time-variant data requires changes in the data model; the type of change depends on the nature of the data. Some time-variant data is equivalent to having a multivalued attribute in your entity. To model this type of time-variant data, you must create a new entity in a 1:M relationship with the original entity. This new entity will contain the new value, the date of the change, and any other attribute that is pertinent to the event being modeled. For example, if you want to track salary histories for each employee, then the EMP_SALARY attribute becomes multivalued, as shown in Figure 5.8. In this case, for each employee, there will be one or more records in the SALARY_HIST entity, which stores the salary amount and the date when the new salary goes into effect.

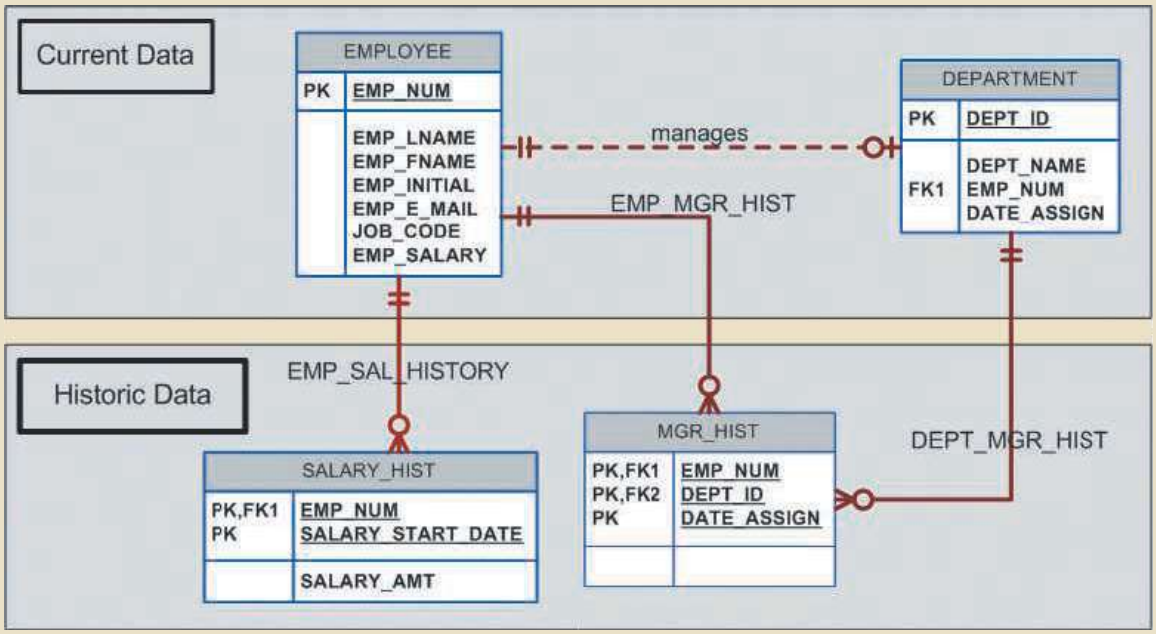
FIGURE 5.8 MAINTAINING SALARY HISTORY



Other time-variant data can turn a 1:M relationship into an M:N relationship. Assume that in addition to employee data, your data model includes data about the different departments in the organization and which employee manages each department. Assuming that each department is managed by only one employee and each employee can manage one department at most, then a 1:1 relationship would exist between EMPLOYEE and DEPARTMENT. This relationship would record the current manager of each department. However, if you want to keep track of the history of all department managers as well as the current manager, you can create the model shown in Figure 5.9.

Note that in Figure 5.9, the MGR_HIST entity has a 1:M relationship with EMPLOYEE and a 1:M relationship with DEPARTMENT to reflect the fact that an employee could be the manager of many different departments over time, and a department could have many different employee managers. Because you are recording time-variant data, you must store the DATE_ASSIGN attribute in the MGR_HIST entity to provide the date that the employee (EMP_NUM) became the department manager. The primary key of MGR_HIST permits the same employee to be the manager of the same department, but on different dates. If that scenario is not the case in your environment—if, for example, an employee is the manager of a department only once—you could make DATE_ASSIGN a nonprime attribute in the MGR_HIST entity.

FIGURE 5.9 MAINTAINING MANAGER HISTORY

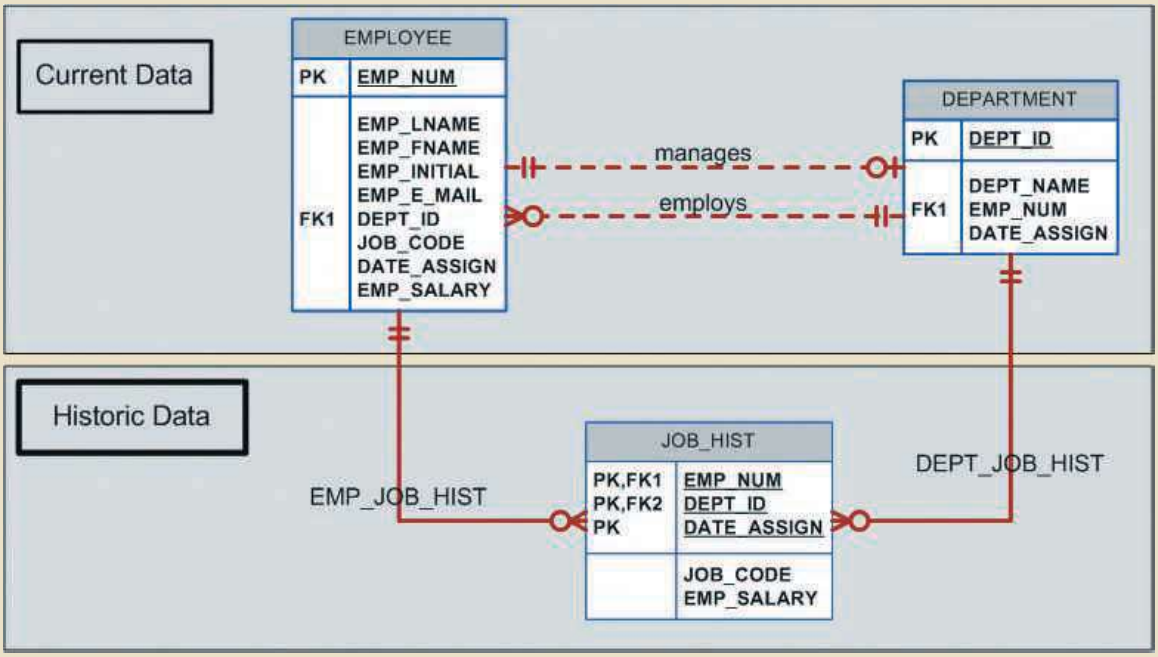


Note in Figure 5.9 that the "manages" relationship is optional in theory and redundant in practice. At any time, you could identify the manager of a department by retrieving the most recent DATE_ASSIGN date from MGR_HIST for a given department. On the other hand, the ERD in Figure 5.9 differentiates between current data and historic data. The *current* manager relationship is implemented by the "manages" relationship between EMPLOYEE and DEPARTMENT. Additionally, the historic data is managed through EMP_MGR_HIST and DEPT_MGR_HIST. The trade-off with that model is that each time a new manager is assigned to a department, there will be two data modifications: one update in the DEPARTMENT entity and one insert in the MGR_HIST entity.

The flexibility of the model proposed in Figure 5.9 becomes more apparent when you add the 1:M "one department employs many employees" relationship. In that case, the PK of the "1" side (DEPT_ID) appears in the "many" side (EMPLOYEE) as a foreign key. Now suppose you would like to keep track of the job history for each of the company's employees—you'd probably want to store the department, the job code, the date assigned, and the salary. To accomplish that task, you could modify the model in Figure 5.9 by adding a JOB_HIST entity. Figure 5.10 shows the use of the new JOB_HIST entity to maintain the employee's history.

Again, it is worth emphasizing that the "manages" and "employs" relationships are theoretically optional and redundant in practice. You can always find out where each employee works by looking at the job history and selecting only the most current data row for each employee. However, as you will discover in Chapter 7, Introduction to Structured Query Language (SQL), and in Chapter 8, Advanced SQL, finding where each employee works is not a trivial task. Therefore, the model represented in Figure 5.10 includes the admittedly redundant but unquestionably useful "manages" and "employs" relationships to separate current data from historic data.

FIGURE 5.10 MAINTAINING JOB HISTORY



5-4c Design Case 3: Fan Traps

Creating a data model requires proper identification of the data relationships among entities. However, due to miscommunication or incomplete understanding of the business rules or processes, it is not uncommon to misidentify relationships among entities. Under those circumstances, the ERD may contain a design trap. A **design trap** occurs when a relationship is improperly or incompletely identified and is therefore represented in a way that is not consistent with the real world. The most common design trap is known as a *fan trap*.

A **fan trap** occurs when you have one entity in two 1:M relationships to other entities, thus producing an association among the other entities that is not expressed in the model. For example, assume that the JCB basketball league has many divisions. Each division has many players, and each division has many teams. Given those "incomplete" business rules, you might create an ERD that looks like the one in Figure 5.11.

design trap
A problem that occurs when a relationship is improperly or incompletely identified and therefore is represented in a way that is not consistent with the real world. The most common design trap is known as a *fan trap*.

fan trap
A design trap that occurs when one entity is in two 1:M relationships with other entities, thus producing an association among the other entities that is not expressed in the model.

As you can see in Figure 5.11, DIVISION is in a 1:M relationship with TEAM and in a 1:M relationship with PLAYER. Although that representation is semantically correct, the relationships are not properly identified. For example, there is no way to identify which players belong to which team. Figure 5.11 also shows a sample instance relationship representation for the ERD. Note that the relationship lines for the DIVISION instances fan out to the TEAM and PLAYER entity instances—thus the "fan trap" label.

Figure 5.12 shows the correct ERD after the fan trap has been eliminated. Note that, in this case, DIVISION is in a 1:M relationship with TEAM. In turn, TEAM is in a 1:M relationship with PLAYER. Figure 5.12 also shows the instance relationship representation after eliminating the fan trap.

Given the design in Figure 5.12, note how easy it is to see which players play for which team. However, to find out which players play in which division, you first need to see what teams belong to each division; then you need to find out which players play on each team. In other words, there is a transitive relationship between DIVISION and PLAYER via the TEAM entity.

FIGURE 5.11 INCORRECT ERD WITH FAN TRAP PROBLEM

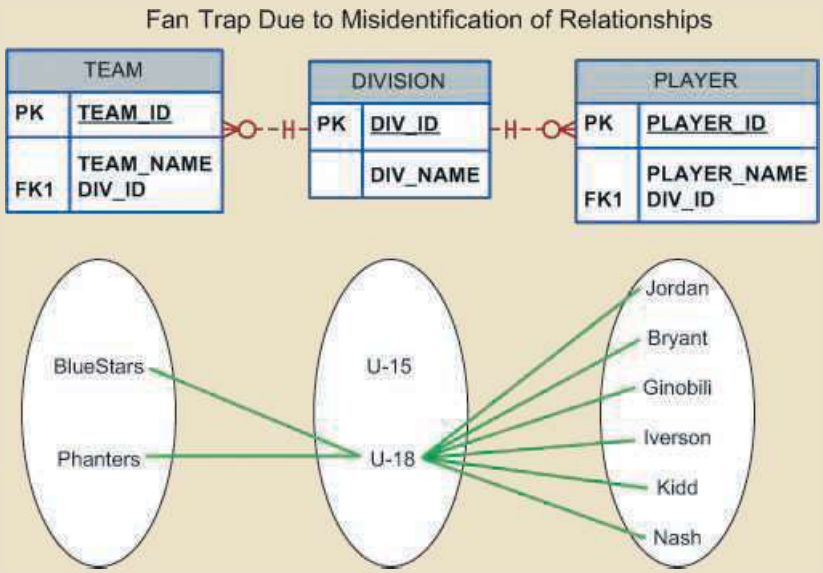
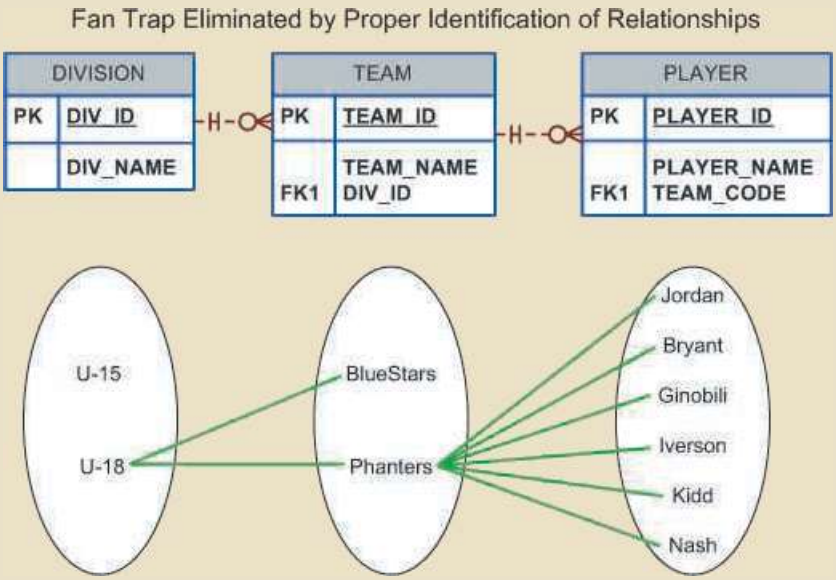


FIGURE 5.12 CORRECTED ERD AFTER REMOVAL OF THE FAN TRAP

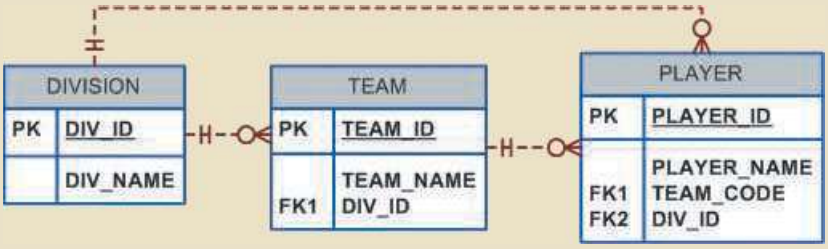


5-4d Design Case 4: Redundant Relationships

Although redundancy is often good to have in computer environments (multiple backups in multiple places, for example), redundancy is seldom good in the database environment. (As you learned in Chapter 3, The Relational Database Model, redundancies can cause data anomalies in a database.) Redundant relationships occur when there are multiple relationship paths between related entities. The main concern with redundant relationships is that they remain consistent across the model. However, it is important to note that some designs use redundant relationships as a way to simplify the design.

An example of redundant relationships was first introduced in Figure 5.9 during the discussion of maintaining a history of time-variant data. However, the use of the redundant "manages" and "employs" relationships was justified by the fact that such relationships dealt with current data rather than historic data. Another more specific example of a redundant relationship is represented in Figure 5.13.

FIGURE 5.13 A REDUNDANT RELATIONSHIP



In Figure 5.13, note the transitive 1:M relationship between DIVISION and PLAYER through the TEAM entity set. Therefore, the relationship that connects DIVISION and PLAYER is redundant, for all practical purposes. In that case, the relationship could be safely deleted without losing any information-generation capabilities in the model.

Summary

- The extended entity relationship (EER) model adds semantics to the ER model via entity supertypes, subtypes, and clusters. An entity supertype is a generic entity type that is related to one or more entity subtypes.
- A specialization hierarchy depicts the arrangement and relationships between entity supertypes and entity subtypes. Inheritance means that an entity subtype inherits the attributes and relationships of the supertype. Subtypes can be disjoint or overlapping. A subtype discriminator is used to determine to which entity subtype the supertype occurrence is related. The subtypes can exhibit partial or total completeness. There are basically two approaches to developing a specialization hierarchy of entity supertypes and subtypes: specialization and generalization.
- An entity cluster is a "virtual" entity type used to represent multiple entities and relationships in the ERD. An entity cluster is formed by combining multiple interrelated entities and relationships into a single, abstract entity object.
- Natural keys are identifiers that exist in the real world. Natural keys sometimes make good primary keys, but not always. Primary keys must have unique values, they should be nonintelligent, they must not change over time, and they are preferably numeric and composed of a single attribute.
- Composite keys are useful to represent M:N relationships and weak (strong identifying) entities.
- Surrogate primary keys are useful when there is no natural key that makes a suitable primary key, when the primary key is a composite primary key with multiple data types, or when the primary key is too long to be usable.
- In a 1:1 relationship, place the PK of the mandatory entity as a foreign key in the optional entity, as an FK in the entity that causes the fewest nulls, or as an FK where the role is played.

- Time-variant data refers to data whose values change over time and require that you keep a history of data changes. To maintain the history of time-variant data, you must create an entity that contains the new value, the date of change, and any other time-relevant data. This entity maintains a 1:M relationship with the entity for which the history is to be maintained.
- A fan trap occurs when you have one entity in two 1:M relationships to other entities, and there is an association among the other entities that is not expressed in the model. Redundant relationships occur when there are multiple relationship paths between related entities. The main concern with redundant relationships is that they remain consistent across the model.

Key Terms

completeness constraint
design trap
disjoint subtype
EER diagram (EERD)
entity cluster
entity subtype
entity supertype
extended entity relationship model (EERM)
fan trap
generalization
inheritance
natural key (natural identifier)
nonoverlapping subtype
overlapping subtype
partial completeness
specialization
specialization hierarchy
subtype discriminator
surrogate key
time-variant data
total completeness

Online Content

Flashcards and crossword puzzles for key term practice are available at *www.cengagebrain.com*.

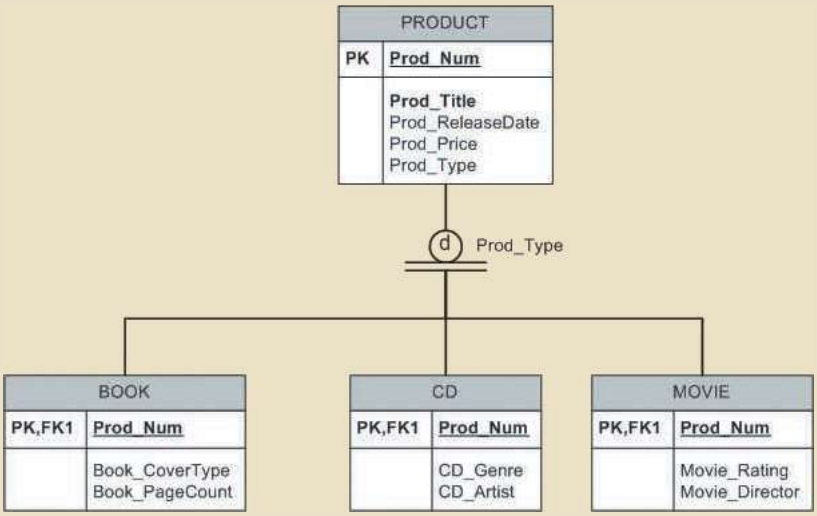
Review Questions

1. What is an entity supertype, and why is it used?
2. What kinds of data would you store in an entity subtype?
3. What is a specialization hierarchy?
4. What is a subtype discriminator? Give an example of its use.
5. What is an overlapping subtype? Give an example.
6. What is the difference between partial completeness and total completeness?

For Questions 7–9, refer to Figure Q5.7.

7. List all of the attributes of a movie.
8. According to the data model, is it required that every entity instance in the PRODUCT table be associated with an entity instance in the CD table? Why, or why not?
9. Is it possible for a book to appear in the BOOK table without appearing in the PRODUCT table? Why, or why not?
10. What is an entity cluster, and what advantages are derived from its use?
11. What primary key characteristics are considered desirable? Explain *why* each characteristic is considered desirable.
12. Under what circumstances are composite primary keys appropriate?
13. What is a surrogate primary key, and when would you use one?

FIGURE Q5.7 THE PRODUCT DATA MODEL



14. When implementing a 1:1 relationship, where should you place the foreign key if one side is mandatory and one side is optional? Should the foreign key be mandatory or optional?
15. What is time-variant data, and how would you deal with such data from a database design point of view?
16. What is the most common design trap, and how does it occur?

Problems

1. Given the following business scenario, create a Crow's Foot ERD using a specialization hierarchy if appropriate. Two-Bit Drilling Company keeps information on employees and their insurance dependents. Each employee has an employee number, name, date of hire, and title. If an employee is an inspector, then the date of certification and certification renewal date should also be recorded in the system. For all employees, the Social Security number and dependent names should be kept. All dependents must be associated with one and only one employee. Some employees will not have dependents, while others will have many dependents.
2. Given the following business scenario, create a Crow's Foot ERD using a specialization hierarchy if appropriate. Tiny Hospital keeps information on patients and hospital rooms. The system assigns each patient a patient ID number. In addition, the patient's name and date of birth are recorded. Some patients are resident patients who spend at least one night in the hospital, and others are outpatients who are treated and released. Resident patients are assigned to a room. Each room is identified by a room number. The system also stores the room type (private or semiprivate) and room fee. Over time, each room will have many patients. Each resident patient will stay in only one room. Every room must have had a patient, and every resident patient must have a room.
3. Given the following business scenario, create a Crow's Foot ERD using a specialization hierarchy if appropriate. Granite Sales Company keeps information on

employees and the departments in which they work. For each department, the department name, internal mail box number, and office phone extension are kept. A department can have many assigned employees, and each employee is assigned to only one department. Employees can be salaried, hourly, or work on contract. All employees are assigned an employee number, which is kept along with the employee's name and address. For hourly employees, hourly wages and target weekly work hours are stored; for example, the company may target 40 hours/week for some employees, 32 for others, and 20 for others. Some salaried employees are salespeople who can earn a commission in addition to their base salary. For all salaried employees, the yearly salary amount is recorded in the system. For salespeople, their commission percentage on sales and commission percentage on profit are stored in the system. For example, John is a salesperson with a base salary of $50,000 per year plus a 2 percent commission on the sales price for all sales he makes, plus another 5 percent of the profit on each of those sales. For contract employees, the beginning date and end date of their contracts are stored along with the billing rate for their hours.

4. In Chapter 4, you saw the creation of the Tiny College database design, which reflected such business rules as "a professor may advise many students" and "a professor may chair one department." Modify the design shown in Figure 4.36 to include these business rules:
   - An employee could be staff, a professor, or an administrator.
   - A professor may also be an administrator.
   - Staff employees have a work-level classification, such as Level I or Level II.
   - Only professors can chair a department. A department is chaired by only one professor.
   - Only professors can serve as the dean of a college. Each of the university's colleges is served by one dean.
   - A professor can teach many classes.
   - Administrators have a position title.

   Given that information, create the complete ERD that contains all primary keys, foreign keys, and main attributes.
5. Tiny College wants to keep track of the history of all its administrative appointments, including dates of appointment and dates of termination. (*Hint:* Time-variant data is at work.) The Tiny College chancellor may want to know how many deans worked in the College of Business between January 1, 1960, and January 1, 2016, or who the dean of the College of Education was in 1990. Given that information, create the complete ERD that contains all primary keys, foreign keys, and main attributes.
6. Some Tiny College staff employees are information technology (IT) personnel. Some IT personnel provide technology support for academic programs, some provide technology infrastructure support, and some provide support for both. IT personnel are not professors; they are required to take periodic training to retain their technical expertise. Tiny College tracks all IT personnel training by date, type, and results (completed versus not completed). Given that information, create the complete ERD that contains all primary keys, foreign keys, and main attributes.
7. The FlyRight Aircraft Maintenance (FRAM) division of the FlyRight Company (FRC) performs all maintenance for FRC's aircraft. Produce a data model segment that reflects the following business rules:

- All mechanics are FRC employees. Not all employees are mechanics.
- Some mechanics are specialized in engine (EN) maintenance. Others are specialized in airframe (AF) maintenance or avionics (AV) maintenance. (Avionics are the electronic components of an aircraft that are used in communication and navigation.) All mechanics take periodic refresher courses to stay current in their areas of expertise. FRC tracks all courses taken by each mechanic—date, course type, certification (Y/N), and performance.
- FRC keeps an employment history of all mechanics. The history includes the date hired, date promoted, and date terminated.

Given those requirements, create the Crow's Foot ERD segment.

Cases

8. "Martial Arts R Us" (MARU) needs a database. MARU is a martial arts school with hundreds of students. The database must keep track of all the classes that are offered, who is assigned to teach each class, and which students attend each class. Also, it is important to track the progress of each student as they advance. Create a complete Crow's Foot ERD for these requirements:
   - Students are given a student number when they join the school. The number is stored along with their name, date of birth, and the date they joined the school.
   - All instructors are also students, but clearly not all students are instructors. In addition to the normal student information, for all instructors, the date that they start working as an instructor must be recorded along with their instructor status (compensated or volunteer).
   - An instructor may be assigned to teach any number of classes, but each class has one and only one assigned instructor. Some instructors, especially volunteer instructors, may not be assigned to any class.
   - A class is offered for a specific level at a specific time, day of the week, and location. For example, one class taught on Mondays at 5:00 p.m. in Room 1 is an intermediate-level class. Another class taught on Mondays at 6:00 p.m. in Room 1 is a beginner-level class. A third class taught on Tuesdays at 5:00 p.m. in Room 2 is an advanced-level class.
   - Students may attend any class of the appropriate level during each week, so there is no expectation that any particular student will attend any particular class session. Therefore, the attendance of students at each individual class meeting must be tracked.
   - A student will attend many different class meetings, and each class meeting is normally attended by many students. Some class meetings may not be attended by any students. New students may not have attended any class meetings yet.
   - At any given meeting of a class, instructors other than the assigned instructor may show up to help. Therefore, a given class meeting may have a head instructor and many assistant instructors, but it will always have at least the one instructor who is assigned to that class. For each class meeting, the date of the class and the instructors' roles (head instructor or assistant instructor) need to be recorded. For example, Mr. Jones is assigned to teach the Monday, 5:00 p.m., intermediate class in Room 1. During a particular meeting of that class, Mr. Jones was the head instructor and Ms. Chen served as an assistant instructor.

- Each student holds a rank in the martial arts. The rank name, belt color, and rank requirements are stored. Most ranks have numerous rank requirements, but each requirement is associated with only one particular rank. All ranks except white belt have at least one requirement.
- A given rank may be held by many students. While it is customary to think of a student as having a single rank, it is necessary to track each student's progress through the ranks. Therefore, every rank that a student attains is kept in the system. New students joining the school are automatically given the rank of white belt. The date that a student is awarded each rank should be kept in the system. All ranks have at least one student who has achieved that rank at some time.

9. The *Journal of E-commerce Research Knowledge* is a prestigious information systems research journal. It uses a peer-review process to select manuscripts for publication. Only about 10 percent of the manuscripts submitted to the journal are accepted for publication. A new issue of the journal is published each quarter. Create a complete ERD to support the business needs described below.
   - Unsolicited manuscripts are submitted by authors. When a manuscript is received, the editor assigns it a number and records some basic information about it in the system, including the title of the manuscript, the date it was received, and a manuscript status of "received." Information about the author(s) is also recorded, including each author's name, mailing address, email address, and affiliation (the author's school or company). Every manuscript must have an author. Only authors who have submitted manuscripts are kept in the system. It is typical for a manuscript to have several authors. A single author may have submitted many different manuscripts to the journal. Additionally, when a manuscript has multiple authors, it is important to record the order in which the authors are listed in the manuscript credits.
   - At his or her earliest convenience, the editor will briefly review the topic of the manuscript to ensure that its contents fall within the scope of the journal. If the content is not appropriate for the journal, the manuscript's status is changed to "rejected," and the author is notified via email. If the content is within the scope of the journal, then the editor selects three or more reviewers to review the manuscript. Reviewers work for other companies or universities and read manuscripts to ensure their scientific validity. For each reviewer, the system records a reviewer number, name, email address, affiliation, and areas of interest. Areas of interest are predefined areas of expertise that the reviewer has specified. An area of interest is identified by an IS code and includes a description (for example, IS2003 is the code for "database modeling"). A reviewer can have many areas of interest, and an area of interest can be associated with many reviewers. All reviewers must specify at least one area of interest. It is unusual, but possible, to have an area of interest for which the journal has no reviewers. The editor will change the status of the manuscript to "under review" and record which reviewers received the manuscript and the date it was sent to each reviewer. A reviewer will typically receive several manuscripts to review each year, although new reviewers may not have received any manuscripts yet.
   - The reviewers will read the manuscript at their earliest convenience and provide feedback to the editor. The feedback from each reviewer includes rating the manuscript on a 10-point scale for appropriateness, clarity, methodology, and contribution to the field, as well as a recommendation for publication (accept or reject). The editor will record all of this information in the system for each review received, along with the date the feedback was received. Once all of the reviewers

have provided their evaluations, the editor will decide whether to publish the manuscript and change its status to "accepted" or "rejected." If the manuscript will be published, the date of acceptance is recorded.

- Once a manuscript has been accepted for publication, it must be scheduled. For each issue of the journal, the publication period (fall, winter, spring, or summer), publication year, volume, and number are recorded. An issue will contain many manuscripts, although the issue may be created in the system before it is known which manuscripts will be published in that issue. An accepted manuscript appears in only one issue of the journal. Each manuscript goes through a typesetting process that formats the content, including fonts, font size, line spacing, justification, and so on. Once the manuscript has been typeset, its number of pages is recorded in the system. The editor will then decide which issue each accepted manuscript will appear in and the order of manuscripts within each issue. The order and the beginning page number for each manuscript must be stored in the system. Once the manuscript has been scheduled for an issue, the status of the manuscript is changed to "scheduled." Once an issue is published, the print date for the issue is recorded, and the status of each manuscript in that issue is changed to "published."

10. Global Unified Technology Sales (GUTS) is moving toward a "bring your own device" (BYOD) model for employee computing. Employees can use traditional desktop computers in their offices. They can also use a variety of personal mobile computing devices such as tablets, smartphones, and laptops. The new computing model introduces some security risks that GUTS is attempting to address. The company wants to ensure that any devices connecting to their servers are properly registered and approved by the Information Technology department. Create a complete ERD to support the business needs described below:

    - Every employee works for a department that has a department code, name, mail box number, and phone number. The smallest department currently has 5 employees, and the largest department has 40 employees. This system will only track in which department an employee is currently employed. Very rarely, a new department can be created within the company. At such times, the department may exist temporarily without any employees. For every employee, an employee number and name (first, last, and middle initial) are recorded in the system. It is also necessary to keep each employee's title.
    - An employee can have many devices registered in the system. Each device is assigned an identification number when it is registered. Most employees have at least one device, but newly hired employees might not have any devices registered initially. For each device, the brand and model need to be recorded. Only devices that are registered to an employee will be in the system. While unlikely, it is possible that a device could transfer from one employee to another. However, if that happens, only the employee who currently owns the device is tracked in the system. When a device is registered in the system, the date of that registration needs to be recorded.
    - Devices can be either desktop systems that reside in a company office or mobile devices. Desktop devices are typically provided by the company and are intended to be a permanent part of the company network. As such, each desktop device is assigned a static IP address, and the MAC address for the computer hardware is kept in the system. A desktop device is kept in a static location (building name and office number). This location should also be kept in the system so that, if the device becomes compromised, the IT department can dispatch someone to remediate the problem.
    - For mobile devices, it is important to also capture the device's serial number, which operating system (OS) it is using, and the version of the OS. The IT department is also verifying that each mobile device has a screen lock enabled and has

encryption enabled for data. The system should support storing information on whether or not each mobile device has these capabilities enabled.

- Once a device is registered in the system, and the appropriate capabilities are enabled if it is a mobile device, the device may be approved for connections to one or more servers. Not all devices meet the requirements to be approved at first so the device might be in the system for a period of time before it is approved to connect to any server. GUTS has a number of servers, and a device must be approved for each server individually. Therefore, it is possible for a single device to be approved for several servers but not for all servers.
- Each server has a name, brand, and IP address. Within the IT department's facilities are a number of climate-controlled server rooms where the physical servers can be located. Which room each server is in should also be recorded. Further, it is necessary to track which operating system is being used on each server. Some servers are virtual servers and some are physical servers. If a server is a virtual server, then the system should track which physical server it is running on. A single physical server can host many virtual servers, but each virtual server is hosted on only one physical server. Only physical servers can host a virtual server. In other words, one virtual server cannot host another virtual server. Not all physical servers host a virtual server.
- A server will normally have many devices that are approved to access the server, but it is possible for new servers to be created that do not yet have any approved devices. When a device is approved for connection to a server, the date of that approval should be recorded. It is also possible for a device that was approved for a server to lose its approval. If that happens, the date that the approval was removed should be recorded. If a device loses its approval, it may regain that approval at a later date if whatever circumstance that lead to the removal is resolved.
- A server can provide many user services, such as email, chat, homework managers, and others. Each service on a server has a unique identification number and name. The date that GUTS began offering that service should be recorded. Each service runs on only one server although new servers might not offer any services initially. Client-side services are not tracked in this system so every service must be associated with a server.
- Employees must get permission to access a service before they can use it. Most employees have permissions to use a wide array of services, but new employees might not have permission on any service. Each service can support multiple approved employees as users, but new services might not have any approved users at first. The date on which the employee is approved to use a service is tracked by the system. The first time an employee is approved to access a service, the employee must create a username and password. This will be the same username and password that the employee will use for every service for which the employee is eventually approved.

11. Global Computer Solutions (GCS) is an information technology consulting company with many offices throughout the United States. The company's success is based on its ability to maximize its resources—that is, its ability to match highly skilled employees with projects according to region. To better manage its projects, GCS has contacted you to design a database so GCS managers can keep track of their customers, employees, projects, project schedules, assignments, and invoices.

    The GCS database must support all of GCS's operations and information requirements. A basic description of the main entities follows:

    - The *employees* of GCS must have an employee ID, a last name, a middle initial, a first name, a region, and a date of hire recorded in the system.

- Valid *regions* are as follows: Northwest (NW), Southwest (SW), Midwest North (MN), Midwest South (MS), Northeast (NE), and Southeast (SE).
- Each employee has many skills, and many employees have the same skill.
- Each *skill* has a skill ID, description, and rate of pay. Valid skills are as follows: Data Entry I, Data Entry II, Systems Analyst I, Systems Analyst II, Database Designer I, Database Designer II, Cobol I, Cobol II, C++ I, C++ II, VB I, VB II, ColdFusion I, ColdFusion II, ASP I, ASP II, Oracle DBA, MS SQL Server DBA, Network Engineer I, Network Engineer II, Web Administrator, Technical Writer, and Project Manager. Table P5.11a shows an example of the Skills Inventory.

TABLE P5.11A

SKILL	EMPLOYEE
Data Entry I	Seaton Amy; Williams Josh; Underwood Trish
Data Entry II	Williams Josh; Seaton Amy
Systems Analyst I	Craig Brett; Sewell Beth; Robbins Erin; Bush Emily; Zebras Steve
Systems Analyst II	Chandler Joseph; Burklow Shane; Robbins Erin
DB Designer I	Yarbrough Peter; Smith Mary
DB Designer II	Yarbrough Peter; Pascoe Jonathan
Cobol I	Kattan Chris; Ephanor Victor; Summers Anna; Ellis Maria
Cobol II	Kattan Chris; Ephanor Victor; Batts Melissa
C++ I	Smith Jose; Rogers Adam; Cope Leslie
C++ II	Rogers Adam; Bible Hanah
VB I	Zebras Steve; Ellis Maria
VB II	Zebras Steve; Newton Christopher
ColdFusion I	Duarte Miriam; Bush Emily
ColdFusion II	Bush Emily; Newton Christopher
ASP I	Duarte Miriam; Bush Emily
ASP II	Duarte Miriam; Newton Christopher
Oracle DBA	Smith Jose; Pascoe Jonathan
SQL Server DBA	Yarbrough Peter; Smith Jose
Network Engineer I	Bush Emily; Smith Mary
Network Engineer II	Bush Emily; Smith Mary
Web Administrator	Bush Emily; Smith Mary; Newton Christopher
Technical Writer	Kilby Surgena; Bender Larry
Project Manager	Paine Brad; Mudd Roger; Kenyon Tiffany; Connor Sean

- GCS has many *customers*. Each customer has a customer ID, name, phone number, and region.
- GCS works by *projects*. A project is based on a contract between the customer and GCS to design, develop, and implement a computerized solution. Each project has specific characteristics such as the project ID, the customer to which the project belongs, a brief description, a project date (the date the contract was signed), an estimated project start date and end date, an estimated project budget, an actual start date, an actual end date, an actual cost, and one employee assigned as the manager of the project.

- The actual cost of the project is updated each Friday by adding that week's cost to the actual cost. The week's cost is computed by multiplying the hours each employee worked by the rate of pay for that skill.
- The employee who is the manager of the project must complete a *project schedule*, which effectively is a design and development plan. In the project schedule (or plan), the manager must determine the tasks that will be performed to take the project from beginning to end. Each task has a task ID, a brief task description, starting and ending dates, the types of skills needed, and the number of employees (with the required skills) needed to complete the task. General tasks are the initial interview, database and system design, implementation, coding, testing, and final evaluation and sign-off. For example, GCS might have the project schedule shown in Table P5.11b.

TABLE P5.11B

PROJECT ID: 1		DESCRIPTION: SALES MANAGEMENT SYSTEM		
COMPANY: SEE ROCKS		CONTRACT DATE: 2/12/2016		REGION: NW
START DATE: 3/1/2016		END DATE: 7/1/2016		BUDGET: $15,500
START DATE	END DATE	TASK DESCRIPTION	SKILL(S) REQUIRED	QUANTITY REQUIRED
3/1/16	3/6/16	Initial interview	Project Manager Systems Analyst II DB Designer I	1 1 1
3/11/16	3/15/16	Database design	DB Designer I	1
3/11/16	4/12/16	System design	Systems Analyst II Systems Analyst I	1 2
3/18/16	3/22/16	Database implementation	Oracle DBA	1
3/25/16	5/20/16	System coding and testing	Cobol I Cobol II Oracle DBA	2 1 1
3/25/16	6/7/16	System documentation	Technical Writer	1
6/10/16	6/14/16	Final evaluation	Project Manager Systems Analyst II DB Designer I Cobol II	1 1 1 1
6/17/16	6/21/16	On-site system online and data loading	Project Manager Systems Analyst II DB Designer I Cobol II	1 1 1 1
7/1/16	7/1/16	Sign-off	Project Manager	1

- GCS pools all of its employees by region; from this pool, employees are assigned to a specific task scheduled by the project manager. For example, in the first project's schedule, you know that a Systems Analyst II, Database Designer I, and Project Manager are needed for the period from 3/1/16 to 3/6/16. The project manager is assigned when the project is created and remains for the duration of the project. Using that information, GCS searches the employees who are located in the same region as the customer, matches the skills required, and assigns the employees to the project task.

- Each project schedule task can have many employees assigned to it, and a given employee can work on multiple project tasks. However, an employee can work on only one project task at a time. For example, if an employee is already assigned to work on a project task from 2/20/16 to 3/3/16, the employee cannot work on another task until the current assignment is closed (ends). The date that an assignment is closed does not necessarily match the ending date of the project schedule task because a task can be completed ahead of or behind schedule.
- Given all of the preceding information, you can see that the assignment associates an employee with a project task, using the project schedule. Therefore, to keep track of the *assignment*, you require at least the following information: assignment ID, employee, project schedule task, assignment start date, and assignment end date. The end date could be any date, as some projects run ahead of or behind schedule. Table P5.11c shows a sample assignment form.

TABLE P5.11C

PROJECT ID: 1		DESCRIPTION: SALES MANAGEMENT SYSTEM				
COMPANY: SEE ROCKS		CONTRACT DATE: 2/12/2016		AS OF: 03/29/16		
SCHEDULED				ACTUAL ASSIGNMENTS		
PROJECT TASK	START DATE	END DATE	SKILL	EMPLOYEE	START DATE	END DATE
Initial interview	3/1/16	3/6/16	Project Mgr. Sys. Analyst II DB Designer I	101-Connor S. 102-Burklow S. 103-Smith M.	3/1/16 3/1/16 3/1/16	3/6/16 3/6/16 3/6/16
Database design	3/11/16	3/15/16	DB Designer I	104-Smith M.	3/11/16	3/14/16
System design	3/11/16	4/12/16	Sys. Analyst II Sys. Analyst I Sys. Analyst I	105-Burklow S. 106-Bush E. 107-Zebras S.	3/11/16 3/11/16 3/11/16	
Database implementation	3/18/16	3/22/16	Oracle DBA	108-Smith J.	3/15/16	3/19/16
System coding and testing	3/25/16	5/20/16	Cobol I Cobol I Cobol II Oracle DBA	109-Summers A. 110-Ellis M. 111-Ephanor V. 112-Smith J.	3/21/16 3/21/16 3/21/16 3/21/16	
System documentation	3/25/16	6/7/16	Tech. Writer	113-Kilby S.	3/25/16	
Final evaluation	6/10/16	6/14/16	Project Mgr. Sys. Analyst II DB Designer I Cobol II			
On-site system online and data loading	6/17/16	6/21/16	Project Mgr. Sys. Analyst II DB Designer I Cobol II			
Sign-off	7/1/16	7/1/16	Project Mgr.			

(*Note:* The assignment number is shown as a prefix of the employee name—for example, 101 or 102.) Assume that the assignments shown previously are the only ones as of the date of this design. The assignment number can be any number that matches your database design.

- Employee work hours are kept in a *work log*, which contains a record of the actual hours worked by employees on a given assignment. The work log is a form that the employee fills out at the end of each week (Friday) or at the end of each month. The form contains the date, which is either the current Friday of the month or the last workday of the month if it does not fall on a Friday. The form also contains the assignment ID, the total hours worked either that week or up to the end of the month, and the bill number to which the work-log entry is charged. Obviously, each work-log entry can be related to only one bill. A sample list of the current work-log entries for the first sample project is shown in Table P5.11d.

TABLE P5.11D

EMPLOYEE NAME	WEEK ENDING	ASSIGNMENT NUMBER	HOURS WORKED	BILL NUMBER
Burklow S.	3/1/16	1-102	4	xxx
Connor S.	3/1/16	1-101	4	xxx
Smith M.	3/1/16	1-103	4	xxx
Burklow S.	3/8/16	1-102	24	xxx
Connor S.	3/8/16	1-101	24	xxx
Smith M.	3/8/16	1-103	24	xxx
Burklow S.	3/15/16	1-105	40	xxx
Bush E.	3/15/16	1-106	40	xxx
Smith J.	3/15/16	1-108	6	xxx
Smith M.	3/15/16	1-104	32	xxx
Zebras S.	3/15/16	1-107	35	xxx
Burklow S.	3/22/16	1-105	40	
Bush E.	3/22/16	1-106	40	
Ellis M.	3/22/16	1-110	12	
Ephanor V.	3/22/16	1-111	12	
Smith J.	3/22/16	1-108	12	
Smith J.	3/22/16	1-112	12	
Summers A.	3/22/16	1-109	12	
Zebras S.	3/22/16	1-107	35	
Burklow S.	3/29/16	1-105	40	
Bush E.	3/29/16	1-106	40	
Ellis M.	3/29/16	1-110	35	
Ephanor V.	3/29/16	1-111	35	
Kilby S.	3/29/16	1-113	40	
Smith J.	3/29/16	1-112	35	
Summers A.	3/29/16	1-109	35	
Zebras S.	3/29/16	1-107	35	

Note: xxx represents the bill ID. Use the one that matches the bill number in your database.

- Finally, every 15 days, a *bill* is written and sent to the customer for the total hours worked on the project during that period. When GCS generates a bill, it uses the bill number to update the work-log entries that are part of the bill. In summary, a bill can refer to many work-log entries, and each work-log entry can be related to only one bill. GCS sent one bill on 3/15/16 for the first project (SEE ROCKS), totaling the hours worked between 3/1/16 and 3/15/16. Therefore, you can safely assume that there is only one bill in this table and that the bill covers the work-log entries shown in the preceding form.

Your assignment is to create a database that fulfills the operations described in this problem. The minimum required entities are employee, skill, customer, region, project, project schedule, assignment, work log, and bill. (There are additional required entities that are not listed.)

- Create all of the required tables and required relationships.
- Create the required indexes to maintain entity integrity when using surrogate primary keys.
- Populate the tables as needed, as indicated in the sample data and forms.

Chapter 6

Normalization of Database Tables

In this chapter, you will learn:

- What normalization is and what role it plays in the database design process
- About the normal forms 1NF, 2NF, 3NF, BCNF, and 4NF
- How normal forms can be transformed from lower normal forms to higher normal forms
- That normalization and ER modeling are used concurrently to produce a good database design
- That some situations require denormalization to generate information efficiently

Preview

Good database design must be matched to good table structures. In this chapter, you will learn to evaluate and design good table structures to control data redundancies, thereby avoiding data anomalies. The process that yields such desirable results is known as normalization.

To recognize and appreciate the characteristics of a good table structure, it is useful to examine a poor one. Therefore, the chapter begins by examining the characteristics of a poor table structure and the problems it creates. You then learn how to correct the table structure. This methodology will yield important dividends: you will know how to design a good table structure and how to repair a poor one.

You will discover not only that data anomalies can be eliminated through normalization, but that a properly normalized set of table structures is actually less complicated to use than an unnormalized set. In addition, you will learn that the normalized set of table structures more faithfully reflects an organization's real operations.

Data Files and Available Formats

	MS Access	Oracle	MS SQL	My SQL		MS Access	Oracle	MS SQL	My SQL
CH06_ConstructCo	✓	✓	✓	✓	CH06_Service	✓	✓	✓	✓
CH06_Eval	✓	✓	✓	✓					

Data Files Available on cengagebrain.com

6-1 Database Tables and Normalization

normalization
A process that assigns attributes to entities so that data redundancies are reduced or eliminated.

denormalization
A process by which a table is changed from a higher-level normal form to a lower-level normal form, usually to increase processing speed. Denormalization potentially yields data anomalies.

prime attribute
A key attribute; that is, an attribute that is part of a key or is the whole key. See also *key attributes*.

key attributes
The attributes that form a primary key. See also *prime attribute*.

nonprime attribute
An attribute that is not part of a key.

nonkey attribute
See *nonprime attribute*.

Having good relational database software is not enough to avoid the data redundancy discussed in Chapter 1, Database Systems. If the database tables are treated as though they are files in a file system, the relational database management system (RDBMS) never has a chance to demonstrate its superior data-handling capabilities.

The table is the basic building block of database design. Consequently, the table's structure is of great interest. Ideally, the database design process explored in Chapter 4, Entity Relationship (ER) Modeling, yields good table structures. Yet, it is possible to create poor table structures even in a good database design. How do you recognize a poor table structure, and how do you produce a good table? The answer to both questions involves normalization. **Normalization** is a process for evaluating and correcting table structures to minimize data redundancies, thereby reducing the likelihood of data anomalies. The normalization process involves assigning attributes to tables based on the concept of determination you learned in Chapter 3, The Relational Database Model.

Normalization works through a series of stages called normal forms. The first three stages are described as first normal form (1NF), second normal form (2NF), and third normal form (3NF). From a structural point of view, 2NF is better than 1NF, and 3NF is better than 2NF. For most purposes in business database design, 3NF is as high as you need to go in the normalization process. However, you will discover that properly designed 3NF structures also meet the requirements of fourth normal form (4NF).

Although normalization is a very important ingredient in database design, you should not assume that the highest level of normalization is always the most desirable. Generally, the higher the normal form, the more relational join operations you need to produce a specified output. Also, more resources are required by the database system to respond to end-user queries. A successful design must also consider end-user demand for fast performance. Therefore, you will occasionally need to *denormalize* some portions of a database design to meet performance requirements. **Denormalization** produces a lower normal form; that is, a 3NF will be converted to a 2NF through denormalization. However, the price you pay for increased performance through denormalization is greater data redundancy.

Note

Although the word *table* is used throughout this chapter, formally, normalization is concerned with relations. In Chapter 3 you learned that the terms *table* and *relation* are frequently used interchangeably. In fact, you can say that a table is the implementation view of a logical relation that meets some specific conditions. (See Table 3.1.) However, being more rigorous, the mathematical relation does not allow duplicate tuples; whereas they could exist in tables (see Section 6-5). Also, in normalization terminology, any attribute that is at least part of a key is known as a **prime attribute** instead of the more common term **key attribute**, which was introduced earlier. Conversely, a **nonprime attribute**, or a **nonkey attribute**, is not part of any candidate key.

6-2 The Need For Normalization

Normalization is typically used in conjunction with the entity relationship modeling that you learned in the previous chapters. Database designers commonly use normalization in two situations. When designing a new database structure based on the business requirements of the end users, the database designer will construct a data model using a technique such as Crow's Foot notation ERDs. After the initial design is complete,

the designer can use normalization to analyze the relationships among the attributes within each entity and determine if the structure can be improved through normalization. Alternatively, database designers are often asked to modify existing data structures that can be in the form of flat files, spreadsheets, or older database structures. Again, by analyzing relationships among the attributes or fields in the data structure, the database designer can use the normalization process to improve the existing data structure and create an appropriate database design. Whether you are designing a new database structure or modifying an existing one, the normalization process is the same.

To get a better idea of the normalization process, consider the simplified database activities of a construction company that manages several building projects. Each project has its own project number, name, assigned employees, and so on. Each employee has an employee number, name, and job classification, such as engineer or computer technician.

The company charges its clients by billing the hours spent on each contract. The hourly billing rate is dependent on the employee's position. For example, one hour of computer technician time is billed at a different rate than one hour of engineer time. Periodically, a report is generated that contains the information displayed in Table 6.1.

The total charge in Table 6.1 is a derived attribute and is not stored in the table at this point.

The easiest short-term way to generate the required report might seem to be a table whose contents correspond to the reporting requirements. (See Figure 6.1.)

FIGURE 6.1 TABULAR REPRESENTATION OF THE REPORT FORMAT

Table name: RPT_FORMAT **Database name: Ch06_ConstructCo**

PROJ_NUM	PROJ_NAME	EMP_NUM	EMP_NAME	JOB_CLASS	CHG_HOUR	HOURS
15	Evergreen	103	June E. Arbough	Elect. Engineer	84.50	23.8
		101	John G. News	Database Designer	105.00	19.4
		105	Alice K. Johnson *	Database Designer	105.00	35.7
		106	William Smithfield	Programmer	35.75	12.6
		102	David H. Senior	Systems Analyst	96.75	23.8
18	Amber Wave	114	Annelise Jones	Applications Designer	48.10	24.6
		118	James J. Frommer	General Support	18.36	45.3
		104	Anne K. Ramoras *	Systems Analyst	96.75	32.4
		112	Darlene M. Smithson	DSS Analyst	45.95	44.0
22	Rolling Tide	105	Alice K. Johnson	Database Designer	105.00	64.7
		104	Anne K. Ramoras	Systems Analyst	96.75	48.4
		113	Delbert K. Joenbrood *	Applications Designer	48.10	23.6
		111	Geoff B. Wabash	Clerical Support	26.87	22.0
		106	William Smithfield	Programmer	35.75	12.8
25	Starflight	107	Maria D. Alonzo	Programmer	35.75	24.6
		115	Travis B. Bawangi	Systems Analyst	96.75	45.8
		101	John G. News *	Database Designer	105.00	56.3
		114	Annelise Jones	Applications Designer	48.10	33.1
		108	Ralph B. Washington	Systems Analyst	96.75	23.6
		118	James J. Frommer	General Support	18.36	30.5
		112	Darlene M. Smithson	DSS Analyst	45.95	41.4

Note that the data in Figure 6.1 reflects the assignment of employees to projects. Apparently, an employee can be assigned to more than one project. For example, Darlene Smithson (EMP_NUM = 112) has been assigned to two projects: Amber Wave and Starflight. Given the structure of the dataset, each project includes only a single occurrence of any one employee. Therefore, knowing the PROJ_NUM and EMP_NUM values will let you find the job classification and its hourly charge. In addition, you will know the total number of hours each employee worked on each project. (The total charge—a derived attribute whose value can be computed by multiplying the hours billed and the charge per hour—has not been included in Figure 6.1. No structural harm is done if this derived attribute is included.)

TABLE 6.1

A SAMPLE REPORT LAYOUT

PROJECT NUMBER	PROJECT NAME	EMPLOYEE NUMBER	EMPLOYEE NAME	JOB CLASS	CHARGE/ HOUR	HOURS BILLED	TOTAL CHARGE
15	Evergreen	103	June E. Arbough	Elec. Engineer	$ 84.50	23.8	$ 2,011.10
		101	John G. News	Database Designer	$105.00	19.4	$ 2,037.00
		105	Alice K. Johnson *	Database Designer	$105.00	35.7	$ 3,748.50
		106	William Smithfield	Programmer	$ 35.75	12.6	$ 450.45
		102	David H. Senior	Systems Analyst	$ 96.75	23.8	$ 2,302.65
				Subtotal			**$10,549.70**
18	Amber Wave	114	Annelise Jones	Applications Designer	$ 48.10	24.6	$ 1,183.26
		118	James J. Frommer	General Support	$ 18.36	45.3	$ 831.71
		104	Anne K. Ramoras *	Systems Analyst	$ 96.75	32.4	$ 3,134.70
		112	Darlene M. Smithson	DSS Analyst	$ 45.95	44.0	$ 2,021.80
				Subtotal			**$ 7,171.47**
22	Rolling Tide	105	Alice K. Johnson	Database Designer	$105.00	64.7	$ 6,793.50
		104	Anne K. Ramoras	Systems Analyst	$96.75	48.4	$ 4,682.70
		113	Delbert K. Joenbrood *	Applications Designer	$48.10	23.6	$ 1,135.16
		111	Geoff B. Wabash	Clerical Support	$26.87	22.0	$ 591.14
		106	William Smithfield	Programmer	$35.75	12.8	$ 457.60
				Subtotal			**$13,660.10**
25	Starflight	107	Maria D. Alonzo	Programmer	$ 35.75	24.6	$ 879.45
		115	Travis B. Bawangi	Systems Analyst	$ 96.75	45.8	$ 4,431.15
		101	John G. News *	Database Designer	$105.00	56.3	$ 5,911.50
		114	Annelise Jones	Applications Designer	$ 48.10	33.1	$ 1,592.11
		108	Ralph B. Washington	Systems Analyst	$ 96.75	23.6	$ 2,283.30
		118	James J. Frommer	General Support	$ 18.36	30.5	$ 559.98
		112	Darlene M. Smithson	DSS Analyst	$ 45.95	41.4	$ 1,902.33
				Subtotal			**$17,559.82**
				Total			**$48,941.09**

Note: A * indicates the project leader.

Unfortunately, the structure of the dataset in Figure 6.1 does not conform to the requirements discussed in Chapter 3, nor does it handle data very well. Consider the following deficiencies:

1. The project number (PROJ_NUM) is apparently intended to be a primary key (PK) or at least a part of a PK, but it contains nulls. Given the preceding discussion, you know that PROJ_NUM + EMP_NUM will define each row.
2. The table entries invite data inconsistencies. For example, the JOB_CLASS value "Elect. Engineer" might be entered as "Elect.Eng." in some cases, "El. Eng." in others, and "EE" in still others.
3. The table displays data redundancies that yield the following anomalies:
   a. *Update anomalies*. Modifying the JOB_CLASS for employee number 105 requires many potential alterations, one for each EMP_NUM = 105.
   b. *Insertion anomalies*. Just to complete a row definition, a new employee must be assigned to a project. If the employee is not yet assigned, a phantom project must be created to complete the employee data entry.
   c. *Deletion anomalies*. Suppose that only one employee is associated with a given project. If that employee leaves the company and the employee data is deleted, the project information will also be deleted. To prevent the loss of the project information, a fictitious employee must be created.

In spite of those structural deficiencies, the table structure *appears* to work; the report is generated with ease. Unfortunately, the report might yield varying results depending on what data anomaly has occurred. For example, if you want to print a report to show the total "hours worked" value by the job classification "Database Designer," that report will not include data for "DB Design" and "Database Design" data entries. Such reporting anomalies cause a multitude of problems for managers—and cannot be fixed through application programming.

Even if careful data-entry auditing can eliminate most of the reporting problems (at a high cost), it is easy to demonstrate that even a simple data entry becomes inefficient. Given the existence of update anomalies, suppose Darlene M. Smithson is assigned to work on the Evergreen project. The data-entry clerk must update the PROJECT file with the following entry:

15 Evergreen 112 Darlene M. Smithson DSS Analyst $45.95 0.0

to match the attributes PROJ_NUM, PROJ_NAME, EMP_NUM, EMP_NAME, JOB_CLASS, CHG_HOUR, and HOURS. (If Smithson has just been assigned to the project, the total number of hours worked is 0.0.)

Note

Remember that the naming convention makes it easy to see what each attribute stands for and its likely origin. For example, PROJ_NAME uses the prefix PROJ to indicate that the attribute is associated with the PROJECT table, while the NAME component is self-documenting as well. However, keep in mind that name length is also an issue, especially in the prefix designation. For that reason, the prefix CHG was used rather than CHARGE. (Given the database's context, it is not likely that the prefix will be misunderstood.)

Each time another employee is assigned to a project, some data entries (such as PROJ_NAME, EMP_NAME, and CHG_HOUR) are unnecessarily repeated. Imagine

the data-entry chore when 200 or 300 table entries must be made! The entry of the employee number should be sufficient to identify Darlene M. Smithson, her job description, and her hourly charge. Because only one person is identified by the number 112, that person's characteristics (name, job classification, and so on) should not have to be entered each time the main file is updated. Unfortunately, the structure displayed in Figure 6.1 does not make allowances for that possibility.

The data redundancy evident in Figure 6.1 leads to wasted data storage space. Even worse, data redundancy produces data anomalies. For example, suppose the data-entry clerk had entered the data as:

15 Evergeen 112 Darla Smithson DCS Analyst $45.95 0.0

At first glance, the data entry appears to be correct. But is *Evergeen* the same project as *Evergreen*? And is *DCS Analyst* supposed to be *DSS Analyst*? Is Darla Smithson the same person as Darlene M. Smithson? Such confusion is a data integrity problem because the data entry failed to conform to the rule that all copies of redundant data must be identical.

The possibility of introducing data integrity problems caused by data redundancy must be considered during database design. The relational database environment is especially well suited to help the designer overcome those problems.

6-3 The Normalization Process

In this section, you will learn how to use normalization to produce a set of normalized tables to store the data that will be used to generate the required information. The objective of normalization is to ensure that each table conforms to the concept of well-formed relations—in other words, tables that have the following characteristics:

- Each table represents a single subject. For example, a COURSE table will contain only data that directly pertain to courses. Similarly, a STUDENT table will contain only student data.
- No data item will be *unnecessarily* stored in more than one table (in short, tables have minimum controlled redundancy). The reason for this requirement is to ensure that the data is updated in only one place.
- All nonprime attributes in a table are dependent on the primary key—the entire primary key and nothing but the primary key. The reason for this requirement is to ensure that the data is uniquely identifiable by a primary key value.
- Each table is void of insertion, update, or deletion anomalies, which ensures the integrity and consistency of the data.

To accomplish the objective, the normalization process takes you through the steps that lead to successively higher normal forms. The most common normal forms and their basic characteristic are listed in Table 6.2. You will learn the details of these normal forms in the indicated sections.

The concept of keys is central to the discussion of normalization. Recall from Chapter 3 that a candidate key is a minimal (irreducible) superkey. The primary key is the candidate key selected to be the primary means used to identify the rows in the table. Although normalization is typically presented from the perspective of candidate keys, this initial discussion assumes for the sake of simplicity that each table has only one candidate key; therefore, that candidate key is the primary key.

From the data modeler's point of view, the objective of normalization is to ensure that all tables are at least in third normal form (3NF). Even higher-level normal forms exist.

TABLE 6.2

NORMAL FORMS

NORMAL FORM	CHARACTERISTIC	SECTION
First normal form (1NF)	Table format, no repeating groups, and PK identified	6-3a
Second normal form (2NF)	1NF and no partial dependencies	6-3b
Third normal form (3NF)	2NF and no transitive dependencies	6-3c
Boyce-Codd normal form (BCNF)	Every determinant is a candidate key (special case of 3NF)	6-6a
Fourth normal form (4NF)	3NF and no independent multivalued dependencies	6-6b

However, normal forms such as the fifth normal form (5NF) and domain-key normal form (DKNF) are not likely to be encountered in a business environment and are mainly of theoretical interest. Such higher normal forms usually increase joins, which slows performance without adding any value in the elimination of data redundancy. Some very specialized applications, such as statistical research, might require normalization beyond the 4NF, but those applications fall outside the scope of most business operations. Because this book focuses on practical applications of database techniques, the higher-level normal forms are not covered.

Functional Dependence Before outlining the normalization process, it is a good idea to review the concepts of determination and functional dependence that were covered in detail in Chapter 3. Table 6.3 summarizes the main concepts.

TABLE 6.3

FUNCTIONAL DEPENDENCE CONCEPTS

CONCEPT	DEFINITION
Functional dependence	The attribute *B* is fully functionally dependent on the attribute *A* if each value of *A* determines one and only one value of *B*. Example: PROJ_NUM → PROJ_NAME (read as *PROJ_NUM functionally determines PROJ_NAME*) In this case, the attribute PROJ_NUM is known as the determinant attribute, and the attribute PROJ_NAME is known as the dependent attribute.
Functional dependence (generalized definition)	Attribute *A* determines attribute *B* (that is, *B* is functionally dependent on *A*) if all (generalized definition) of the rows in the table that agree in value for attribute *A* also agree in value for attribute *B*.
Fully functional dependence (composite key)	If attribute *B* is functionally dependent on a composite key *A* but not on any subset of that composite key, the attribute *B* is fully functionally dependent on *A*.

It is crucial to understand these concepts because they are used to derive the set of functional dependencies for a given relation. The normalization process works one relation at a time, identifying the dependencies on that relation and normalizing the relation. As you will see in the following sections, normalization starts by identifying the dependencies of a given relation and progressively breaking up the relation (table) into a set of new relations (tables) based on the identified dependencies.

Two types of functional dependencies that are of special interest in normalization are partial dependencies and transitive dependencies. A **partial dependency** exists when there is a functional dependence in which the determinant is only part of the primary key (remember the assumption that there is only one candidate key). For example, if (A, B) → (C, D), B → C, and (A, B) is the primary key, then the functional

partial dependency
A condition in which an attribute is dependent on only a portion (subset) of the primary key.

dependence B → C is a partial dependency because only part of the primary key (B) is needed to determine the value of C. Partial dependencies tend to be straightforward and easy to identify.

A **transitive dependency** exists when there are functional dependencies such that X → Y, Y → Z, and X is the primary key. In that case, the dependency X → Z is a transitive dependency because X determines the value of Z via Y. Unlike partial dependencies, transitive dependencies are more difficult to identify among a set of data. Fortunately, there is an effective way to identify transitive dependencies: they occur only when a functional dependence exists among nonprime attributes. In the previous example, the actual transitive dependency is X → Z. However, the dependency Y → Z signals that a transitive dependency exists. Hence, throughout the discussion of the normalization process, the existence of a functional dependence among nonprime attributes will be considered a sign of a transitive dependency. To address the problems related to transitive dependencies, changes to the table structure are made based on the functional dependence that signals the transitive dependency's existence. Therefore, to simplify the description of normalization, from this point forward the signaling dependency will be called the *transitive dependency*.

6-3a Conversion To First Normal Form

Because the relational model views data as part of a table or a collection of tables in which all key values must be identified, the data depicted in Figure 6.1 might not be stored as shown. Note that Figure 6.1 contains what is known as repeating groups. A **repeating group** derives its name from the fact that a group of multiple entries of the same type can exist for any *single* key attribute occurrence. In Figure 6.1, note that each single project number (PROJ_NUM) occurrence can reference a group of related data entries. For example, the Evergreen project (PROJ_NUM = 15) shows five entries at this point—and those entries are related because they each share the PROJ_NUM = 15 characteristic. Each time a new record is entered for the Evergreen project, the number of entries in the group grows by one.

A relational table must not contain repeating groups. The existence of repeating groups provides evidence that the RPT_FORMAT table in Figure 6.1 fails to meet even the lowest normal form requirements, thus reflecting data redundancies.

Normalizing the table structure will reduce the data redundancies. If repeating groups do exist, they must be eliminated by making sure that each row defines a single entity. In addition, the dependencies must be identified to diagnose the normal form. Identification of the normal form lets you know where you are in the normalization process. Normalization starts with a simple three-step procedure.

transitive dependency
A condition in which an attribute is dependent on another attribute that is not part of the primary key.

repeating group
In a relation, a characteristic describing a group of multiple entries of the same type for a single key attribute occurrence. For example, a car can have multiple colors for its top, interior, bottom, trim, and so on.

Step 1: Eliminate the Repeating Groups

Start by presenting the data in a tabular format, where each cell has a single value and there are no repeating groups. To eliminate the repeating groups, eliminate the nulls by making sure that each repeating group attribute contains an appropriate data value. That change converts the table in Figure 6.1 to 1NF in Figure 6.2.

Step 2: Identify the Primary Key

The layout in Figure 6.2 represents more than a mere cosmetic change. Even a casual observer will note that PROJ_NUM is not an adequate primary key because the project number does not uniquely identify all of the remaining entity (row) attributes. For example, the PROJ_NUM value 15 can identify any one of five employees. To maintain a proper primary key that will *uniquely* identify any attribute value, the new key must be composed of a *combination* of PROJ_NUM and EMP_NUM. For example, using the data shown in Figure 6.2, if you know that

FIGURE 6.2 A TABLE IN FIRST NORMAL FORM

Table name: DATA_ORG_1NF **Database name: Ch06_ConstructCo**

PROJ_NUM	PROJ_NAME	EMP_NUM	EMP_NAME	JOB_CLASS	CHG_HOUR	HOURS
15	Evergreen	103	June E. Arbough	Elect. Engineer	84.50	23.8
15	Evergreen	101	John G. News	Database Designer	105.00	19.4
15	Evergreen	105	Alice K. Johnson *	Database Designer	105.00	35.7
15	Evergreen	106	William Smithfield	Programmer	35.75	12.6
15	Evergreen	102	David H. Senior	Systems Analyst	96.75	23.8
18	Amber Wave	114	Annelise Jones	Applications Designer	48.10	24.6
18	Amber Wave	118	James J. Frommer	General Support	18.36	45.3
18	Amber Wave	104	Anne K. Ramoras *	Systems Analyst	96.75	32.4
18	Amber Wave	112	Darlene M. Smithson	DSS Analyst	45.95	44.0
22	Rolling Tide	105	Alice K. Johnson	Database Designer	105.00	64.7
22	Rolling Tide	104	Anne K. Ramoras	Systems Analyst	96.75	48.4
22	Rolling Tide	113	Delbert K. Joenbrood *	Applications Designer	48.10	23.6
22	Rolling Tide	111	Geoff B. Wabash	Clerical Support	26.87	22.0
22	Rolling Tide	106	William Smithfield	Programmer	35.75	12.8
25	Starflight	107	Maria D. Alonzo	Programmer	35.75	24.6
25	Starflight	115	Travis B. Bawangi	Systems Analyst	96.75	45.8
25	Starflight	101	John G. News *	Database Designer	105.00	56.3
25	Starflight	114	Annelise Jones	Applications Designer	48.10	33.1
25	Starflight	108	Ralph B. Washington	Systems Analyst	96.75	23.6
25	Starflight	118	James J. Frommer	General Support	18.36	30.5
25	Starflight	112	Darlene M. Smithson	DSS Analyst	45.95	41.4

PROJ_NUM = 15 and EMP_NUM = 103, the entries for the attributes PROJ_NAME, EMP_NAME, JOB_CLASS, CHG_HOUR, and HOURS must be Evergreen, June E. Arbough, Elect. Engineer, $84.50, and 23.8, respectively.

Step 3: Identify All Dependencies The identification of the PK in Step 2 means that you have already identified the following dependency:

PROJ_NUM, EMP_NUM → PROJ_NAME, EMP_NAME, JOB_CLASS, CHG_HOUR, HOURS

That is, the PROJ_NAME, EMP_NAME, JOB_CLASS, CHG_HOUR, and HOURS values are all dependent on—they are determined by—the combination of PROJ_NUM and EMP_NUM. There are additional dependencies. For example, the project number identifies (determines) the project name. In other words, the project name is dependent on the project number. You can write that dependency as:

PROJ_NUM → PROJ_NAME

Also, if you know an employee number, you also know that employee's name, job classification, and charge per hour. Therefore, you can identify the dependency shown next:

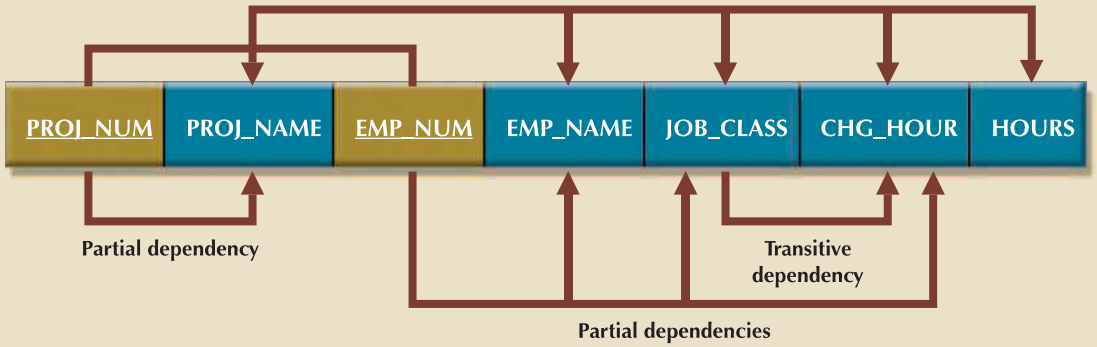
EMP_NUM → EMP_NAME, JOB_CLASS, CHG_HOUR

In simpler terms, an employee has the following attributes: a number, a name, a job classification, and a charge per hour. However, by further studying the data in Figure 6.2, you can see that knowing the job classification means knowing the charge per hour for that job classification. (Notice that all "System Analyst" or "Programmer" positions have the same charge per hour regardless of the project or employee.) In other words, the charge per hour depends on the job classification, not the employee. Therefore, you can identify one last dependency:

JOB_CLASS → CHG_HOUR

This dependency exists between two nonprime attributes; therefore, it is a signal that a transitive dependency exists, and we will refer to it as a transitive dependency. The dependencies you have just examined can also be depicted with the help of the diagram shown in Figure 6.3. Because such a diagram depicts all dependencies found within a given table structure, it is known as a **dependency diagram**. Dependency diagrams are very helpful in getting a bird's-eye view of all the relationships among a table's attributes, and their use makes it less likely that you will overlook an important dependency.

FIGURE 6.3 FIRST NORMAL FORM (1NF) DEPENDENCY DIAGRAM



1NF (PROJ_NUM, EMP_NUM, PROJ_NAME, EMP_NAME, JOB_CLASS, CHG_HOURS, HOURS)

PARTIAL DEPENDENCIES:
(PROJ_NUM → PROJ_NAME)
(EMP_NUM → EMP_NAME, JOB_CLASS, CHG_HOUR)

TRANSITIVE DEPENDENCY:
(JOB_CLASS → CHG_HOUR)

As you examine Figure 6.3, note the following features of a dependency diagram:

1. The primary key attributes are bold, underlined, and in a different color.
2. The arrows above the attributes indicate all desirable dependencies—that is, dependencies based on the primary key. In this case, note that the entity's attributes are dependent on the *combination* of **PROJ_NUM** and **EMP_NUM**.
3. The arrows below the dependency diagram indicate less desirable dependencies. Two types of such dependencies exist:
   a. *Partial dependencies*. You need to know only the PROJ_NUM to determine the PROJ_NAME; that is, the PROJ_NAME is dependent on only part of the primary key. Also, you need to know only the EMP_NUM to find the EMP_NAME, the JOB_CLASS, and the CHG_HOUR. A dependency based on only a part of a composite primary key is a partial dependency.
   b. *Transitive dependencies*. Note that CHG_HOUR is dependent on JOB_CLASS. Because neither CHG_HOUR nor JOB_CLASS is a prime attribute—that is, neither attribute is at least part of a key—the condition is a transitive dependency. In other words, a transitive dependency is a dependency of one nonprime attribute on another nonprime attribute. The problem with transitive dependencies is that they still yield data anomalies.

dependency diagram
A representation of all data dependencies (primary key, partial, or transitive) within a table.

Figure 6.3 includes the relational schema for the table in 1NF and a textual notation for each identified dependency.

Note

The term **first normal form (1NF)** describes the tabular format in which:

- All of the key attributes are defined.
- There are no repeating groups in the table. In other words, each row/column intersection contains one and only one value, not a set of values.
- All attributes are dependent on the primary key.

All relational tables satisfy the 1NF requirements. The problem with the 1NF table structure shown in Figure 6.3 is that it contains partial dependencies—dependencies based on only a part of the primary key.

While partial dependencies are sometimes used for performance reasons, they should be used with caution. Such caution is warranted because a table that contains partial dependencies is still subject to data redundancies, and therefore to various anomalies. The data redundancies occur because every row entry requires duplication of data. For example, if Alice K. Johnson submits her work log, then the user would have to make multiple entries during the course of a day. For each entry, the EMP_NAME, JOB_CLASS, and CHG_HOUR must be entered each time, even though the attribute values are identical for each row entered. Such duplication of effort is very inefficient, and it helps create data anomalies; nothing prevents the user from typing slightly different versions of the employee name, the position, or the hourly pay. For instance, the employee name for EMP_NUM = 102 might be entered as *Dave Senior* or *D. Senior*. The project name might also be entered correctly as *Evergreen* or misspelled as *Evergeen*. Such data anomalies violate the relational database's integrity and consistency rules.

6-3b Conversion To Second Normal Form

Conversion to 2NF occurs only when the 1NF has a composite primary key. If the 1NF has a single-attribute primary key, then the table is automatically in 2NF. The 1NF-to-2NF conversion is simple. Starting with the 1NF format displayed in Figure 6.3, you take the following steps:

Step 1: Make New Tables to Eliminate Partial Dependencies For each component of the primary key that acts as a determinant in a partial dependency, create a new table with a copy of that component as the primary key. While these components are placed in the new tables, it is important that they also remain in the original table as well. The determinants must remain in the original table because they will be the foreign keys for the relationships needed to relate these new tables to the original table. To construct the revised dependency diagram, write each key component on a separate line and then write the original (composite) key on the last line. For example:

PROJ_NUM

EMP_NUM

PROJ_NUM EMP_NUM

Each component will become the key in a new table. In other words, the original table is now divided into three tables (PROJECT, EMPLOYEE, and ASSIGNMENT).

first normal form (1NF)
The first stage in the normalization process. It describes a relation depicted in tabular format, with no repeating groups and a primary key identified. All nonkey attributes in the relation are dependent on the primary key.

Step 2: Reassign Corresponding Dependent Attributes Use Figure 6.3 to determine attributes that are dependent in the partial dependencies. The dependencies for the original key components are found by examining the arrows below the dependency diagram shown in Figure 6.3. The attributes that are dependent in a partial dependency are removed from the original table and placed in the new table with the dependency's determinant. Any attributes that are not dependent in a partial dependency will remain in the original table. In other words, the three tables that result from the conversion to 2NF are given appropriate names (PROJECT, EMPLOYEE, and ASSIGNMENT) and are described by the following relational schemas:

PROJECT (**PROJ_NUM**, PROJ_NAME)

EMPLOYEE (**EMP_NUM**, EMP_NAME, JOB_CLASS, CHG_HOUR)

ASSIGNMENT (**PROJ_NUM**, **EMP_NUM**, ASSIGN_HOURS)

Because the number of hours spent on each project by each employee is dependent on both PROJ_NUM and EMP_NUM in the ASSIGNMENT table, you leave those hours in the ASSIGNMENT table as ASSIGN_HOURS. Notice that the ASSIGNMENT table contains a composite primary key composed of the attributes PROJ_NUM and EMP_NUM. Notice that by leaving the determinants in the original table as well as making them the primary keys of the new tables, primary key/foreign key relationships have been created. For example, in the EMPLOYEE table, EMP_NUM is the primary key. In the ASSIGNMENT table, EMP_NUM is part of the composite primary key (PROJ_NUM, EMP_NUM) and is a foreign key relating the EMPLOYEE table to the ASSIGNMENT table.

The results of Steps 1 and 2 are displayed in Figure 6.4. At this point, most of the anomalies discussed earlier have been eliminated. For example, if you now want to add, change, or delete a PROJECT record, you need to go only to the PROJECT table and make the change to only one row.

second normal form (2NF)
The second stage in the normalization process, in which a relation is in 1NF and there are no partial dependencies (dependencies in only part of the primary key).

Because a partial dependency can exist only when a table's primary key is composed of several attributes, a table whose primary key consists of only a single attribute is automatically in 2NF once it is in 1NF.

Figure 6.4 still shows a transitive dependency, which can generate anomalies. For example, if the charge per hour changes for a job classification held by many employees, that change must be made for *each* of those employees. If you forget to update some of the employee records that are affected by the charge per hour change, different employees with the same job description will generate different hourly charges.

Note

A table is in **second normal form (2NF)** when:

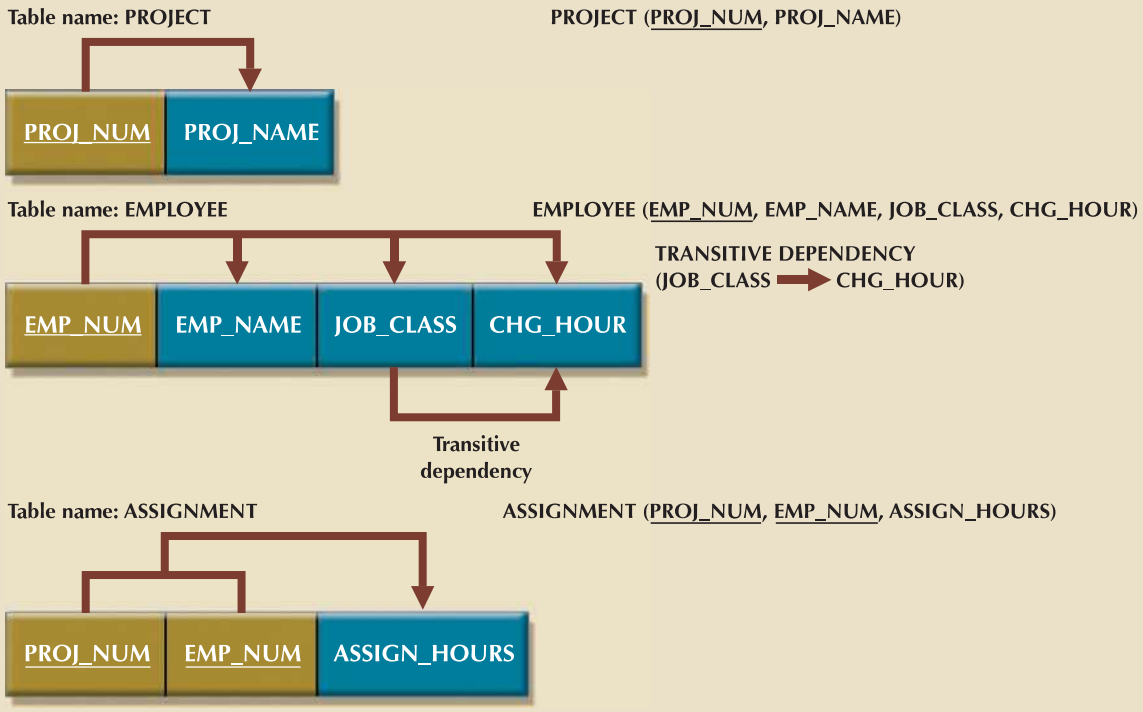
- It is in 1NF.

and

- It includes no partial dependencies; that is, no attribute is dependent on only a portion of the primary key.

It is still possible for a table in 2NF to exhibit transitive dependency. That is, the primary key may rely on one or more nonprime attributes to functionally determine other nonprime attributes, as indicated by a functional dependence among the nonprime attributes.

FIGURE 6.4 SECOND NORMAL FORM (2NF) CONVERSION RESULTS



6-3c Conversion To Third Normal Form

The data anomalies created by the database organization shown in Figure 6.4 are easily eliminated by completing the following two steps:

Step 1: Make New Tables to Eliminate Transitive Dependencies For every transitive dependency, write a copy of its determinant as a primary key for a new table. A **determinant** is any attribute whose value determines other values within a row. If you have three different transitive dependencies, you will have three different determinants. As with the conversion to 2NF, it is important that the determinant remain in the original table to serve as a foreign key. Figure 6.4 shows only one table that contains a transitive dependency. Therefore, write the determinant for this transitive dependency as:

JOB_CLASS

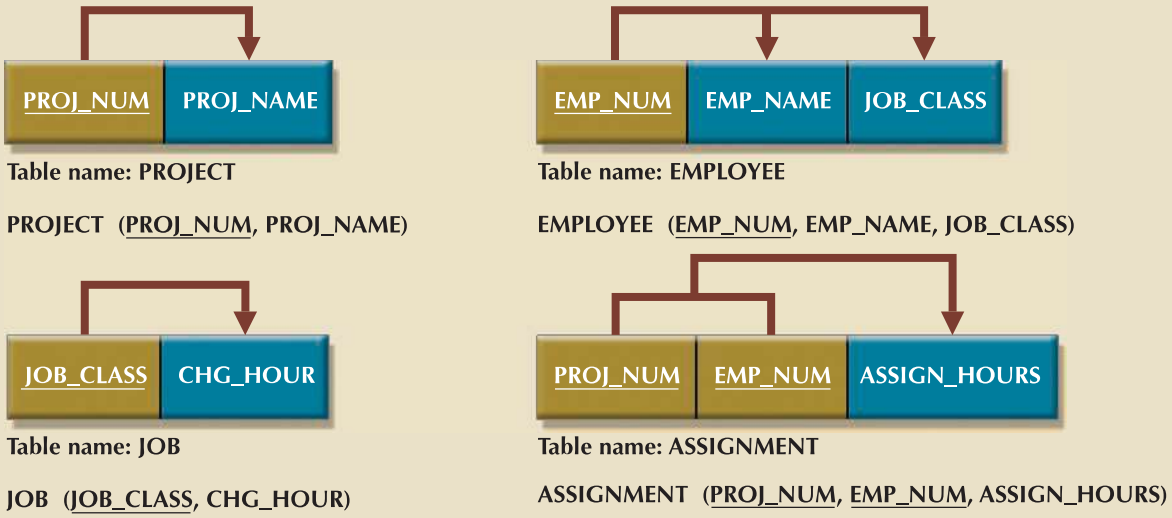
Step 2: Reassign Corresponding Dependent Attributes Using Figure 6.4, identify the attributes that are dependent on each determinant identified in Step 1. Place the dependent attributes in the new tables with their determinants and remove them from their original tables. In this example, eliminate CHG_HOUR from the EMPLOYEE table shown in Figure 6.4 to leave the EMPLOYEE table dependency definition as:

EMP_NUM → EMP_NAME, JOB_CLASS

determinant
Any attribute in a specific row whose value directly determines other values in that row.

Draw a new dependency diagram to show all of the tables you have defined in Steps 1 and 2. Name the table to reflect its contents and function. In this case, JOB seems appropriate. Check all of the tables to make sure that each table has a determinant and that no table contains inappropriate dependencies. When you have completed these steps, you will see the results in Figure 6.5.

FIGURE 6.5 THIRD NORMAL FORM (3NF) CONVERSION RESULTS



In other words, after the 3NF conversion has been completed, your database will contain four tables:

third normal form (3NF)
A table is in 3NF when it is in 2NF and no nonkey attribute is functionally dependent on another nonkey attribute; that is, it cannot include transitive dependencies.

PROJECT (**PROJ_NUM**, PROJ_NAME)

EMPLOYEE (**EMP_NUM**, EMP_NAME, JOB_CLASS)

JOB (**JOB_CLASS**, CHG_HOUR)

ASSIGNMENT (**PROJ_NUM**, **EMP_NUM**, ASSIGN_HOURS)

Note that this conversion has eliminated the original EMPLOYEE table's transitive dependency. The tables are now said to be in third normal form (3NF).

Note

A table is in **third normal form (3NF)** when:

- It is in 2NF.

and

- It contains no transitive dependencies.

It is interesting to note the similarities between resolving 2NF and 3NF problems. To convert a table from 1NF to 2NF, it is necessary to remove the partial dependencies. To convert a table from 2NF to 3NF, it is necessary to remove the transitive dependencies. No matter whether the "problem" dependency is a partial dependency or a transitive dependency,

the solution is the same: create a new table for each problem dependency. The determinant of the problem dependency remains in the original table and is placed as the primary key of the new table. The dependents of the problem dependency are removed from the original table and placed as nonprime attributes in the new table.

Be aware, however, that while the technique is the same, it is imperative that 2NF be achieved before moving on to 3NF; be certain to resolve the partial dependencies before resolving the transitive dependencies. Also, recall the assumption that was made at the beginning of the normalization discussion—that each table has only one candidate key, which is the primary key. If a table has multiple candidate keys, then the overall process remains the same, but there are additional considerations.

For example, if a table has multiple candidate keys and one of them is a composite key, the table can have partial dependencies based on this composite candidate key, even when the primary key chosen is a single attribute. In those cases, following the process described above, those dependencies would be perceived as transitive dependencies and would not be resolved until 3NF. The simplified process described above will allow the designer to achieve the correct result, but through practice, you should recognize all candidate keys and their dependencies as such, and resolve them appropriately. The existence of multiple candidate keys can also influence the identification of transitive dependencies. Previously, a transitive dependency was defined to exist when one nonprime attribute determined another nonprime attribute. In the presence of multiple candidate keys, the definition of a nonprime attribute as an attribute that is not a part of any candidate key is critical. If the determinant of a functional dependence is not the primary key but is a part of another candidate key, then it is not a nonprime attribute and does not signal the presence of a transitive dependency.

6-4 Improving the Design

Now that the table structures have been cleaned up to eliminate the troublesome partial and transitive dependencies, you can focus on improving the database's ability to provide information and on enhancing its operational characteristics. In the next few paragraphs, you will learn about the various types of issues you need to address to produce a good normalized set of tables. Note that for space issues, each section presents just one example—the designer must apply the principle to all remaining tables in the design. Remember that normalization cannot, by itself, be relied on to make good designs. Instead, normalization is valuable because its use helps eliminate data redundancies.

Evaluate PK Assignments Each time a new employee is entered into the EMPLOYEE table, a JOB_CLASS value must be entered. Unfortunately, it is too easy to make data-entry errors that lead to referential integrity violations. For example, entering *DB Designer* instead of *Database Designer* for the JOB_CLASS attribute in the EMPLOYEE table will trigger such a violation. Therefore, it would be better to add a JOB_CODE attribute to create a unique identifier. The addition of a JOB_CODE attribute produces the following dependency:

JOB_CODE → JOB_CLASS, CHG_HOUR

If you assume that the JOB_CODE is a proper primary key, this new attribute does produce the following dependency:

JOB_CLASS → CHG_HOUR

However, this dependency is not a transitive dependency because the determinant is a candidate key. Further, the presence of JOB_CODE greatly decreases the likelihood of referential integrity violations. Note that the new JOB table now has two candidate keys—JOB_CODE and JOB_CLASS. In this case, JOB_CODE is the chosen primary key as well as a surrogate key. A surrogate key, as you should recall, is an artificial PK introduced by the designer with the purpose of simplifying the assignment of primary keys to tables. Surrogate keys are usually numeric, they are often generated automatically by the DBMS, they are free of semantic content (they have no special meaning), and they are usually hidden from the end users.

Evaluate Naming Conventions It is best to adhere to the naming conventions outlined in Chapter 2, Data Models. Therefore, CHG_HOUR will be changed to JOB_CHG_HOUR to indicate its association with the JOB table. In addition, the attribute name JOB_CLASS does not quite describe entries such as *Systems Analyst*, *Database Designer*, and so on; the label JOB_DESCRIPTION fits the entries better. Also, you might have noticed that HOURS was changed to ASSIGN_HOURS in the conversion from 1NF to 2NF. That change lets you associate the hours worked with the ASSIGNMENT table.

Refine Attribute Atomicity It is generally good practice to pay attention to the *atomicity* requirement. An **atomic attribute** is one that cannot be further subdivided. Such an attribute is said to display **atomicity**. Clearly, the use of the EMP_NAME in the EMPLOYEE table is not atomic because EMP_NAME can be decomposed into a last name, a first name, and an initial. By improving the degree of atomicity, you also gain querying flexibility. For example, if you use EMP_LNAME, EMP_FNAME, and EMP_INITIAL, you can easily generate phone lists by sorting last names, first names, and initials. Such a task would be very difficult if the name components were within a single attribute. In general, designers prefer to use simple, single-valued attributes, as indicated by the business rules and processing requirements.

Identify New Attributes If the EMPLOYEE table were used in a real-world environment, several other attributes would have to be added. For example, year-to-date gross salary payments, Social Security payments, and Medicare payments would be desirable. An employee hire date attribute (EMP_HIREDATE) could be used to track an employee's job longevity, and it could serve as a basis for awarding bonuses to long-term employees and for other morale-enhancing measures. The same principle must be applied to all other tables in your design.

atomic attribute
An attribute that cannot be further subdivided to produce meaningful components. For example, a person's last name attribute cannot be meaningfully subdivided.

atomicity
Not being able to be divided into smaller units.

granularity
The level of detail represented by the values stored in a table's row. Data stored at its lowest level of granularity is said to be *atomic data*.

Identify New Relationships According to the original report, the users need to track which employee is acting as the manager of each project. This can be implemented as a relationship between EMPLOYEE and PROJECT. From the original report, it is clear that each project has only one manager. Therefore, the system's ability to supply detailed information about each project's manager is ensured by using the EMP_NUM as a foreign key in PROJECT. That action ensures that you can access the details of each PROJECT's manager data without producing unnecessary and undesirable data duplication. The designer must take care to place the right attributes in the right tables by using normalization principles.

Refine Primary Keys as Required for Data Granularity **Granularity** refers to the level of detail represented by the values stored in a table's row. Data stored at its lowest level of granularity is said to be *atomic data*, as explained earlier. In Figure 6.5, the ASSIGNMENT table in 3NF uses the ASSIGN_HOURS attribute to represent the hours worked by a given employee on a given project. However, are those values recorded at

their lowest level of granularity? In other words, does ASSIGN_HOURS represent the *hourly* total, *daily* total, *weekly* total, *monthly* total, or *yearly* total? Clearly, ASSIGN_HOURS requires more careful definition. In this case, the relevant question would be as follows: for what time frame—hour, day, week, month, and so on—do you want to record the ASSIGN_HOURS data?

For example, assume that the combination of EMP_NUM and PROJ_NUM is an acceptable (composite) primary key in the ASSIGNMENT table. That primary key is useful in representing only the total number of hours an employee worked on a project since its start. Using a surrogate primary key such as ASSIGN_NUM provides lower granularity and yields greater flexibility. For example, assume that the EMP_NUM and PROJ_NUM combination is used as the primary key, and then an employee makes two "hours worked" entries in the ASSIGNMENT table. That action violates the entity integrity requirement. Even if you add the ASSIGN_DATE as part of a composite PK, an entity integrity violation is still generated if any employee makes two or more entries for the same project on the same day. (The employee might have worked on the project for a few hours in the morning and then worked on it again later in the day.) The same data entry yields no problems when ASSIGN_NUM is used as the primary key.

Note

In an ideal database design, the level of desired granularity would be determined during the conceptual design or while the requirements were being gathered. However, as you have already seen in this chapter, many database designs involve the refinement of existing data requirements, thus triggering design modifications. In a real-world environment, changing granularity requirements might dictate changes in primary key selection, and those changes might ultimately require the use of surrogate keys.

Maintain Historical Accuracy Writing the job charge per hour into the ASSIGNMENT table is crucial to maintaining the historical accuracy of the table's data. It would be appropriate to name this attribute ASSIGN_CHG_HOUR. Although this attribute would appear to have the same value as JOB_CHG_HOUR, this is true *only* if the JOB_CHG_HOUR value remains the same forever. It is reasonable to assume that the job charge per hour will change over time. However, suppose that the charges to each project were calculated and billed by multiplying the hours worked from the ASSIGNMENT table by the charge per hour from the JOB table. Those charges would always show the current charge per hour stored in the JOB table rather than the charge per hour that was in effect at the time of the assignment.

Evaluate Using Derived Attributes Finally, you can use a derived attribute in the ASSIGNMENT table to store the actual charge made to a project. That derived attribute, named ASSIGN_CHARGE, is the result of multiplying ASSIGN_HOURS by ASSIGN_CHG_HOUR. This creates a transitive dependency such that:

(ASSIGN_CHARGE + ASSIGN_HOURS) → ASSIGN_CHG_HOUR

From a system functionality point of view, such derived attribute values can be calculated when they are needed to write reports or invoices. However, storing the derived attribute in the table makes it easy to write the application software to produce the desired results. Also, if many transactions must be reported and/or summarized, the availability

of the derived attribute will save reporting time. (If the calculation is done at the time of data entry, it will be completed when the end user presses the Enter key, thus speeding up the process.) Review Chapter 4 for a discussion of the implications of storing derived attributes in a database table.

The enhancements described in the preceding sections are illustrated in the tables and dependency diagrams shown in Figure 6.6.

FIGURE 6.6 THE COMPLETED DATABASE

Table name: PROJECT

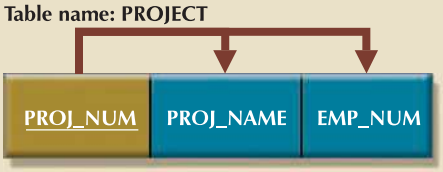


Table name: JOB Database name: Ch06_ConstructCo

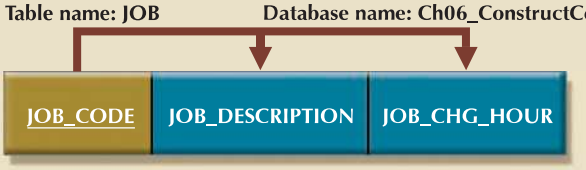


Table name: PROJECT

PROJ_NUM	PROJ_NAME	EMP_NUM
15	Evergreen	105
18	Amber Wave	104
22	Rolling Tide	113
25	Starflight	101

Table name: JOB

JOB_CODE	JOB_DESCRIPTION	JOB_CHG_HOUR
500	Programmer	35.75
501	Systems Analyst	96.75
502	Database Designer	105.00
503	Electrical Engineer	84.50
504	Mechanical Engineer	67.90
505	Civil Engineer	55.78
506	Clerical Support	26.87
507	DSS Analyst	45.95
508	Applications Designer	48.10
509	Bio Technician	34.55
510	General Support	18.36

Table name: ASSIGNMENT

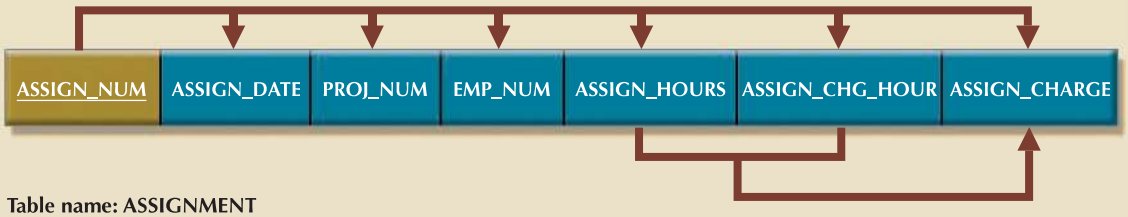


Table name: ASSIGNMENT

ASSIGN_NUM	ASSIGN_DATE	PROJ_NUM	EMP_NUM	ASSIGN_HOURS	ASSIGN_CHG_HOUR	ASSIGN_CHARGE
1001	04-Mar-16	15	103	2.6	84.50	219.70
1002	04-Mar-16	18	118	1.4	18.36	25.70
1003	05-Mar-16	15	101	3.6	105.00	378.00
1004	05-Mar-16	22	113	2.5	48.10	120.25
1005	05-Mar-16	15	103	1.9	84.50	160.55
1006	05-Mar-16	25	115	4.2	96.75	406.35
1007	05-Mar-16	22	105	5.2	105.00	546.00
1008	05-Mar-16	25	101	1.7	105.00	178.50
1009	05-Mar-16	15	105	2.0	105.00	210.00
1010	06-Mar-16	15	102	3.8	96.75	367.65
1011	06-Mar-16	22	104	2.6	96.75	251.55
1012	06-Mar-16	15	101	2.3	105.00	241.50
1013	06-Mar-16	25	114	1.8	48.10	86.58
1014	06-Mar-16	22	111	4.0	26.87	107.48
1015	06-Mar-16	25	114	3.4	48.10	163.54
1016	06-Mar-16	18	112	1.2	45.95	55.14
1017	06-Mar-16	18	118	2.0	18.36	36.72
1018	06-Mar-16	18	104	2.6	96.75	251.55
1019	06-Mar-16	15	103	3.0	84.50	253.50
1020	07-Mar-16	22	105	2.7	105.00	283.50
1021	08-Mar-16	25	108	4.2	96.75	406.35
1022	07-Mar-16	25	114	5.8	48.10	278.98
1023	07-Mar-16	22	106	2.4	35.75	85.80

FIGURE 6.6 THE COMPLETED DATABASE (CONTINUED)

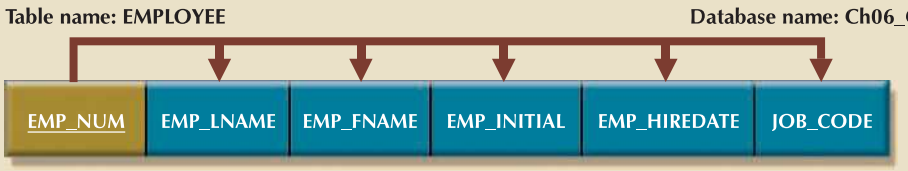


Table name: EMPLOYEE

EMP_NUM	EMP_LNAME	EMP_FNAME	EMP_INITIAL	EMP_HIREDATE	JOB_CODE
101	News	John	G	08-Nov-00	502
102	Senior	David	H	12-Jul-89	501
103	Arbough	June	E	01-Dec-97	503
104	Ramoras	Anne	K	15-Nov-88	501
105	Johnson	Alice	K	01-Feb-94	502
106	Smithfield	William		22-Jun-05	500
107	Alonzo	Maria	D	10-Oct-94	500
108	Washington	Ralph	B	22-Aug-89	501
109	Smith	Larry	W	18-Jul-99	501
110	Olenko	Gerald	A	11-Dec-96	505
111	Wabash	Geoff	B	04-Apr-89	506
112	Smithson	Darlene	M	23-Oct-95	507
113	Joenbrood	Delbert	K	15-Nov-94	508
114	Jones	Annelise		20-Aug-91	508
115	Bawangi	Travis	B	25-Jan-90	501
116	Pratt	Gerald	L	05-Mar-95	510
117	Williamson	Angie	H	19-Jun-94	509
118	Frommer	James	J	04-Jan-06	510

Figure 6.6 is a vast improvement over the original database design. If the application software is designed properly, the most active table (ASSIGNMENT) requires the entry of only the PROJ_NUM, EMP_NUM, and ASSIGN_HOURS values. The values for the attributes ASSIGN_NUM and ASSIGN_DATE can be generated by the application. For example, the ASSIGN_NUM can be created by using a counter, and the ASSIGN_DATE can be the system date read by the application and automatically entered into the ASSIGNMENT table. In addition, the application software can automatically insert the correct ASSIGN_CHG_HOUR value by writing the appropriate JOB table's JOB_CHG_HOUR value into the ASSIGNMENT table. (The JOB and ASSIGNMENT tables are related through the JOB_CODE attribute.) If the JOB table's JOB_CHG_HOUR value changes, the next insertion of that value into the ASSIGNMENT table will reflect the change automatically. The table structure thus minimizes the need for human intervention. In fact, if the system requires the employees to enter their own work hours, they can scan their EMP_NUM into the ASSIGNMENT table by using a magnetic card reader that enters their identity. Thus, the ASSIGNMENT table's structure can set the stage for maintaining some desired level of security.

6-5 Surrogate Key Considerations

Although this design meets the vital entity and referential integrity requirements, the designer must still address some concerns. For example, a composite primary key might become too cumbersome to use as the number of attributes grows. (It becomes difficult to create a suitable foreign key when the related table uses a composite primary key. In addition, a composite primary key makes it more difficult to write search routines.) Or, a primary key attribute might simply have too much descriptive content to be usable—which is why the JOB_CODE attribute was added to the JOB table to serve as its primary key. When the primary key is considered to be unsuitable for some reason, designers use surrogate keys, as discussed in the previous chapter.

At the implementation level, a surrogate key is a system-defined attribute generally created and managed via the DBMS. Usually, a system-defined surrogate key is numeric, and its value is automatically incremented for each new row. For example, Microsoft Access uses an AutoNumber data type, Microsoft SQL Server uses an identity column, and Oracle uses a sequence object.

Recall from Section 6-4 that the JOB_CODE attribute was designated to be the JOB table's primary key. However, remember that the JOB_CODE attribute does not prevent duplicate entries, as shown in the JOB table in Table 6.4.

TABLE 6.4

DUPLICATE ENTRIES IN THE JOB TABLE

JOB_CODE	JOB_DESCRIPTION	JOB_CHG_HOUR
511	Programmer	$35.75
512	Programmer	$35.75

Clearly, the data entries in Table 6.4 are inappropriate because they duplicate existing records—yet there has been no violation of either entity integrity or referential integrity. This problem of multiple duplicate records was created when the JOB_CODE attribute was added as the PK. (When the JOB_DESCRIPTION was initially designated to be the PK, the DBMS would ensure unique values for all job description entries when it was asked to enforce entity integrity. However, that option created the problems that caused the use of the JOB_CODE attribute in the first place!) In any case, if JOB_CODE is to be the surrogate PK, you still must ensure the existence of unique values in the JOB_DESCRIPTION *through the use of a unique index*.

Note that all of the remaining tables (PROJECT, ASSIGNMENT, and EMPLOYEE) are subject to the same limitations. For example, if you use the EMP_NUM attribute in the EMPLOYEE table as the PK, you can make multiple entries for the same employee. To avoid that problem, you might create a unique index for EMP_LNAME, EMP_FNAME, and EMP_INITIAL, but how would you then deal with two employees named Joe B. Smith? In that case, you might use another (preferably externally defined) attribute to serve as the basis for a unique index.

It is worth repeating that database design often involves trade-offs and the exercise of professional judgment. In a real-world environment, you must strike a balance between design integrity and flexibility. For example, you might design the ASSIGNMENT table to use a unique index on PROJ_NUM, EMP_NUM, and ASSIGN_DATE if you want to limit an employee to only one ASSIGN_HOURS entry per date. That limitation would ensure that employees could not enter the same hours multiple times for any given date. Unfortunately, that limitation is likely to be undesirable from a managerial point of view. After all, if an employee works several different times on a project during any given day, it must be possible to make multiple entries for that same employee and the same project during that day. In that case, the best solution might be to add a new externally defined attribute—such as a stub, voucher, or ticket number—to ensure uniqueness. In any case, frequent data audits would be appropriate.

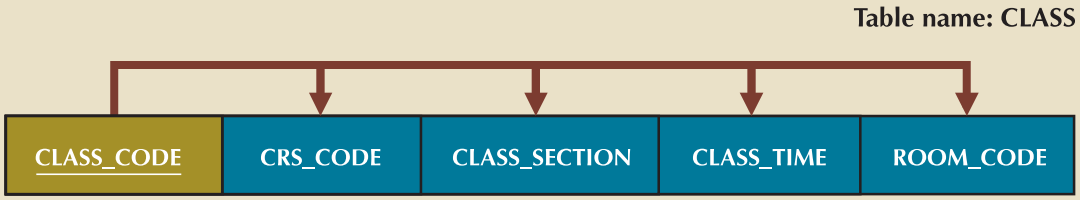
6-6 Higher-Level Normal Forms

Tables in 3NF will perform suitably in business transactional databases. However, higher normal forms are sometimes useful. In this section, you will learn about a special case of 3NF, known as Boyce-Codd normal form, and about fourth normal form (4NF).

6-6a The Boyce-Codd Normal Form

A table is in Boyce-Codd normal form (BCNF) when every determinant in the table is a candidate key. (Recall from Chapter 3 that a candidate key has the same characteristics as a primary key, but for some reason, it was not chosen to be the primary key.) Clearly, when a table contains only one candidate key, the 3NF and the BCNF are equivalent. In other words, BCNF can be violated only when the table contains more than one candidate key. In the previous normal form examples, tables with only one candidate key were used to simplify the explanations. Remember, however, that multiple candidate keys are always possible, and normalization rules focus on candidate keys, not just the primary key. Consider the table structure shown in Figure 6.7.

FIGURE 6.7 TABLES WITH MULTIPLE CANDIDATE KEYS



The CLASS table has two candidate keys:

- CLASS_CODE
- CRS_CODE + CLASS_SECTION

The table is in 1NF because the key attributes are defined and all nonkey attributes are determined by the key. This is true for both candidate keys. Both candidate keys have been identified, and all of the other attributes can be determined by either candidate key. The table is in 2NF because it is in 1NF and there are no partial dependencies on either candidate key. Since CLASS_CODE is a single attribute candidate key, the issue of partial dependencies doesn't apply. However, the composite candidate key of CRS_CODE + CLASS_SECTION could potentially have a partial dependency so 2NF must be evaluated for that candidate key. In this case, there are no partial dependencies involving the composite key. Finally, the table is in 3NF because there are no transitive dependencies. Remember, because CRS_CODE + CLASS_SECTION is a candidate key, the fact that this composite can determine the CLASS_TIME and ROOM_CODE is not a transitive dependency. A transitive dependency exists when a *nonkey* attribute can determine another nonkey attribute, and CRS_CODE + CLASS_SECTION is a key.

Note

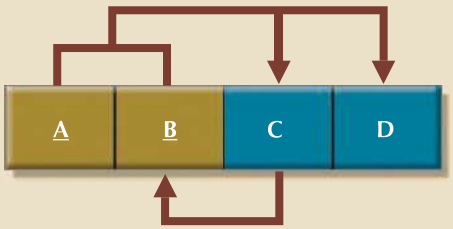
A table is in **Boyce-Codd normal form (BCNF)** when every determinant in the table is a candidate key.

Boyce-Codd normal form (BCNF)
A special type of third normal form (3NF) in which every determinant is a candidate key. A table in BCNF must be in 3NF. See also *determinant*.

Most designers consider the BCNF to be a special case of the 3NF. In fact, if the techniques shown in this chapter are used, most tables conform to the BCNF requirements once the 3NF is reached. So, how can a table be in 3NF and not be in BCNF? To answer that question, you must keep in mind that a transitive dependency exists when one nonprime attribute is dependent on another nonprime attribute.

In other words, a table is in 3NF when it is in 2NF and there are no transitive dependencies, but what about a case in which one key attribute is the determinant of another key attribute? That condition does not violate 3NF, yet it fails to meet the BCNF requirements (see Figure 6.8) because BCNF requires that every determinant in the table be a candidate key.

FIGURE 6.8 A TABLE THAT IS IN 3NF BUT NOT IN BCNF



Note these functional dependencies in Figure 6.8:

A + B → C, D

A + C → B, D

C → B

Notice that this structure has two candidate keys: (A + B) and (A + C). The table structure shown in Figure 6.8 has no partial dependencies, nor does it contain transitive dependencies. (The condition C → B indicates that *one key attribute determines part of the primary key*—and *that* dependency *is not* transitive or partial because the dependent is a prime attribute!) Thus, the table structure in Figure 6.8 meets the 3NF requirements, although the condition C → B causes the table to fail to meet the BCNF requirements.

To convert the table structure in Figure 6.8 into table structures that are in 3NF and in BCNF, first change the primary key to A + C. This change is appropriate because the dependency C → B means that C is effectively a superset of B. At this point, the table is in 1NF because it contains a partial dependency, C → B. Next, follow the standard decomposition procedures to produce the results shown in Figure 6.9.

To see how this procedure can be applied to an actual problem, examine the sample data in Table 6.5.

Table 6.5 reflects the following conditions:

- Each CLASS_CODE identifies a class uniquely. This condition illustrates the case in which a course might generate many classes. For example, a course labeled INFS 420 might be taught in two classes (sections), each identified by a unique code to facilitate registration. Thus, the CLASS_CODE 32456 might identify INFS 420, class section 1, while the CLASS_CODE 32457 might identify INFS 420, class section 2. Or, the CLASS_CODE 28458 might identify QM 362, class section 5.
- A student can take many classes. Note, for example, that student 125 has taken both 21334 and 32456, earning the grades A and C, respectively.
- A staff member can teach many classes, but each class is taught by only one staff member. Note that staff member 20 teaches the classes identified as 32456 and 28458.

The structure shown in Table 6.5 is reflected in Panel A of Figure 6.10:

STU_ID + STAFF_ID → CLASS_CODE, ENROLL_GRADE

CLASS_CODE → STAFF_ID

FIGURE 6.9 DECOMPOSITION TO BCNF

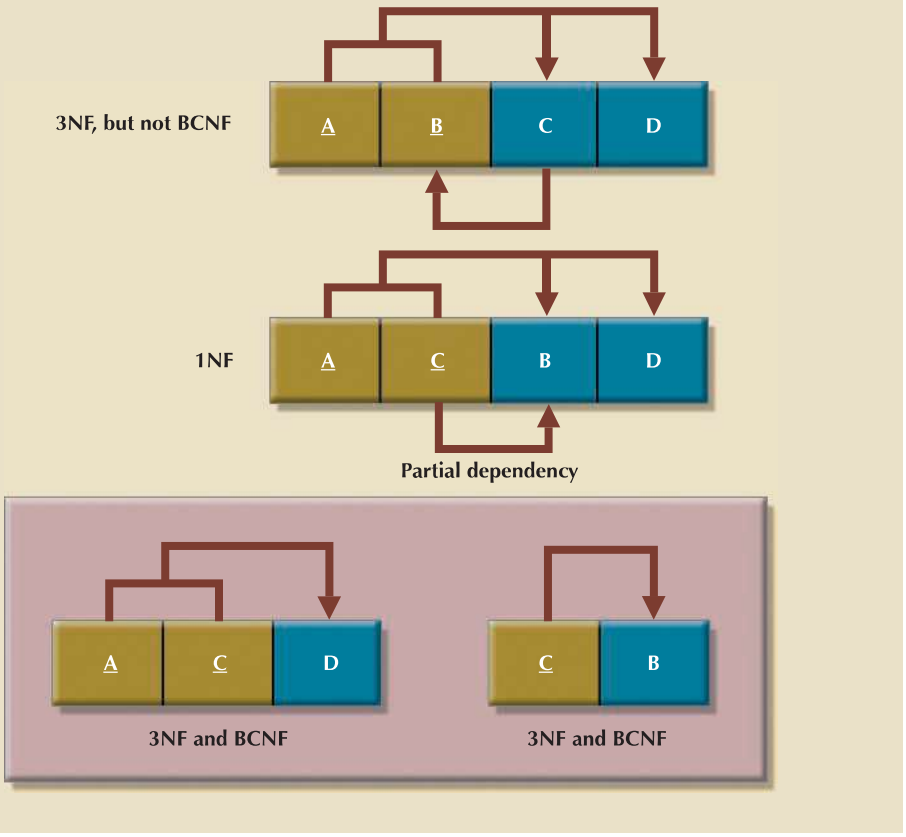


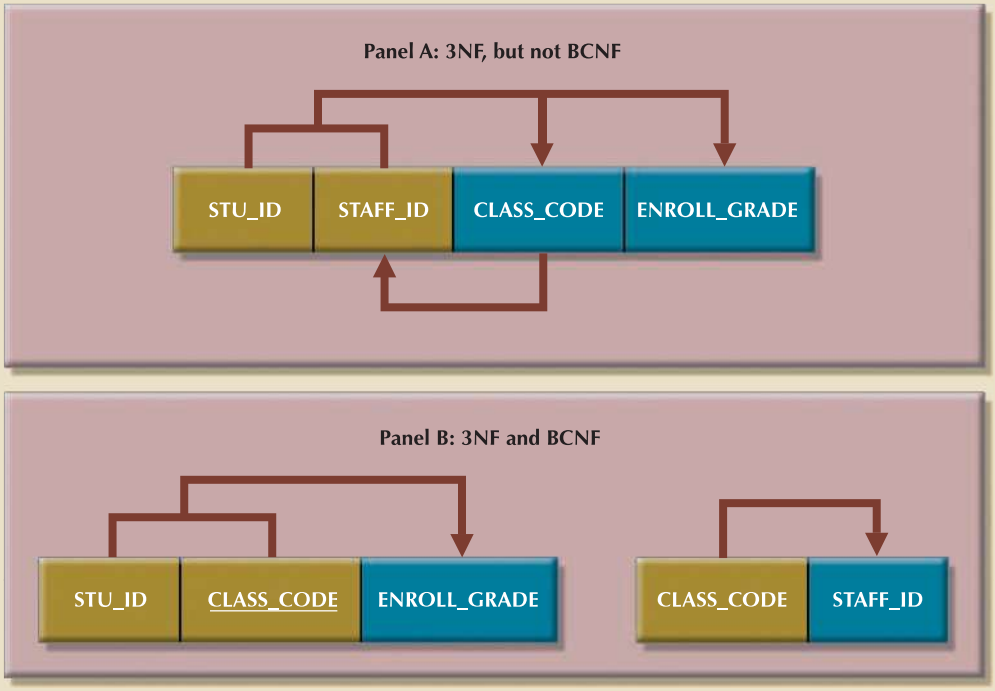
TABLE 6.5

SAMPLE DATA FOR A BCNF CONVERSION

STU_ID	STAFF_ID	CLASS_CODE	ENROLL_GRADE
125	25	21334	A
125	20	32456	C
135	20	28458	B
144	25	27563	C
144	20	32456	B

Panel A of Figure 6.10 shows a structure that is clearly in 3NF, but the table represented by this structure has a major problem because it is trying to describe two things: staff assignments to classes and student enrollment information. Such a dual-purpose table structure will cause anomalies. For example, if a different staff member is assigned to teach class 32456, two rows will require updates, thus producing an update anomaly. Also, if student 135 drops class 28458, information about who taught that class is lost, thus producing a deletion anomaly. The solution to the problem is to decompose the table structure, following the procedure outlined earlier. The decomposition of Panel B shown in Figure 6.10 yields two table structures that conform to both 3NF and BCNF requirements.

FIGURE 6.10 ANOTHER BCNF DECOMPOSITION



Remember that a table is in BCNF when every determinant in that table is a candidate key. Therefore, when a table contains only one candidate key, 3NF and BCNF are equivalent.

6-6b Fourth Normal Form (4NF)

You might encounter poorly designed databases, or you might be asked to convert spreadsheets into a database format in which multiple multivalued attributes exist. For example, consider the possibility that an employee can have multiple assignments and can also be involved in multiple service organizations. Suppose employee 10123 volunteers for the Red Cross and United Way. In addition, the same employee might be assigned to work on three projects: 1, 3, and 4. Figure 6.11 illustrates how that set of facts can be recorded in very different ways.

FIGURE 6.11 TABLES WITH MULTIVALUED DEPENDENCIES

Database name: Ch06_Service

Table name: VOLUNTEER_V1

EMP_NUM	ORG_CODE	ASSIGN_NUM
10123	RC	1
10123	UW	3
10123		4

Table name: VOLUNTEER_V2

EMP_NUM	ORG_CODE	ASSIGN_NUM
10123	RC	
10123	UW	
10123		1
10123		3
10123		4

Table name: VOLUNTEER_V3

EMP_NUM	ORG_CODE	ASSIGN_NUM
10123	RC	1
10123	RC	3
10123	UW	4

There is a problem with the tables in Figure 6.11. The attributes ORG_CODE and ASSIGN_NUM each may have many different values. In normalization terminology, this situation is referred to as a multivalued dependency, which occurs when one key determines multiple values of two other attributes and those attributes are independent of each other. (One employee can have many service entries and many assignment entries. Therefore, one EMP_NUM can determine multiple values of ORG_CODE and multiple values of ASSIGN_NUM; however, ORG_CODE and ASSIGN_NUM are independent of each other.) The presence of a multivalued dependency means that if table versions 1 and 2 are implemented, the tables are likely to contain quite a few null values; in fact, the tables do not even have a viable candidate key. (The EMP_NUM values are not unique, so they cannot be PKs. No combination of the attributes in table versions 1 and 2 can be used to create a PK because some of them contain nulls.) Such a condition is not desirable, especially when there are thousands of employees, many of whom may have multiple job assignments and many service activities. Version 3 at least has a PK, but it is composed of all the attributes in the table. In fact, version 3 meets 3NF requirements, yet it contains many redundancies that are clearly undesirable.

The solution is to eliminate the problems caused by the multivalued dependency. You do this by creating new tables for the components of the multivalued dependency. In this example, the multivalued dependency is resolved and eliminated by creating the ASSIGNMENT and SERVICE_V1 tables depicted in Figure 6.12. Those tables are said to be in 4NF.

FIGURE 6.12 A SET OF TABLES IN 4NF

Database name: CH06_Service

Table name: PROJECT

PROJ_CODE	PROJ_NAME	PROJ_BUDGET
1	BeThere	1023245.00
2	BlueMoon	20198608.00
3	GreenThumb	3234456.00
4	GoFast	5674000.00
5	GoSlow	1002500.00

Table name: EMPLOYEE

EMP_NUM	EMP_LNAME
10121	Rogers
10122	O'Leery
10123	Panera
10124	Johnson

Table name: ASSIGNMENT

ASSIGN_NUM	EMP_NUM	PROJ_CODE
1	10123	1
2	10121	2
3	10123	3
4	10123	4
5	10121	1
6	10124	2
7	10124	3
8	10124	5

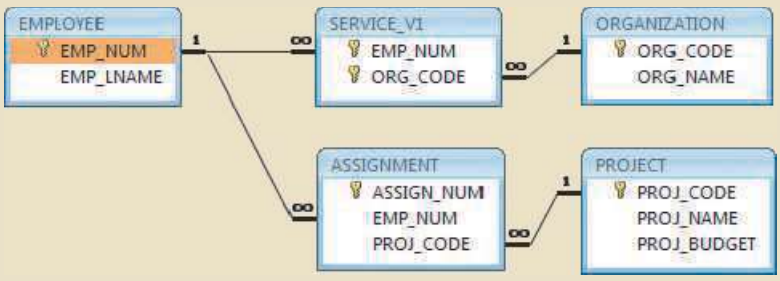
Table name: ORGANIZATION

ORG_CODE	ORG_NAME
RC	Red Cross
UW	United Way
WF	Wildlife Fund

Table name: SERVICE_V1

EMP_NUM	ORG_CODE
10123	RC
10123	UW
10123	WF

The relational diagram



If you follow the proper design procedures illustrated in this book, you should not encounter the problem shown in Figure 6.11. Specifically, the discussion of 4NF is largely academic if you make sure that your tables conform to the following two rules:

1. All attributes must be dependent on the primary key, but they must be independent of each other.
2. No row may contain two or more multivalued facts about an entity.

Note

A table is in **fourth normal form (4NF)** when it is in 3NF and has no multivalued dependencies.

6-7 Normalization and Database Design

The tables shown in Figure 6.6 illustrate how normalization procedures can be used to produce good tables from poor ones. You will likely have ample opportunity to put this skill into practice when you begin to work with real-world databases. *Normalization should be part of the design process*. Therefore, make sure that proposed entities meet the required normal form *before* the table structures are created. Keep in mind that if you follow the design procedures discussed in Chapters 3 and 4, the likelihood of data anomalies will be small. However, even the best database designers are known to make occasional mistakes that come to light during normalization checks. Also, many of the real-world databases you encounter will have been improperly designed or burdened with anomalies if they were improperly modified over the course of time. That means you might be asked to redesign and modify existing databases that are, in effect, anomaly traps. Therefore, you should be aware of good design principles and procedures as well as normalization procedures.

First, an ERD is created through an iterative process. You begin by identifying relevant entities, their attributes, and their relationships. Then you use the results to identify additional entities and attributes. The ERD provides the big picture, or macro view, of an organization's data requirements and operations.

Second, normalization focuses on the characteristics of specific entities; that is, normalization represents a micro view of the entities within the ERD. Also, as you learned in the previous sections of this chapter, the normalization process might yield additional entities and attributes to be incorporated into the ERD. Therefore, it is difficult to separate normalization from ER modeling; the two techniques are used in an iterative and incremental process.

To understand the proper role of normalization in the design process, you should reexamine the operations of the contracting company whose tables were normalized in the preceding sections. Those operations can be summarized by using the following business rules:

- The company manages many projects.
- Each project requires the services of many employees.
- An employee may be assigned to several different projects.
- Some employees are not assigned to a project and perform duties not specifically related to a project. Some employees are part of a labor pool, to be shared by all project teams. For example, the company's executive secretary would not be assigned to any one particular project.

fourth normal form (4NF)
A table is in 4NF if it is in 3NF and contains no multiple independent sets of multivalued dependencies.

- Each employee has a single primary job classification, which determines the hourly billing rate.
- Many employees can have the same job classification. For example, the company employs more than one electrical engineer.

Given that simple description of the company's operations, two entities and their attributes are initially defined:

- PROJECT (**PROJ_NUM**, PROJ_NAME)
- EMPLOYEE (**EMP_NUM**, EMP_LNAME, EMP_FNAME, EMP_INITIAL, JOB_DESCRIPTION, JOB_CHG_HOUR)

Those two entities constitute the initial ERD shown in Figure 6.13.

FIGURE 6.13 INITIAL CONTRACTING COMPANY ERD

After creating the initial ERD shown in Figure 6.13, the normal forms are defined:

- PROJECT is in 3NF and needs no modification at this point.
- EMPLOYEE requires additional scrutiny. The JOB_DESCRIPTION attribute defines job classifications such as Systems Analyst, Database Designer, and Programmer. In turn, those classifications determine the billing rate, JOB_CHG_HOUR. Therefore, EMPLOYEE contains a transitive dependency.

The removal of EMPLOYEE's transitive dependency yields three entities:

- PROJECT (**PROJ_NUM**, PROJ_NAME)
- EMPLOYEE (**EMP_NUM**, EMP_LNAME, EMP_FNAME, EMP_INITIAL, JOB_CODE)
- JOB (**JOB_CODE**, JOB_DESCRIPTION, JOB_CHG_HOUR)

Because the normalization process yields an additional entity (JOB), the initial ERD is modified as shown in Figure 6.14.

To represent the M:N relationship between EMPLOYEE and PROJECT, you might think that two 1:M relationships could be used—an employee can be assigned to many projects, and each project can have many employees assigned to it. (See Figure 6.15.) Unfortunately, that representation yields a design that cannot be correctly implemented.

Because the M:N relationship between EMPLOYEE and PROJECT cannot be implemented, the ERD in Figure 6.15 must be modified to include the ASSIGNMENT entity to track the assignment of employees to projects, thus yielding the ERD shown in Figure 6.16. The ASSIGNMENT entity in Figure 6.16 uses the primary keys from the entities PROJECT and EMPLOYEE to serve as its foreign keys. However, note that in this implementation, the ASSIGNMENT entity's surrogate primary key is ASSIGN_NUM, to avoid the use of a composite primary key. Therefore, the "enters" relationship between EMPLOYEE and ASSIGNMENT and the "requires" relationship between PROJECT and ASSIGNMENT are shown as weak or nonidentifying.

FIGURE 6.14 MODIFIED CONTRACTING COMPANY ERD

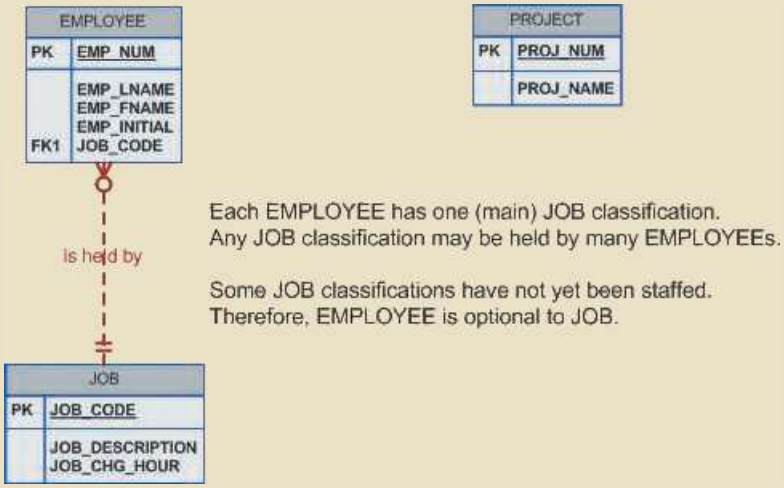
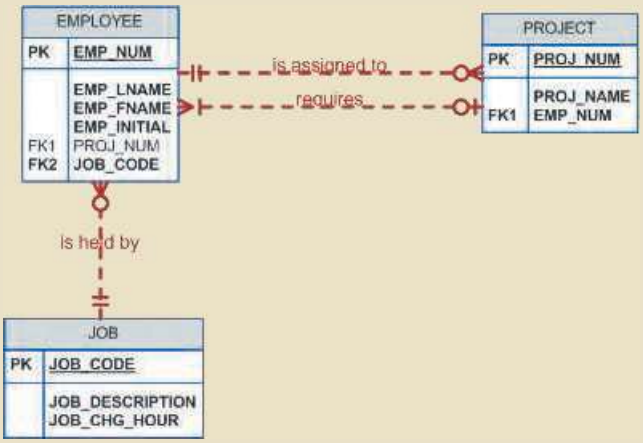


FIGURE 6.15 INCORRECT M:N RELATIONSHIP REPRESENTATION



In Figure 6.16, the ASSIGN_HOURS attribute is assigned to the composite entity named ASSIGNMENT. Because you will likely need detailed information about each project's manager, the creation of a "manages" relationship is useful. The "manages" relationship is implemented through the foreign key in PROJECT. Finally, some additional attributes may be created to improve the system's ability to generate additional information. For example, you may want to include the date the employee was hired (EMP_HIREDATE) to keep track of worker longevity. Based on this last modification, the model should include four entities and their attributes:

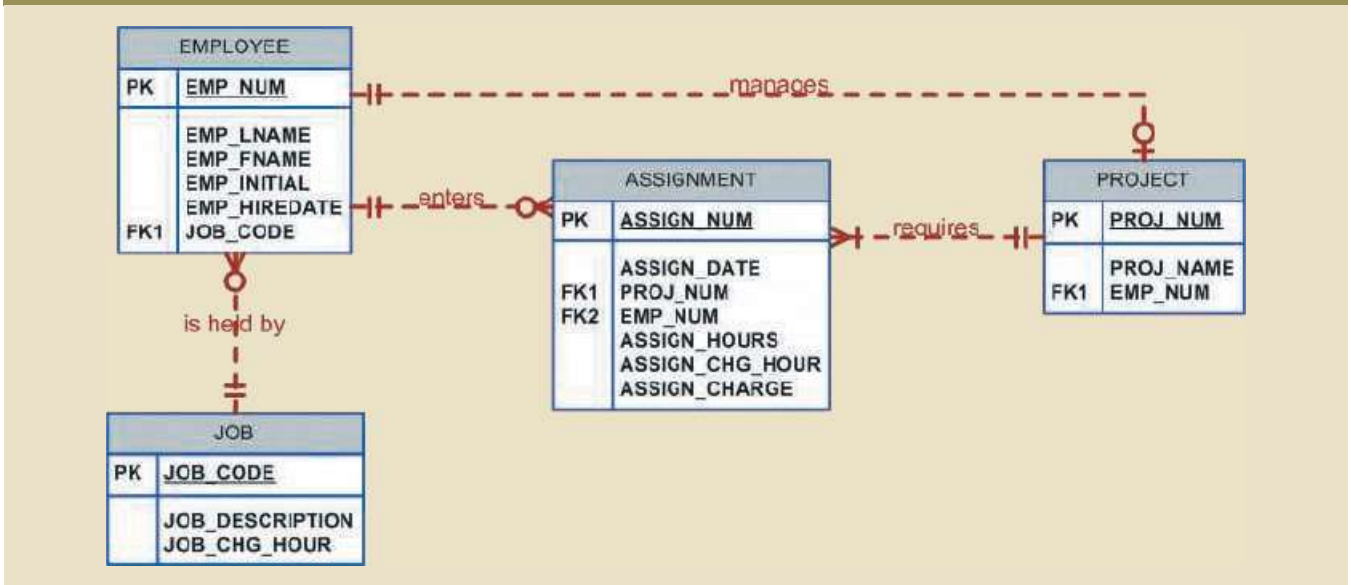
PROJECT (**PROJ_NUM**, PROJ_NAME, EMP_NUM)

EMPLOYEE (**EMP_NUM**, EMP_LNAME, EMP_FNAME, EMP_INITIAL, EMP_HIREDATE, JOB_CODE)

JOB (**JOB_CODE**, JOB_DESCRIPTION, JOB_CHG_HOUR)

ASSIGNMENT (**ASSIGN_NUM**, ASSIGN_DATE, PROJ_NUM, EMP_NUM, ASSIGN_HOURS, ASSIGN_CHG_HOUR, ASSIGN_CHARGE)

FIGURE 6.16 FINAL CONTRACTING COMPANY ERD



The design process is now on the right track. The ERD represents the operations accurately, and the entities now reflect their conformance to 3NF. The combination of normalization and ER modeling yields a useful ERD, whose entities may now be translated into appropriate table structures. In Figure 6.15, note that PROJECT is optional to EMPLOYEE in the "manages" relationship. This optionality exists because not all employees manage projects. The final database contents are shown in Figure 6.17.

6-8 Denormalization

It is important to remember that the optimal relational database implementation requires that all tables be at least in third normal form (3NF). A good relational DBMS excels at managing normalized relations—that is, relations void of any unnecessary redundancies that might cause data anomalies. Although the creation of normalized relations is an important database design goal, it is only one of many such goals. Good database design also considers processing (or reporting) requirements and processing speed. The problem with normalization is that as tables are decomposed to conform to normalization requirements, the number of database tables expands. Therefore, in order to generate information, data must be put together from various tables. Joining a large number of tables takes additional input/output (I/O) operations and processing logic, thereby reducing system speed. Most relational database systems are able to handle joins very efficiently. However, rare and occasional circumstances may allow some degree of denormalization so processing speed can be increased.

Keep in mind that the advantage of higher processing speed must be carefully weighed against the disadvantage of data anomalies. On the other hand, some anomalies are of only theoretical interest. For example, should people in a real-world database environment worry that a ZIP_CODE determines CITY in a CUSTOMER table whose primary key is the customer number? Is it really practical to produce a separate table for

ZIP (**ZIP_CODE**, CITY)

to eliminate a transitive dependency from the CUSTOMER table? (Perhaps your answer to that question changes if you are in the business of producing mailing lists.) As explained earlier, the problem with denormalized relations and redundant data is that data integrity could be compromised due to the possibility of insert, update, and

FIGURE 6.17 THE IMPLEMENTED DATABASE

Table name: EMPLOYEE

Database name: Ch06_ConstructCo

EMP_NUM	EMP_LNAME	EMP_FNAME	EMP_INITIAL	EMP_HIREDATE	JOB_CODE
101	News	John	G	08-Nov-00	502
102	Senior	David	H	12-Jul-89	501
103	Arbough	June	E	01-Dec-97	503
104	Ramoras	Anne	K	15-Nov-88	501
105	Johnson	Alice	K	01-Feb-94	502
106	Smithfield	William		22-Jun-05	500
107	Alonzo	Maria	D	10-Oct-94	500
108	Washington	Ralph	B	22-Aug-89	501
109	Smith	Larry	W	18-Jul-99	501
110	Olenko	Gerald	A	11-Dec-96	505
111	Wabash	Geoff	B	04-Apr-89	506
112	Smithson	Darlene	M	23-Oct-95	507
113	Joenbrood	Delbert	K	15-Nov-94	508
114	Jones	Annelise		20-Aug-91	508
115	Bawangi	Travis	B	25-Jan-90	501
116	Pratt	Gerald	L	05-Mar-95	510
117	Williamson	Angie	H	19-Jun-94	509
118	Frommer	James	J	04-Jan-06	510

Table name: JOB

JOB_CODE	JOB_DESCRIPTION	JOB_CHG_HOUR
500	Programmer	35.75
501	Systems Analyst	96.75
502	Database Designer	105.00
503	Electrical Engineer	84.50
504	Mechanical Engineer	67.90
505	Civil Engineer	55.78
506	Clerical Support	26.87
507	DSS Analyst	45.95
508	Applications Designer	48.10
509	Bio Technician	34.55
510	General Support	18.36

Table name: PROJECT

PROJ_NUM	PROJ_NAME	EMP_NUM
15	Evergreen	105
18	Amber Wave	104
22	Rolling Tide	113
25	Starflight	101

Table name: ASSIGNMENT

ASSIGN_NUM	ASSIGN_DATE	PROJ_NUM	EMP_NUM	ASSIGN_HOURS	ASSIGN_CHG_HOUR	ASSIGN_CHARGE
1001	04-Mar-16	15	103	2.6	84.50	219.70
1002	04-Mar-16	18	118	1.4	18.36	25.70
1003	05-Mar-16	15	101	3.6	105.00	378.00
1004	05-Mar-16	22	113	2.5	48.10	120.25
1005	05-Mar-16	15	103	1.9	84.50	160.55
1006	05-Mar-16	25	115	4.2	96.75	406.35
1007	05-Mar-16	22	105	5.2	105.00	546.00
1008	05-Mar-16	25	101	1.7	105.00	178.50
1009	05-Mar-16	15	105	2.0	105.00	210.00
1010	06-Mar-16	15	102	3.8	96.75	367.65
1011	06-Mar-16	22	104	2.6	96.75	251.55
1012	06-Mar-16	15	101	2.3	105.00	241.50
1013	06-Mar-16	25	114	1.8	48.10	86.58
1014	06-Mar-16	22	111	4.0	26.87	107.48
1015	06-Mar-16	25	114	3.4	48.10	163.54
1016	06-Mar-16	18	112	1.2	45.95	55.14
1017	06-Mar-16	18	118	2.0	18.36	36.72
1018	06-Mar-16	18	104	2.6	96.75	251.55
1019	06-Mar-16	15	103	3.0	84.50	253.50
1020	07-Mar-16	22	105	2.7	105.00	283.50
1021	08-Mar-16	25	108	4.2	96.75	406.35
1022	07-Mar-16	25	114	5.8	48.10	278.98
1023	07-Mar-16	22	106	2.4	35.75	85.80

deletion anomalies. The advice is simple: use common sense during the normalization process.

Furthermore, the database design process could, in some cases, introduce some small degree of redundant data in the model, as seen in the previous example. This, in effect, creates "denormalized" relations. Table 6.6 shows some common examples of data redundancy that are generally found in database implementations.

A more comprehensive example of the need for denormalization due to reporting requirements is the case of a faculty evaluation report in which each row lists the scores obtained during the last four semesters taught. (See Figure 6.18.)

Although this report seems simple enough, the problem is that the data is stored in a normalized table in which each row represents a different score for a given faculty member in a given semester. (See Figure 6.19.)

The difficulty of transposing multirow data to multicolumn data is compounded by the fact that the last four semesters taught are not necessarily the same for all faculty members. Some might have taken sabbaticals, some might have had research appointments, some might be new faculty with only two semesters on the job, and so on. To generate this report, the two tables in Figure 6.18 were used. The EVALDATA table is

TABLE 6.6

COMMON DENORMALIZATION EXAMPLES

CASE	EXAMPLE	RATIONALE AND CONTROLS
Redundant data	Storing ZIP and CITY attributes in the AGENT table when ZIP determines CITY (see Figure 2.2)	Avoid extra join operations Program can validate city (drop-down box) based on the zip code
Derived data	Storing STU_HRS and STU_CLASS (student classification) when STU_HRS determines STU_CLASS (see Figure 3.28)	Avoid extra join operations Program can validate classification (lookup) based on the student hours
Preaggregated data (also derived data)	Storing the student grade point average (STU_GPA) aggregate value in the STUDENT table when this can be calculated from the ENROLL and COURSE tables (see Figure 3.28)	Avoid extra join operations Program computes the GPA every time a grade is entered or updated STU_GPA can be updated only via administrative routine
Information requirements	Using a temporary denormalized table to hold report data; this is required when creating a tabular report in which the columns represent data that are stored in the table as rows (see Figures 6.17 and 6.18)	Impossible to generate the data required by the report using plain SQL No need to maintain table Temporary table is deleted once report is done Processing speed is not an issue

FIGURE 6.18 THE FACULTY EVALUATION REPORT

Faculty Evaluation Report

Instructor	Department	I		II		III		IV		Last Two Sem. Avg.
		Semester	Mean	Semester	Mean	Semester	Mean	Semester	Mean	
Alton	INFS	2015S	2.91	2014F	2.84	2014S	2.55	2013F	2.51	2.875
Ames	INFS	2015S	3.24	2014F	3.26	2014S	3.31	2013F	3.19	3.250
Crandon	INFS	2015S	3.93	2014F	3.95	2014S	3.91	2013F	3.88	3.940
Dumas	MGMT	2014F	3.66	2014S	3.69	2013F	3.56	2013S	3.72	3.675
Landon	BMOM	2015S	3.57	2014F	3.64	2014S	3.39	2013F	3.57	3.605
Lohar	ECON	2009F	3.53	2008F	3.53					3.530
Rolman	INFS	2006S	3.50							3.500

the master data table containing the evaluation scores for each faculty member for each semester taught; this table is normalized. The FACHIST table contains the last four data points—that is, evaluation score and semester—for each faculty member. The FACHIST table is a temporary denormalized table created from the EVALDATA table via a series of queries. (The FACHIST table is the basis for the report shown in Figure 6.18.)

As shown in the faculty evaluation report, the conflicts between design efficiency, information requirements, and performance are often resolved through compromises that may include denormalization. In this case, and assuming there is enough storage space, the designer's choices could be narrowed down to:

- Store the data in a permanent denormalized table. This is not the recommended solution because the denormalized table is subject to data anomalies (insert, update, and delete). This solution is viable only if performance is an issue.
- Create a temporary denormalized table from the permanent normalized table(s). The denormalized table exists only as long as it takes to generate the report; it disappears after the report is produced. Therefore, there are no data anomaly problems. This solution is practical only if performance is not an issue and there are no other viable processing options.

As shown, *normalization purity is often difficult to sustain in the modern database environment*. You will learn in Chapter 13, Business Intelligence and Data Warehouses, that lower

FIGURE 6.19 THE EVALDATA AND FACHIST TABLES

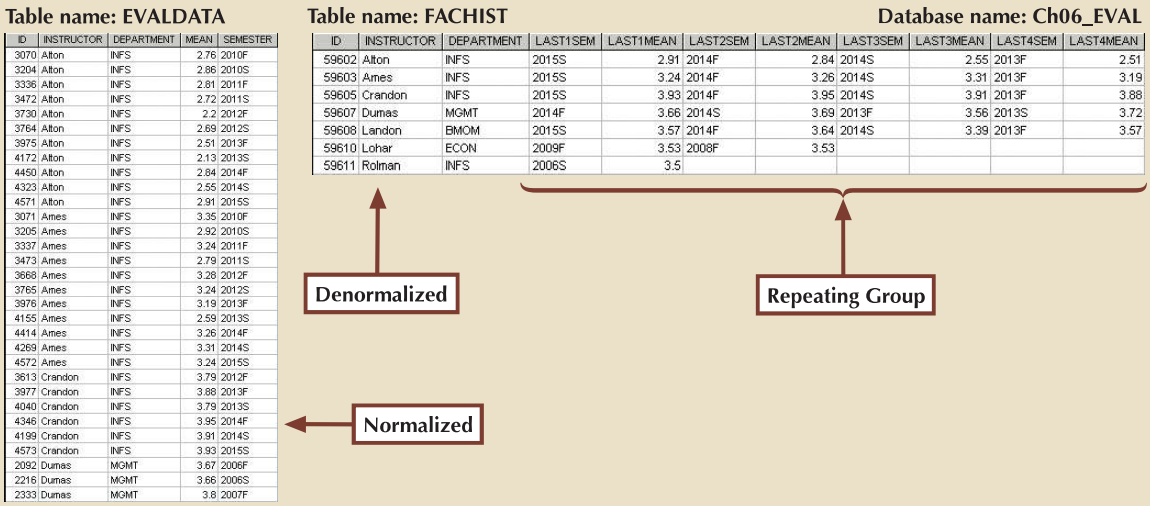


Table name: EVALDATA

ID	INSTRUCTOR	DEPARTMENT	MEAN	SEMESTER
3070	Alton	INFS	2.76	2010F
3204	Alton	INFS	2.86	2010S
3336	Alton	INFS	2.81	2011F
3472	Alton	INFS	2.72	2011S
3730	Alton	INFS	2.2	2012F
3764	Alton	INFS	2.69	2012S
3975	Alton	INFS	2.51	2013F
4172	Alton	INFS	2.13	2013S
4450	Alton	INFS	2.84	2014F
4323	Alton	INFS	2.55	2014S
4571	Alton	INFS	2.91	2015S
3071	Ames	INFS	3.35	2010F
3205	Ames	INFS	2.92	2010S
3337	Ames	INFS	3.24	2011F
3473	Ames	INFS	2.79	2011S
3668	Ames	INFS	3.28	2012F
3765	Ames	INFS	3.24	2012S
3976	Ames	INFS	3.19	2013F
4155	Ames	INFS	2.59	2013S
4414	Ames	INFS	3.26	2014F
4269	Ames	INFS	3.31	2014S
4572	Ames	INFS	3.24	2015S
3613	Crandon	INFS	3.79	2012F
3977	Crandon	INFS	3.88	2013F
4040	Crandon	INFS	3.79	2013S
4346	Crandon	INFS	3.95	2014F
4199	Crandon	INFS	3.91	2014S
4573	Crandon	INFS	3.93	2015S
2092	Dumas	MGMT	3.67	2006F
2216	Dumas	MGMT	3.66	2006S
2333	Dumas	MGMT	3.8	2007F

Table name: FACHIST

Database name: Ch06_EVAL

ID	INSTRUCTOR	DEPARTMENT	LAST1SEM	LAST1MEAN	LAST2SEM	LAST2MEAN	LAST3SEM	LAST3MEAN	LAST4SEM	LAST4MEAN
59602	Alton	INFS	2015S	2.91	2014F	2.84	2014S	2.55	2013F	2.51
59603	Ames	INFS	2015S	3.24	2014F	3.26	2014S	3.31	2013F	3.19
59605	Crandon	INFS	2015S	3.93	2014F	3.95	2014S	3.91	2013F	3.88
59607	Dumas	MGMT	2014F	3.66	2014S	3.69	2013F	3.56	2013S	3.72
59608	Landon	BMOM	2015S	3.57	2014F	3.64	2014S	3.39	2013F	3.57
59610	Lohar	ECON	2009F	3.53	2008F	3.53				
59611	Rolman	INFS	2006S	3.5						

normalization forms occur (and are even required) in specialized databases known as data warehouses. Such specialized databases reflect the ever-growing demand for greater scope and depth in the data on which decision support systems increasingly rely. You will discover that the data warehouse routinely uses 2NF structures in its complex, multilevel, multisource data environment. In short, although normalization is very important, especially in the so-called production database environment, 2NF is no longer disregarded as it once was.

Although 2NF tables cannot always be avoided, the problem of working with tables that contain partial and/or transitive dependencies in a production database environment should not be minimized. Aside from the possibility of troublesome data anomalies being created, unnormalized tables in a production database tend to suffer from these defects:

- Data updates are less efficient because programs that read and update tables must deal with larger tables.
- Indexing is more cumbersome. It is simply not practical to build all of the indexes required for the many attributes that might be located in a single unnormalized table.
- Unnormalized tables yield no simple strategies for creating virtual tables known as *views*. You will learn how to create and use views in Chapter 8, Advanced SQL.

Remember that good design cannot be created in the application programs that use a database. Also keep in mind that unnormalized database tables often lead to various data redundancy disasters in production databases, such as the problems examined thus far. In other words, use denormalization cautiously and make sure that you can explain why the unnormalized tables are a better choice in certain situations than their normalized counterparts.

6-9 Data-Modeling Checklist

In the chapters of Part 2, you have learned how data modeling translates a specific real-world environment into a data model that represents the real-world data, users, processes, and interactions. The modeling techniques you have learned thus far give you the tools needed to produce successful database designs. However, just as any good pilot uses a checklist to ensure that all is in order for a successful flight, the data-modeling checklist shown in Table 6.7 will help ensure that you perform data-modeling tasks successfully based on the concepts and tools you have learned in this text.

Note

You can also find this data-modeling checklist on the inside front cover of this book for easy reference.

TABLE 6.7

DATA-MODELING CHECKLIST

BUSINESS RULES

- Properly document and verify all business rules with the end users.
- Ensure that all business rules are written precisely, clearly, and simply. The business rules must help identify entities, attributes, relationships, and constraints.
- Identify the source of all business rules, and ensure that each business rule is justified, dated, and signed off by an approving authority.

DATA MODELING

Naming conventions: All names should be limited in length (database-dependent size).

- Entity names:
  - Should be nouns that are familiar to business and should be short and meaningful
  - Should document abbreviations, synonyms, and aliases for each entity
  - Should be unique within the model
  - For composite entities, may include a combination of abbreviated names of the entities linked through the composite entity
- Attribute names:
  - Should be unique within the entity
  - Should use the entity abbreviation as a prefix
  - Should be descriptive of the characteristic
  - Should use suffixes such as _ID, _NUM, or _CODE for the PK attribute
  - Should not be a reserved word
  - Should not contain spaces or special characters such as @, !, or &
- Relationship names:
  - Should be active or passive verbs that clearly indicate the nature of the relationship

Entities:

- Each entity should represent a single subject.
- Each entity should represent a set of distinguishable entity instances.
- All entities should be in 3NF or higher. Any entities below 3NF should be justified.
- The granularity of the entity instance should be clearly defined.
- The PK should be clearly defined and support the selected data granularity.

Attributes:

- Should be simple and single-valued (atomic data)
- Should document default values, constraints, synonyms, and aliases
- Derived attributes should be clearly identified and include source(s)
- Should not be redundant unless this is required for transaction accuracy, performance, or maintaining a history
- Nonkey attributes must be fully dependent on the PK attribute

Relationships:

- Should clearly identify relationship participants
- Should clearly define participation, connectivity, and document cardinality

ER model:

- Should be validated against expected processes: inserts, updates, and deletions
- Should evaluate where, when, and how to maintain a history
- Should not contain redundant relationships except as required (see attributes)
- Should minimize data redundancy to ensure single-place updates
- Should conform to the minimal data rule: All that is needed is there, and all that is there is needed.

Summary

- Normalization is a technique used to design tables in which data redundancies are minimized. The first three normal forms (1NF, 2NF, and 3NF) are the most common. From a structural point of view, higher normal forms are better than lower normal forms because higher normal forms yield relatively fewer data redundancies in the database. Almost all business designs use 3NF as the ideal normal form. A special, more restricted 3NF known as Boyce-Codd normal form, or BCNF, is also used.
- A table is in 1NF when all key attributes are defined and all remaining attributes are dependent on the primary key. However, a table in 1NF can still contain both partial and transitive dependencies. A partial dependency is one in which an attribute is functionally dependent on only a part of a multiattribute primary key. A transitive dependency is one in which an attribute is functionally dependent on another nonkey attribute. A table with a single-attribute primary key cannot exhibit partial dependencies.
- A table is in 2NF when it is in 1NF and contains no partial dependencies. Therefore, a 1NF table is automatically in 2NF when its primary key is based on only a single attribute. A table in 2NF may still contain transitive dependencies.
- A table is in 3NF when it is in 2NF and contains no transitive dependencies. Given that definition, the Boyce-Codd normal form (BCNF) is merely a special 3NF case in which all determinant keys are candidate keys. When a table has only a single candidate key, a 3NF table is automatically in BCNF.
- A table that is not in 3NF may be split into new tables until all of the tables meet the 3NF requirements.
- Normalization is an important part—but only a part—of the design process. As entities and attributes are defined during the ER modeling process, subject each entity (set) to normalization checks and form new entities (sets) as required. Incorporate the normalized entities into the ERD and continue the iterative ER process until all entities and their attributes are defined and all equivalent tables are in 3NF.
- A table in 3NF might contain multivalued dependencies that produce either numerous null values or redundant data. Therefore, it might be necessary to convert a 3NF table to the fourth normal form (4NF) by splitting the table to remove the multivalued dependencies. Thus, a table is in 4NF when it is in 3NF and contains no multivalued dependencies.
- The larger the number of tables, the more additional I/O operations and processing logic you need to join them. Therefore, tables are sometimes denormalized to yield less I/O in order to increase processing speed. Unfortunately, with larger tables, you pay for the increased processing speed by making the data updates less efficient, by making indexing more cumbersome, and by introducing data redundancies that are likely to yield data anomalies. In the design of production databases, use denormalization sparingly and cautiously.
- The data-modeling checklist provides a way for the designer to check that the ERD meets a set of minimum requirements.

Key Term

atomic attribute
atomicity
Boyce-Codd normal form (BCNF)
denormalization
dependency diagram
determinant
first normal form (1NF)
fourth normal form (4NF)
granularity
key attribute
nonkey attribute
nonprime attribute
normalization
partial dependency
prime attribute
repeating group
second normal form (2NF)
third normal form (3NF)
transitive dependency

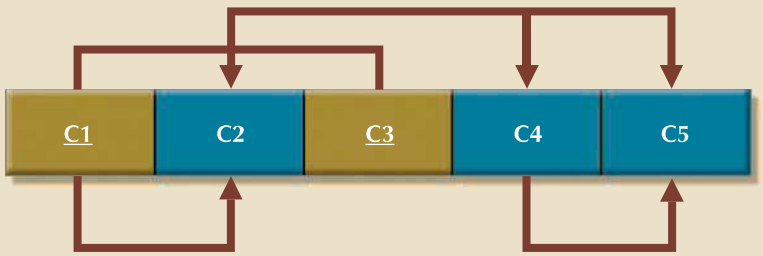
Online Content

Flashcards and crossword puzzles for key term practice are available at *www.cengagebrain.com*.

Review Questions

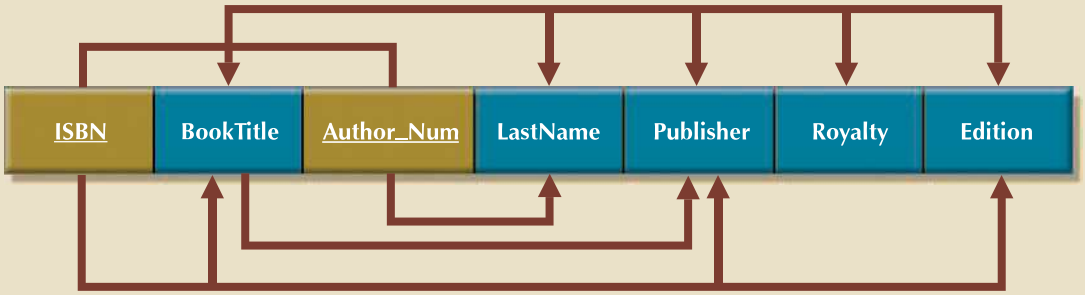
1. What is normalization?
2. When is a table in 1NF?
3. When is a table in 2NF?
4. When is a table in 3NF?
5. When is a table in BCNF?
6. Given the dependency diagram shown in Figure Q6.6, answer Items 6a–6c.

FIGURE Q6.6 DEPENDENCY DIAGRAM FOR QUESTION 6



   a. Identify and discuss each of the indicated dependencies.
   b. Create a database whose tables are at least in 2NF, showing the dependency diagrams for each table.
   c. Create a database whose tables are at least in 3NF, showing the dependency diagrams for each table.
7. The dependency diagram in Figure Q6.7 indicates that authors are paid royalties for each book they write for a publisher. The amount of the royalty can vary by author, by book, and by edition of the book.

FIGURE Q6.7 BOOK ROYALTY DEPENDENCY DIAGRAM

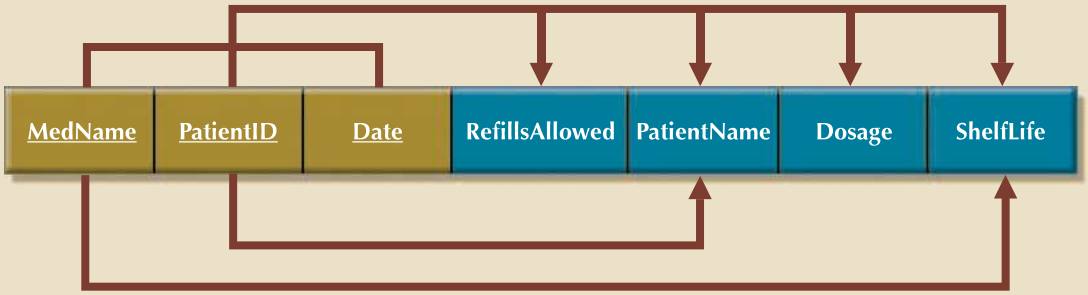


a. Based on the dependency diagram, create a database whose tables are at least in 2NF, showing the dependency diagram for each table.

b. Create a database whose tables are at least in 3NF, showing the dependency diagram for each table.

8. The dependency diagram in Figure Q6.8 indicates that a patient can receive many prescriptions for one or more medicines over time. Based on the dependency diagram, create a database whose tables are in at least 2NF, showing the dependency diagram for each table.

FIGURE Q6.8 PRESCRIPTION DEPENDENCY DIAGRAM

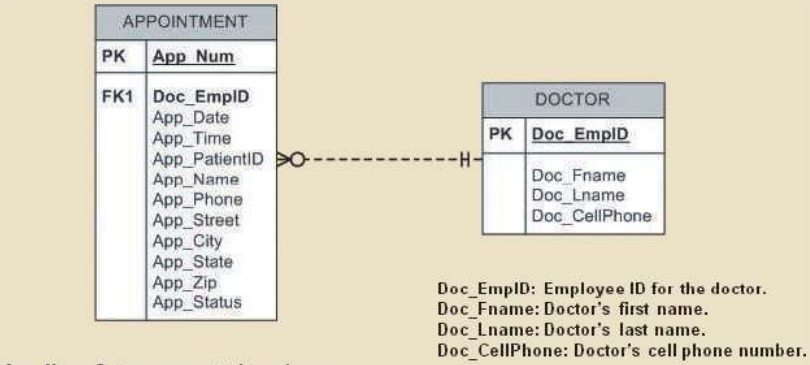


9. What is a partial dependency? With what normal form is it associated?
10. What three data anomalies are likely to be the result of data redundancy? How can such anomalies be eliminated?
11. Define and discuss the concept of transitive dependency.
12. What is a surrogate key, and when should you use one?
13. Why is a table whose primary key consists of a single attribute automatically in 2NF when it is in 1NF?
14. How would you describe a condition in which one attribute is dependent on another attribute when neither attribute is part of the primary key?
15. Suppose someone tells you that an attribute that is part of a composite primary key is also a candidate key. How would you respond to that statement?
16. A table is in _____ normal form when it is in _____ and there are no transitive dependencies.

Problems

1. Using the descriptions of the attributes given in the figure, convert the ERD shown in Figure P6.1 into a dependency diagram that is in at least 3NF.

FIGURE P6.1 APPOINTMENT ERD FOR PROBLEM 1



Doc_EmpID: Employee ID for the doctor.
Doc_Fname: Doctor's first name.
Doc_Lname: Doctor's last name.
Doc_CellPhone: Doctor's cell phone number.

App_Num: System-generated number.
App_Date: The date of the appointment.
App_Time: The time of the appointment.
App_PatientID: The ID number of the patient.
App_Name: The name of the patient.
App_Phone: The contact phone number of the patient.
App_Street: The street address for the patient.
App_City: The city the patient lives in.
App_State: The state the patient lives in.
App_Zip: The zip code for the patient's address.
App_Status: The status of the appointment (pending, closed, cancelled)

2. Using the descriptions of the attributes given in the figure, convert the ERD shown in Figure P6.2 into a dependency diagram that is in at least 3NF.

FIGURE P6.2 PRESENTATION ERD FOR PROBLEM 2

Reg_Date: The date the attendee registered.

Pres_SessionNum: System-generated number.
Pres_Order: Number indicating the order of the presentations during the session.
Pres_Date: The date that the presentation is scheduled to be given.
Pres_Room: The room in which the presentation will be given.
Pres_Title: The title of the presentation.
Pres_AuthorID: System-generated number assigned to presentation authors.
Pres_FName: The first name of the presentation author.
Pres_LName: The last name of the presentation author.

Att_Num: System-generated number.
Att_FName: The first name of the attendee.
Att_LName: The last name of the attendee.
Att_Status: Whether or not the attendee has paid the registration fee.

3. Using the INVOICE table structure shown in Table P6.3, do the following:

TABLE P6.3

ATTRIBUTE NAME	SAMPLE VALUE	SAMPLE VALUE	SAMPLE VALUE	SAMPLE VALUE	SAMPLE VALUE
INV_NUM	211347	211347	211347	211348	211349
PROD_NUM	AA-E3422QW	QD-300932X	RU-995748G	AA-E3422QW	GH-778345P
SALE_DATE	15-Jan-2016	15-Jan-2016	15-Jan-2016	15-Jan-2016	16-Jan-2016
PROD_LABEL	Rotary sander	0.25-in. drill bit	Band saw	Rotary sander	Power drill
VEND_CODE	211	211	309	211	157
VEND_NAME	NeverFail, Inc.	NeverFail, Inc.	BeGood, Inc.	NeverFail, Inc.	ToughGo, Inc.
QUANT_SOLD	1	8	1	2	1
PROD_PRICE	$49.95	$3.45	$39.99	$49.95	$87.75

   a. Write the relational schema, draw its dependency diagram, and identify all dependencies, including all partial and transitive dependencies. You can assume that the table does not contain repeating groups and that an invoice number references more than one product. (*Hint:* This table uses a composite primary key.)

   b. Remove all partial dependencies, write the relational schema, and draw the new dependency diagrams. Identify the normal forms for each table structure you created.

Note

You can assume that any given product is supplied by a single vendor, but a vendor can supply many products. Therefore, it is proper to conclude that the following dependency exists:

PROD_NUM → PROD_LABEL, PROD_PRICE, VEND_CODE, VEND_NAME

(*Hint*: Your actions should produce three dependency diagrams.)

   c. Remove all transitive dependencies, write the relational schema, and draw the new dependency diagrams. Also identify the normal forms for each table structure you created.

   d. Draw the Crow's Foot ERD.

4. Using the STUDENT table structure shown in Table P6.4, do the following:

   a. Write the relational schema and draw its dependency diagram. Identify all dependencies, including all transitive dependencies.

   b. Write the relational schema and draw the dependency diagram to meet the 3NF requirements to the greatest practical extent possible. If you believe that practical considerations dictate using a 2NF structure, explain why your decision to retain 2NF is appropriate. If necessary, add or modify attributes to create appropriate determinants and to adhere to the naming conventions.

   c. Using the results of Problem 4, draw the Crow's Foot ERD.

TABLE P6.4

ATTRIBUTE NAME	SAMPLE VALUE	SAMPLE VALUE	SAMPLE VALUE	SAMPLE VALUE	SAMPLE VALUE
STU_NUM	211343	200128	199876	198648	223456
STU_LNAME	Stephanos	Smith	Jones	Ortiz	McKulski
STU_MAJOR	Accounting	Accounting	Marketing	Marketing	Statistics
DEPT_CODE	ACCT	ACCT	MKTG	MKTG	MATH
DEPT_NAME	Accounting	Accounting	Marketing	Marketing	Mathematics
DEPT_PHONE	4356	4356	4378	4378	3420
COLLEGE_NAME	Business Admin	Business Admin	Business Admin	Business Admin	Arts & Sciences
ADVISOR_LNAME	Grastrand	Grastrand	Gentry	Tillery	Chen
ADVISOR_OFFICE	T201	T201	T228	T356	J331
ADVISOR_BLDG	Torre Building	Torre Building	Torre Building	Torre Building	Jones Building
ADVISOR_PHONE	2115	2115	2123	2159	3209
STU_GPA	3.87	2.78	2.31	3.45	3.58
STU_HOURS	75	45	117	113	87
STU_CLASS	Junior	Sophomore	Senior	Senior	Junior

Note

Although the completed student hours (STU_HOURS) do determine the student classification (STU_CLASS), this dependency is not as obvious as you might initially assume it to be. For example, a student is considered a junior if the student has completed between 61 and 90 credit hours.

5. To keep track of office furniture, computers, printers, and other office equipment, the FOUNDIT Company uses the table structure shown in Table P6.5.

TABLE P6.5

ATTRIBUTE NAME	SAMPLE VALUE	SAMPLE VALUE	SAMPLE VALUE
ITEM_ID	231134-678	342245-225	254668-449
ITEM_LABEL	HP DeskJet 895Cse	HP Toner	DT Scanner
ROOM_NUMBER	325	325	123
BLDG_CODE	NTC	NTC	CSF
BLDG_NAME	Nottooclear	Nottooclear	Canseefar
BLDG_MANAGER	I. B. Rightonit	I. B. Rightonit	May B. Next

   a. Given that information, write the relational schema and draw the dependency diagram. Make sure that you label the transitive and/or partial dependencies.

   b. Write the relational schema and create a set of dependency diagrams that meet 3NF requirements. Rename attributes to meet the naming conventions, and create new entities and attributes as necessary.

   c. Draw the Crow's Foot ERD.

6. The table structure shown in Table P6.6 contains many unsatisfactory components and characteristics. For example, there are several multivalued attributes, naming conventions are violated, and some attributes are not atomic.

TABLE P6.6

EMP_NUM	1003	1018	1019	1023
EMP_LNAME	Willaker	Smith	McGuire	McGuire
EMP_EDUCATION	BBA, MBA	BBA		BS, MS, Ph.D.
JOB_CLASS	SLS	SLS	JNT	DBA
EMP_DEPENDENTS	Gerald (spouse), Mary (daughter), John (son)		JoAnne (spouse)	George (spouse) Jill (daughter)
DEPT_CODE	MKTG	MKTG	SVC	INFS
DEPT_NAME	Marketing	Marketing	General Service	Info. Systems
DEPT_MANAGER	Jill H. Martin	Jill H. Martin	Hank B. Jones	Carlos G. Ortez
EMP_TITLE	Sales Agent	Sales Agent	Janitor	DB Admin
EMP_DOB	23-Dec-1968	28-Mar-1979	18-May-1982	20-Jul-1959
EMP_HIRE_DATE	14-Oct-1997	15-Jan-2006	21-Apr-2003	15-Jul-1999
EMP_TRAINING	L1, L2	L1	L1	L1, L3, L8, L15
EMP_BASE_SALARY	$38,255.00	$30,500.00	$19,750.00	$127,900.00
EMP_COMMISSION_RATE	0.015	0.010		

   a. Given the structure shown in Table P6.6, write the relational schema and draw its dependency diagram. Label all transitive and/or partial dependencies.

   b. Draw the dependency diagrams that are in 3NF. (*Hint:* You might have to create a few new attributes. Also make sure that the new dependency diagrams contain attributes that meet proper design criteria; i.e., make sure there are no multivalued attributes, that the naming conventions are met, and so on.)

   c. Draw the relational diagram.

   d. Draw the Crow's Foot ERD.

7. Suppose you are given the following business rules to form the basis for a database design. The database must enable the manager of a company dinner club to mail invitations to the club's members, to plan the meals, to keep track of who attends the dinners, and so on.

   - Each dinner serves many members, and each member may attend many dinners.
   - A member receives many invitations, and each invitation is mailed to many members.
   - A dinner is based on a single entree, but an entree may be used as the basis for many dinners. For example, a dinner may be composed of a fish entree, rice, and corn, or the dinner may be composed of a fish entree, a baked potato, and string beans.

   Because the manager is not a database expert, the first attempt at creating the database uses the structure shown in Table P6.7.

   a. Given the table structure illustrated in Table P6.7, write the relational schema and draw its dependency diagram. Label all transitive and/or partial dependencies. (*Hint:* This structure uses a composite primary key.)

TABLE P6.7

ATTRIBUTE NAME	SAMPLE VALUE	SAMPLE VALUE	SAMPLE VALUE
MEMBER_NUM	214	235	214
MEMBER_NAME	Alice B. VanderVoort	Gerald M. Gallega	Alice B. VanderVoort
MEMBER_ADDRESS	325 Meadow Park	123 Rose Court	325 Meadow Park
MEMBER_CITY	Murkywater	Highlight	Murkywater
MEMBER_ZIPCODE	12345	12349	12345
INVITE_NUM	8	9	10
INVITE_DATE	23-Feb-2016	12-Mar-2016	23-Feb-2016
ACCEPT_DATE	27-Feb-2016	15-Mar-2016	27-Feb-2016
DINNER_DATE	15-Mar-2016	17-Mar-2016	15-Mar-2016
DINNER_ATTENDED	Yes	Yes	No
DINNER_CODE	DI5	DI5	DI2
DINNER_DESCRIPTION	Glowing Sea Delight	Glowing Sea Delight	Ranch Superb
ENTREE_CODE	EN3	EN3	EN5
ENTREE_DESCRIPTION	Stuffed crab	Stuffed crab	Marinated steak
DESSERT_CODE	DE8	DE5	DE2
DESSERT_DESCRIPTION	Chocolate mousse with raspberry sauce	Cherries jubilee	Apple pie with honey crust

b. Break up the dependency diagram you drew in Problem 7a to produce dependency diagrams that are in 3NF, and write the relational schema. (*Hint:* You might have to create a few new attributes. Also, make sure that the new dependency diagrams contain attributes that meet proper design criteria; i.e., make sure there are no multivalued attributes, that the naming conventions are met, and so on.)

c. Using the results of Problem 7b, draw the Crow's Foot ERD.

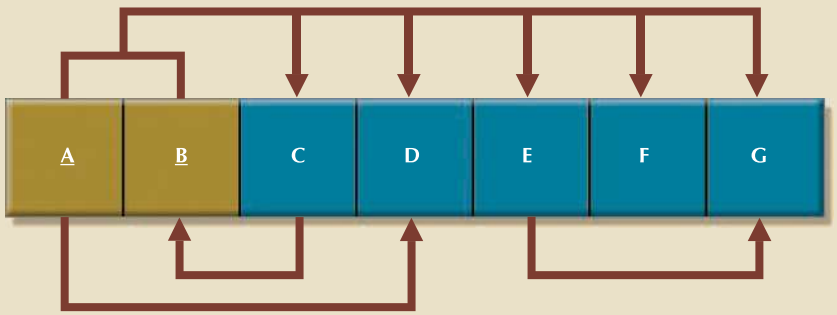
8. Use the dependency diagram shown in Figure P6.8 to work the following problems.

a. Break up the dependency diagram shown in Figure P6.8 to create two new dependency diagrams: one in 3NF and one in 2NF.

b. Modify the dependency diagrams you created in Problem 8a to produce a set of dependency diagrams that are in 3NF. (*Hint:* One of your dependency diagrams should be in 3NF but not in BCNF.)

c. Modify the dependency diagrams you created in Problem 8b to produce a collection of dependency diagrams that are in 3NF and BCNF.

FIGURE P6.8 INITIAL DEPENDENCY DIAGRAM FOR PROBLEM 8



9. Suppose you have been given the table structure and data shown in Table P6.9, which was imported from an Excel spreadsheet. The data reflects that a professor can have multiple advisees, can serve on multiple committees, and can edit more than one journal.

TABLE P6.9

ATTRIBUTE NAME	SAMPLE VALUE	SAMPLE VALUE	SAMPLE VALUE	SAMPLE VALUE
EMP_NUM	123	104	118	
PROF_RANK	Professor	Asst. Professor	Assoc. Professor	Assoc. Professor
EMP_NAME	Ghee	Rankin	Ortega	Smith
DEPT_CODE	CIS	CHEM	CIS	ENG
DEPT_NAME	Computer Info. Systems	Chemistry	Computer Info. Systems	English
PROF_OFFICE	KDD-567	BLF-119	KDD-562	PRT-345
ADVISEE	1215, 2312, 3233, 2218, 2098	3102, 2782, 3311, 2008, 2876, 2222, 3745, 1783, 2378	2134, 2789, 3456, 2002, 2046, 2018, 2764	2873, 2765, 2238, 2901, 2308
COMMITTEE_CODE	PROMO, TRAF, APPL, DEV	DEV	SPR, TRAF	PROMO, SPR, DEV
JOURNAL_CODE	JMIS, QED, JMGT		JCIS, JMGT	

Given the information in Table P6.9:

a. Draw the dependency diagram.

b. Identify the multivalued dependencies.

c. Create the dependency diagrams to yield a set of table structures in 3NF.

d. Eliminate the multivalued dependencies by converting the affected table structures to 4NF.

e. Draw the Crow's Foot ERD to reflect the dependency diagrams you drew in Problem 9c. (*Note:* You might have to create additional attributes to define the proper PKs and FKs. Make sure that all of your attributes conform to the naming conventions.)

10. The manager of a consulting firm has asked you to evaluate a database that contains the table structure shown in Table P6.10.

Table P6.10 was created to enable the manager to match clients with consultants. The objective is to match a client within a given region with a consultant in that region and to make sure that the client's need for specific consulting services is properly matched to the consultant's expertise. For example, if the client needs help with database design and is located in the Southeast, the objective is to make a match with a consultant who is located in the Southeast and whose expertise is in database design. (Although the consulting company manager tries to match consultant and client locations to minimize travel expense, it is not always possible to do so.) The following basic business rules are maintained:

- Each client is located in one region.
- A region can contain many clients.

- Each consultant can work on many contracts.
- Each contract might require the services of many consultants.
- A client can sign more than one contract, but each contract is signed by only one client.
- Each contract might cover multiple consulting classifications. (For example, a contract may list consulting services in database design and networking.)

TABLE P6.10

ATTRIBUTE NAME	SAMPLE VALUE	SAMPLE VALUE	SAMPLE VALUE
CLIENT_NUM	298	289	289
CLIENT_NAME	Marianne R. Brown	James D. Smith	James D. Smith
CLIENT_REGION	Midwest	Southeast	Southeast
CONTRACT_DATE	10-Feb-2016	15-Feb-2016	12-Mar-2016
CONTRACT_NUMBER	5841	5842	5843
CONTRACT_AMOUNT	$2,985,000.00	$670,300.00	$1,250,000.00
CONSULT_CLASS_1	Database Administration	Internet Services	Database Design
CONSULT_CLASS_2	Web Applications		Database Administration
CONSULT_CLASS_3			Network Installation
CONSULT_CLASS_4			
CONSULTANT_NUM_1	29	34	25
CONSULTANT_NAME_1	Rachel G. Carson	Gerald K. Ricardo	Angela M. Jamison
CONSULTANT_REGION_1	Midwest	Southeast	Southeast
CONSULTANT_NUM_2	56	38	34
CONSULTANT_NAME_2	Karl M. Spenser	Anne T. Dimarco	Gerald K. Ricardo
CONSULTANT_REGION_2	Midwest	Southeast	Southeast
CONSULTANT_NUM_3	22	45	
CONSULTANT_NAME_3	Julian H. Donatello	Geraldo J. Rivera	
CONSULTANT_REGION_3	Midwest	Southeast	
CONSULTANT_NUM_4		18	
CONSULTANT_NAME_4		Donald Chen	
CONSULTANT_REGION_4		West	

- Each consultant is located in one region.
- A region can contain many consultants.
- Each consultant has one or more areas of expertise (class). For example, a consultant might be classified as an expert in both database design and networking.
- Each area of expertise (class) can have many consultants. For example, the consulting company might employ many consultants who are networking experts.

a. Given this brief description of the requirements and the business rules, write the relational schema and draw the dependency diagram for the preceding (and very poor) table structure. Label all transitive and/or partial dependencies.

b. Break up the dependency diagram you drew in Problem 10a to produce dependency diagrams that are in 3NF and write the relational schema. (*Hint:* You might

have to create a few new attributes. Also make sure that the new dependency diagrams contain attributes that meet proper design criteria; that is, make sure there are no multivalued attributes, that the naming conventions are met, and so on.)

c. Using the results of Problem 10b, draw the Crow's Foot ERD.

11. Given the sample records in the CHARTER table shown in Table P6.11, do the following:

a. Write the relational schema and draw the dependency diagram for the table structure. Make sure that you label all dependencies. CHAR_PAX indicates the number of passengers carried. The CHAR_MILES entry is based on round-trip miles, including pickup points. (*Hint:* Look at the data values to determine the nature of the relationships. For example, note that employee Melton has flown two charter trips as pilot and one trip as copilot.)

b. Decompose the dependency diagram you drew to solve Problem 11a to create table structures that are in 3NF and write the relational schema.

c. Draw the Crow's Foot ERD to reflect the properly decomposed dependency diagrams you created in Problem 11b. Make sure the ERD yields a database that can track all of the data shown in Problem 11. Show all entities, relationships, connectivities, optionalities, and cardinalities.

TABLE P6.11

ATTRIBUTE NAME	SAMPLE VALUE	SAMPLE VALUE	SAMPLE VALUE	SAMPLE VALUE
CHAR_TRIP	10232	10233	10234	10235
CHAR_DATE	15-Jan-2016	15-Jan-2016	16-Jan-2016	17-Jan-2016
CHAR_CITY	STL	MIA	TYS	ATL
CHAR_MILES	580	1,290	524	768
CUST_NUM	784	231	544	784
CUST_LNAME	Brown	Hanson	Bryana	Brown
CHAR_PAX	5	12	2	5
CHAR_CARGO	235 lbs.	18,940 lbs.	348 lbs.	155 lbs.
PILOT	Melton	Chen	Henderson	Melton
COPILOT		Henderson	Melton	
FLT_ENGINEER		O'Shaski		
LOAD_MASTER		Benkasi		
AC_NUMBER	1234Q	3456Y	1234Q	2256W
MODEL_CODE	PA31-350	CV-580	PA31-350	PA31-350
MODEL_SEATS	10	38	10	10
MODEL_CHG_MILE	$2.79	$23.36	$2.79	$2.79

PART 3
Advanced Design and Implementation

Chapter 7

Introduction to Structured Query Language (SQL)

In this chapter, you will learn:

- The basic commands and functions of SQL
- How to use SQL for data administration (to create tables and indexes)
- How to use SQL for data manipulation (to add, modify, delete, and retrieve data)
- How to use SQL to query a database for useful information

Preview

In this chapter, you will learn the basics of Structured Query Language (SQL). SQL, which is pronounced *S-Q-L* or *sequel*, is composed of commands that enable users to create database and table structures, perform various types of data manipulation and data administration, and query the database to extract useful information. All relational DBMS software supports SQL, and many software vendors have developed extensions to the basic SQL command set.

Although it is quite useful and powerful, SQL is not meant to stand alone in the applications arena. Data entry with SQL is possible but awkward, as are data corrections and additions. SQL itself does not create menus, special report forms, overlays, pop-ups, or other features that end users usually expect. Instead, those features are available as vendor-supplied enhancements. SQL focuses on data definition (creating tables and indexes) and data manipulation (adding, modifying, deleting, and retrieving data). This chapter covers these basic functions. In spite of its limitations, SQL is a powerful tool for extracting information and managing data.

Data Files and Available Formats

	MS Access	Oracle	MS SQL	My SQL		MS Access	Oracle	MS SQL	My SQL
CH07_SaleCo	✓	✓	✓	✓	CH07_ConstructCo	✓	✓	✓	✓
					CH07_LargeCo	✓	✓	✓	✓
					Ch07_Fact	✓	✓	✓	✓

Data Files Available on cengagebrain.com

Note

Although you can use the MS Access databases and SQL script files for creating the tables and loading the data supplied online, it is strongly suggested that you create your own database structures so you can practice the SQL commands illustrated in this chapter.

How you connect to your database depends on how the software was installed on your computer. Follow the instructions provided by your instructor or school.

7-1 Introduction to SQL

Ideally, a database language allows you to create database and table structures, perform basic data management chores (add, delete, and modify), and perform complex queries designed to transform the raw data into useful information. Moreover, a database language must perform such basic functions with minimal user effort, and its command structure and syntax must be easy to learn. Finally, it must be portable; that is, it must conform to some basic standard so a person does not have to relearn the basics when moving from one RDBMS to another. SQL meets those ideal database language requirements well.

SQL functions fit into two broad categories:

- It is a *data definition language (DDL)*. SQL includes commands to create database objects such as tables, indexes, and views, as well as commands to define access rights to those database objects. Some common data definition commands you will learn are listed in Table 7.1.
- It is a *data manipulation language (DML)*. SQL includes commands to insert, update, delete, and retrieve data within the database tables. The data manipulation commands you will learn in this chapter are listed in Table 7.2.

TABLE 7.1

SQL DATA DEFINITION COMMANDS

COMMAND OR OPTION	DESCRIPTION
CREATE SCHEMA AUTHORIZATION	Creates a database schema
CREATE TABLE	Creates a new table in the user's database schema
NOT NULL	Ensures that a column will not have null values
UNIQUE	Ensures that a column will not have duplicate values
PRIMARY KEY	Defines a primary key for a table
FOREIGN KEY	Defines a foreign key for a table
DEFAULT	Defines a default value for a column (when no value is given)
CHECK	Validates data in an attribute
CREATE INDEX	Creates an index for a table
CREATE VIEW	Creates a dynamic subset of rows and columns from one or more tables (see Chapter 8, Advanced SQL)
ALTER TABLE	Modifies a table's definition (adds, modifies, or deletes attributes or constraints)
CREATE TABLE AS	Creates a new table based on a query in the user's database schema
DROP TABLE	Permanently deletes a table (and its data)
DROP INDEX	Permanently deletes an index
DROP VIEW	Permanently deletes a view

SQL is relatively easy to learn. Its basic command set has a vocabulary of fewer than 100 words. Better yet, SQL is a nonprocedural language: you merely command *what* is to be done; you do not have to worry about *how*. For example, a single command creates the complex table structures required to store and manipulate data successfully; end users and programmers do not need to know the physical data storage format or the complex activities that take place when a SQL command is executed.

TABLE 7.2

SQL DATA MANIPULATION COMMANDS

COMMAND OR OPTION	DESCRIPTION
INSERT	Inserts row(s) into a table
SELECT	Selects attributes from rows in one or more tables or views
WHERE	Restricts the selection of rows based on a conditional expression
GROUP BY	Groups the selected rows based on one or more attributes
HAVING	Restricts the selection of grouped rows based on a condition
ORDER BY	Orders the selected rows based on one or more attributes
UPDATE	Modifies an attribute's values in one or more table's rows
DELETE	Deletes one or more rows from a table
COMMIT	Permanently saves data changes
ROLLBACK	Restores data to its original values
Comparison operators	
=, <, >, <=, >=, <>, !=	Used in conditional expressions
Logical operators	
AND/OR/NOT	Used in conditional expressions
Special operators	Used in conditional expressions
BETWEEN	Checks whether an attribute value is within a range
IS NULL	Checks whether an attribute value is null
LIKE	Checks whether an attribute value matches a given string pattern
IN	Checks whether an attribute value matches any value within a value list
EXISTS	Checks whether a subquery returns any rows
DISTINCT	Limits values to unique values
Aggregate functions	Used with SELECT to return mathematical summaries on columns
COUNT	Returns the number of rows with non-null values for a given column
MIN	Returns the minimum attribute value found in a given column
MAX	Returns the maximum attribute value found in a given column
SUM	Returns the sum of all values for a given column
AVG	Returns the average of all values for a given column

The American National Standards Institute (ANSI) prescribes a standard SQL. The ANSI SQL standards are also accepted by the International Organization for Standardization (ISO), a consortium composed of national standards bodies of more than 150 countries. Although adherence to the ANSI/ISO SQL standard is usually required in commercial and government contract database specifications, many RDBMS vendors add their own special enhancements. Consequently, it is seldom possible to move a SQL-based application from one RDBMS to another without making some changes.

However, even though there are several different SQL "dialects," their differences are minor. Whether you use Oracle, Microsoft SQL Server, MySQL, IBM's DB2, Microsoft Access, or any other well-established RDBMS, a software manual should be sufficient to get you up to speed if you know the material presented in this chapter.

At the heart of SQL is the query. In Chapter 1, Database Systems, you learned that a query is a spur-of-the-moment question. Actually, in the SQL environment, the word *query* covers both questions and actions. Most SQL queries are used to answer questions

such as these: "What products currently held in inventory are priced over $100, and what is the quantity on hand for each of those products?" or "How many employees have been hired since January 1, 2016, by each of the company's departments?" However, many SQL queries are used to perform actions such as adding or deleting table rows or changing attribute values within tables. Still other SQL queries create new tables or indexes. In short, for a DBMS, a query is simply a SQL statement that must be executed. However, before you can use SQL to query a database, you must define the database environment for SQL with its data definition commands.

Online Content

The database model in Figure 7.1 is implemented in the Microsoft Access Ch07_SaleCo database, which is available at *www.cengagebrain.com*. (This database contains a few additional tables that are not reflected in Figure 7.1. These tables are used for discussion purposes only.)

7-2 Data Definition Commands

Before you examine the SQL syntax for creating and defining tables and other elements, first examine a simple database model and the database tables that form the basis for the many SQL examples you will explore in this chapter.

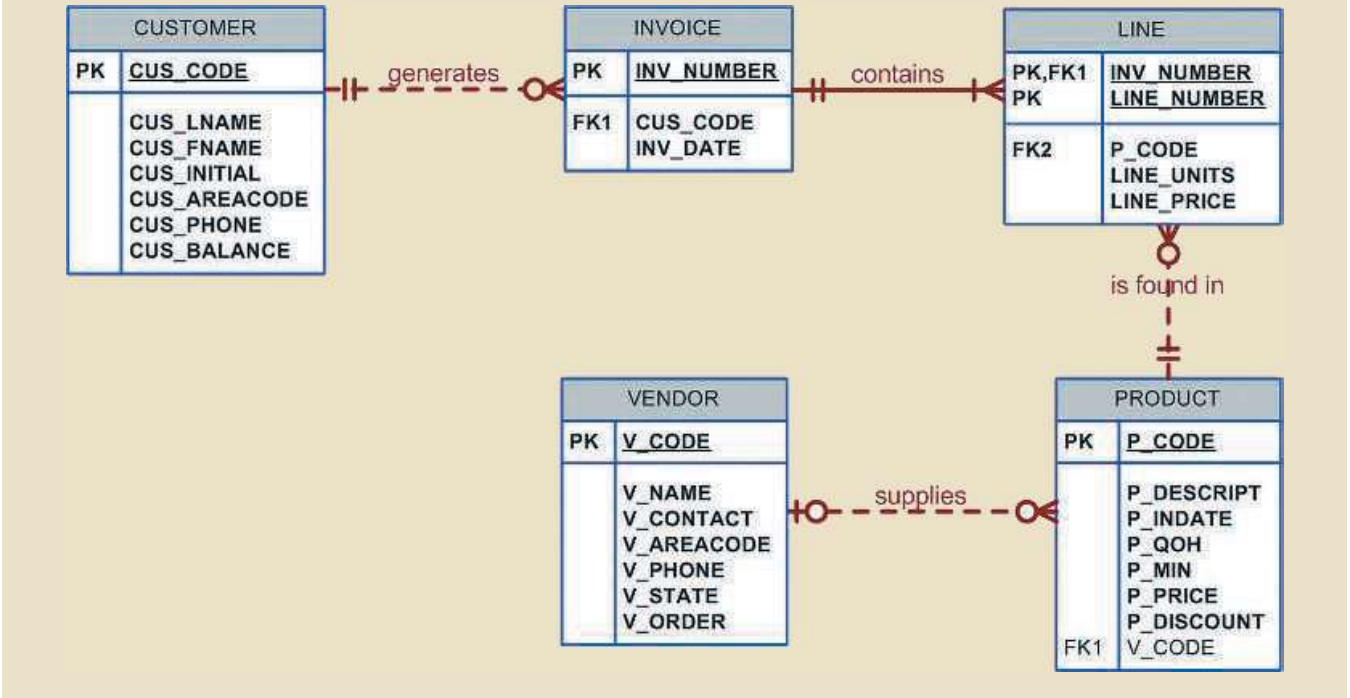
7-2a The Database Model

A simple database composed of the following tables is used to illustrate the SQL commands in this chapter: CUSTOMER, INVOICE, LINE, PRODUCT, and VENDOR. This database model is shown in Figure 7.1.

The database model in Figure 7.1 reflects the following business rules:

- A customer may generate many invoices. Each invoice is generated by one customer.
- An invoice contains one or more invoice lines. Each invoice line is associated with one invoice.
- Each invoice line references one product. A product may be found in many invoice lines. (You can sell more than one hammer to more than one customer.)

FIGURE 7.1 THE DATABASE MODEL



- A vendor *may* supply many products. Some vendors do not yet supply products. For example, a vendor list may include *potential* vendors.
- If a product is vendor-supplied, it is supplied by only a single vendor.
- Some products are not supplied by a vendor. For example, some products may be produced in-house or bought on the open market.

As you can see in Figure 7.1, the database model contains many tables. However, to illustrate the initial set of data definition commands, the focus of attention will be the PRODUCT and VENDOR tables. You will have the opportunity to use the remaining tables later in this chapter and in the Problems section.

To give you a point of reference for understanding the effect of the SQL queries, the contents of the PRODUCT and VENDOR tables are listed in Figure 7.2. In the tables, note the following features, which correspond to the business rules reflected in the ERD shown in Figure 7.1:

- The VENDOR table contains vendors who are not referenced in the PRODUCT table. Database designers note that possibility by saying that PRODUCT is optional to VENDOR; a vendor may exist without a reference to a product. You examined such optional relationships in detail in Chapter 4, Entity Relationship (ER) Modeling.
- Existing V_CODE values in the PRODUCT table must (and do) have a match in the VENDOR table to ensure referential integrity.
- A few products are supplied factory-direct, a few are made in-house, and a few may have been bought in a warehouse sale. In other words, a product is not necessarily supplied by a vendor. Therefore, VENDOR is optional to PRODUCT.

FIGURE 7.2 THE VENDOR AND PRODUCT TABLES

Table name: VENDOR **Database name: Ch07_SaleCo**

V_CODE	V_NAME	V_CONTACT	V_AREACODE	V_PHONE	V_STATE	V_ORDER
21225	Bryson, Inc.	Smithson	615	223-3234	TN	Y
21226	SuperLoo, Inc.	Flushing	904	215-8995	FL	N
21231	D&E Supply	Singh	615	228-3245	TN	Y
21344	Gomez Bros.	Ortega	615	889-2546	KY	N
22567	Dome Supply	Smith	901	678-1419	GA	N
23119	Randsets Ltd.	Anderson	901	678-3998	GA	Y
24004	Brackman Bros.	Browning	615	228-1410	TN	N
24288	ORDVA, Inc.	Hakford	615	898-1234	TN	Y
25443	B&K, Inc.	Smith	904	227-0093	FL	N
25501	Damal Supplies	Smythe	615	890-3529	TN	N
25595	Rubicon Systems	Orton	904	456-0092	FL	Y

Table name: PRODUCT

P_CODE	P_DESCRIPT	P_INDATE	P_QOH	P_MIN	P_PRICE	P_DISCOUNT	V_CODE
11QER/31	Power painter, 15 psi., 3-nozzle	03-Nov-15	8	5	109.99	0.00	25595
13-Q2/P2	7.25-in. pwr. saw blade	13-Dec-15	32	15	14.99	0.05	21344
14-Q1/L3	9.00-in. pwr. saw blade	13-Nov-15	18	12	17.49	0.00	21344
1546-QQ2	Hrd. cloth, 1/4-in., 2x50	15-Jan-16	15	8	39.95	0.00	23119
1558-QW1	Hrd. cloth, 1/2-in., 3x50	15-Jan-16	23	5	43.99	0.00	23119
2232/QTY	B&D jigsaw, 12-in. blade	30-Dec-15	8	5	109.92	0.05	24288
2232/QWE	B&D jigsaw, 8-in. blade	24-Dec-15	6	5	99.87	0.05	24288
2238/QPD	B&D cordless drill, 1/2-in.	20-Jan-16	12	5	38.95	0.05	25595
23109-HB	Claw hammer	20-Jan-16	23	10	9.95	0.10	21225
23114-AA	Sledge hammer, 12 lb.	02-Jan-16	8	5	14.40	0.05	
54778-2T	Rat-tail file, 1/8-in. fine	15-Dec-15	43	20	4.99	0.00	21344
89-WRE-Q	Hicut chain saw, 16 in.	07-Feb-16	11	5	256.99	0.05	24288
PVC23DRT	PVC pipe, 3.5-in., 8-ft	20-Feb-16	188	75	5.87	0.00	
SM-18277	1.25-in. metal screw, 25	01-Mar-16	172	75	6.99	0.00	21225
SW-23116	2.5-in. wd. screw, 50	24-Feb-16	237	100	8.45	0.00	21231
WR3/TT3	Steel matting, 4'x8'x1/6", .5" mesh	17-Jan-16	18	5	119.95	0.10	25595