

The background of the book cover is an abstract composition. It features a grid of four quadrants. The top-left quadrant is a solid blue rectangle. The top-right, bottom-left, and bottom-right quadrants contain a blurred, high-speed photograph of a waterfall. The water is white and frothy at the base, with streaks of blue and green above it, suggesting motion and depth. The overall color palette is dominated by blues and greens, with the white of the water providing contrast.

Systems Analysis & Design

5TH EDITION

DENNIS • WIXOM • ROTH

CHAPTER 5

PROCESS

MODELING

A process model describes business processes—the activities that people do. Process models are developed for the as-is system and/or the to-be system. This chapter describes data flow diagramming, one of the most commonly used process modeling techniques.

OBJECTIVES

- Explain the rules and style guidelines for data flow diagrams.
- Describe the process used to create data flow diagrams.
- Create data flow diagrams.

CHAPTER OUTLINE

Introduction

Data Flow Diagrams

Reading Data Flow Diagrams

Elements of Data Flow Diagrams

Using Data Flow Diagrams to Define

Business Processes

Process Descriptions

Creating Data Flow Diagrams

Creating the Context Diagram

Creating Data Flow Diagram

Fragments

Creating the Level 0 Data Flow

Diagram

Creating Level 1 Data Flow
Diagrams (and Below)

Validating Data Flow Diagrams

Applying the Concepts at Tune Source

Creating the Context Diagram

Creating Data Flow Diagram

Fragments

Creating the Level 0 Data Flow
Diagram

Creating Level 1 Data Flow
Diagrams (and Below)

Validating Data Flow Diagrams

Summary

INTRODUCTION

Chapters 3 and 4 discussed several requirements elicitation activities, such as interviewing and JAD, and how to clarify those requirements by developing more detailed use cases. In this chapter, we discuss how the requirements definition and use cases may be further clarified through a process model. You may have heard the expression “A picture is worth a 1,000 words.” A *process model* is a graphical way of representing how a business system should operate. It illustrates the processes or activities that are performed and how data move among them. A process model can be used to document the current system (i.e., as-is system) or the new system being developed (i.e., to-be system), whether computerized or not.

Process models have been a part of structured systems analysis and design techniques for many years. Today, with use cases gaining favor due to their ability to clarify user requirements in an understandable way, you may see that organizations place less emphasis on process modeling than in the past. An organization that strives to create fully dressed use cases as depicted in Figure 4-1, for example, may not find that process models add much to their understanding of the system under development. Other organizations, however, especially those that create more casual use cases, may find process modeling to be a beneficial part of their analysis phase deliverables. We find that graphically depicting the system that will be developed in a set of well-organized diagrams is a very useful approach. Remember that our goal is to be able to employ an array of tools and techniques that will help us understand and clarify what the new system must do before the new system is actually constructed.

There are many different process modeling techniques in use today. In this chapter, we focus on one of the most commonly used techniques:¹ data flow diagramming. Data flow diagramming is a technique that diagrams the business processes and the data that pass among them. In this chapter, we first describe the basic syntax rules and illustrate how they can be used to draw simple one-page data flow diagrams (DFDs). Then we describe how to create more complex multipage diagrams.

Although the name *data flow diagram* (DFD) implies a focus on data, this is not the case. The focus is mainly on the processes or activities that are performed. Data modeling, discussed in the next chapter, presents how the data created and used by processes are organized. Process modeling—and creating DFDs in particular—is one of the most important skills needed by systems analysts.

In this chapter, we focus on *logical process models*, which are models that describe processes, without suggesting how they are conducted. When reading a logical process model, you will not be able to tell whether a process is computerized or manual, whether a piece of information is collected by paper form or via the Web, or whether information is placed in a filing cabinet or a large database. These physical details are defined during the design phase when these logical models are refined into *physical models*, which provide information that is needed to ultimately build the system. (See Chapter 10.) By focusing on logical processes first, analysts can focus on how the business should run, without being distracted by implementation details.

¹ Another commonly used process modeling technique is IDEF0. IDEF0 is used extensively throughout the U.S. Government. For more information about IDEF0, see FIPS 183: *Integration Definition for Function Modeling (IDEF0)*, Federal Information Processing Standards Publications, Washington, DC: U.S. Department of Commerce, 1993.

In this chapter, we first explain how to read DFDs and describe their basic syntax. Then we describe the process used to build DFDs that draws information from the use cases and from additional requirements information gathered from the users.

DATA FLOW DIAGRAMS

Reading Data Flow Diagrams

Figure 5-1 shows a DFD for the event we introduced in Chapter 4, that of a Lawn Chemical Applicator (LCA) requesting a lawn chemical. By examining the DFD, an analyst can understand the process by which LCAs request lawn chemicals. Take a moment to examine the diagram before reading on. How much do you understand? Before continuing, you may want to review this event's use case in the previous chapter (Figure 4-1) and the functional requirements (Figure 4-4).

Most people from Western cultures start reading diagrams from left to right, top to bottom. So, whenever possible, this is where most analysts try to make the DFD begin. The first item on the left side of Figure 5-1 is the “Lawn Chemical Applicator (LCA)” external entity, which is a rectangle that represents individual employees who must request the chemicals they will use for their lawn care assignments. This symbol has three arrows pointing away from it to rounded rectangular symbols. These arrows represent data flows and show that the external entity (LCA) provides three “bundles” of data to processes that use the data. Now look again at Figure 4-1 and you will see that these same data bundles are listed as Major Inputs in the use case, with the source listed as the LCA. Also, there are several arrows arriving at the LCA external entity from the rounded rectangles, representing bundles of data that the processes produce to flow back to the LCA. These data bundles are listed under Major Outputs in the use case (Figure 4-1), with the destination listed as the LCA.

Now look at the arrow that flows in to the “Determine Chemical Approval Status” process from the right side. In order to determine if the chemical requested is approved for usage, the process has to retrieve some information from storage. The open-ended rectangle labeled “Lawn Chemical Supply” is called a data store, and it represents a collection of stored data. The “Determine Chemical Approval Status” process uses an identifier for the chemical requested to find the requested chemical and to determine whether it is an approved chemical or a disapproved chemical. Notice that the “List of approved chemicals” is listed as a Major Input on the use case (Figure 4-1), with the source listed as the Lawn Chemicals Supply data store. Now, still referring to Figure 4-1, notice that every Major Input listed in the use case flows in to a process from an external entity or stored data (noted by the source). Also notice that every Major Output listed in the use case flows out to a destination (an external entity or data storage) on the data flow diagram.

Now look more closely at the Major Steps Performed section of the use case. You can see that a number of steps are listed in the use case. On the data flow diagram, these steps have been organized into five major processes, each performing one main component of the interactions detailed on the use case. On the DFD (Figure 5-1), as you follow the arrows starting with “Chemical needed” from the LCA to the “Determine Chemical Approval Status” process, imagine the LCA specifying the chemical he needs for a job. The system looks up the chemical and responds with a message verifying it as an approved chemical or informing the

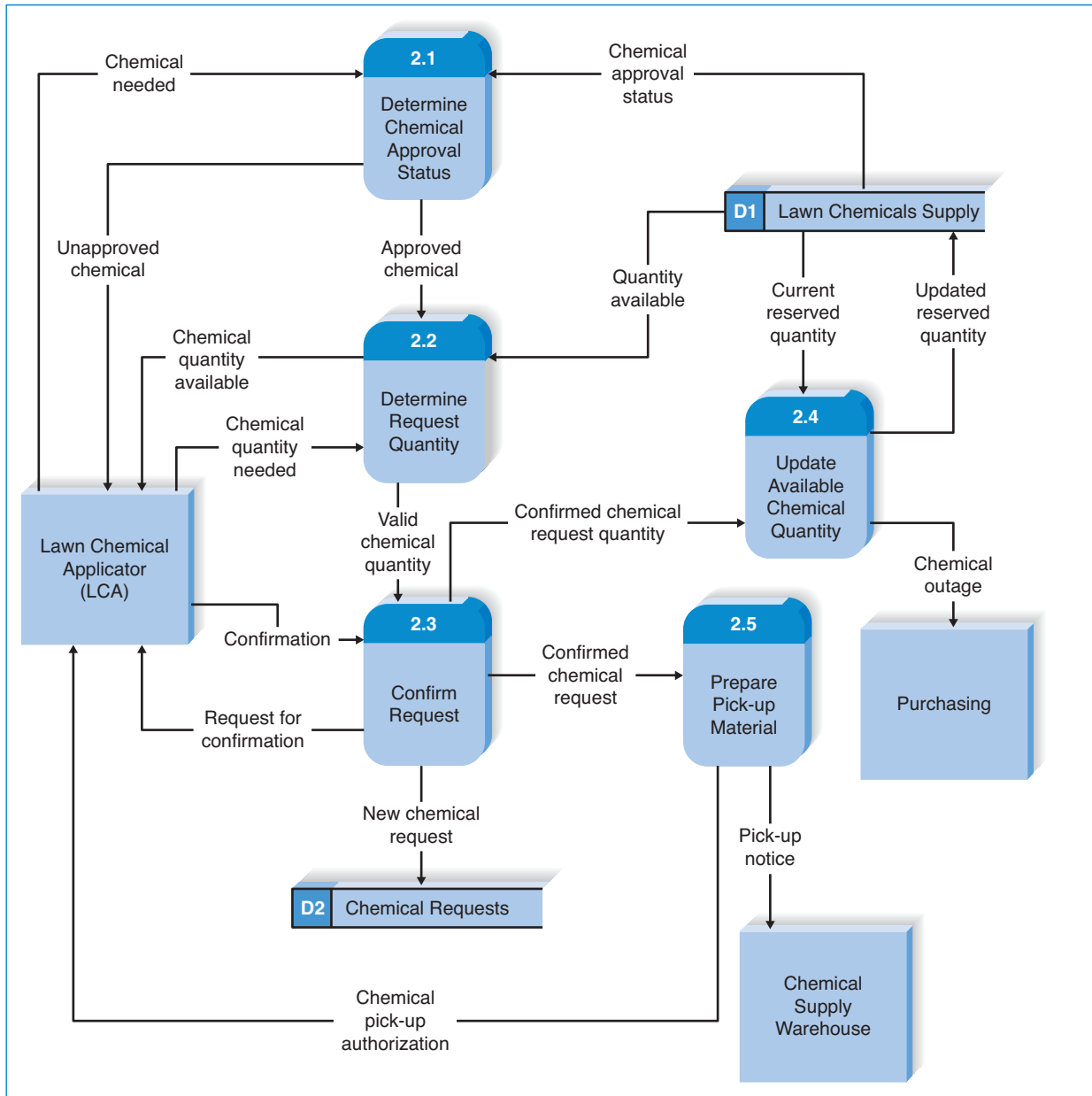


FIGURE 5-1
Request a Chemical Level 1 DFD

LCA that the chemical cannot be used. For an approved chemical, the system looks up how much of the chemical is available and informs the LCA. The LCA indicates the quantity he wants. Now look at the description on the use case (Figure 4-1) for steps 1-4 and notice how the use case describes those processes in words. Notice also how the “Information for Steps” section of the use case lists the data elements that are either used or produced by each step, corresponding to the inflows and outflows from the process symbols (2.1, 2.2) on the data flow diagram (Figure 5-1).

Look at the other three process symbols in the DFD and examine the flows into and out of each process. On the basis of the data flowing in and flowing out, try to understand what the process is doing. Check your understanding by looking at the Major Steps Performed and Information for Steps in the use case.

You probably recognized that the “Confirm Request” process (2.3) receives the LCA’s chemical request confirmation and creates and stores a new Chemical Request. You can also see that two additional processes are performed by the system. The quantity of the chemical available is modified by marking the quantity requested as “reserved” (and no longer available to another LCA). Finally, the pick-up authorization is provided to the LCA and the Chemical Supply Warehouse is notified of the approved pick-up. You can see by the flows from process 2.3 to processes 2.4 and 2.5 that sometimes a process sends a data flow directly to another process.

The use case in Figure 4-1 and the data flow diagram in Figure 5-1 are purposefully related. A well-constructed use case makes developing a data flow diagram quite straightforward, although the analyst will have to make some decisions about how much detail to depict in the DFD. The steps outlined in a use case can be organized into logical processes on a DFD. The Major Inputs and Major Outputs listed on the use case provide a list of the sources and destinations, respectively, of the inflows and outflows of the processes. The Information for Steps section shows the data flowing in or produced by each step of the use case, and these correspond to the data flows that enter or leave each process on the data flow diagram.

Elements of Data Flow Diagrams

Now that you have had a glimpse of a DFD, we will present the language of DFDs, which includes a set of symbols, naming conventions, and syntax rules. There are four symbols in the DFD language (processes, data flows, data stores, and external entities), each of which is represented by a different graphic symbol. There are two commonly used styles of symbols, one set developed by Chris Gane and Trish Sarson and the other by Tom DeMarco and Ed Yourdon² (Figure 5-2). Neither is better than the other; some organizations use the Gane and Sarson style of symbols, and others use the DeMarco/Yourdon style. We will use the Gane and Sarson style in this book.

Process A *process* is an activity or a function that is performed for some specific business reason. Processes can be manual or computerized. Every process should be named starting with a verb and ending with a noun (e.g., “Determine request quantity”). Names should be short, yet contain enough information so that the reader can easily understand exactly what they do. In general, each process performs only one activity, so most system analysts avoid using the word “and” in process names because it suggests that the process performs several activities. In addition, every process must have at least one input data flow and at least one output data flow.

Figure 5-2 shows the basic elements of a process and how they are usually named in CASE tools. Every process has a unique identification number, a name, and a description, all of which are noted in the CASE repository. Descriptions clearly and

² See Chris Gane and Trish Sarson, *Structured Systems Analysis: Tools and Techniques*, Englewood Cliffs, NJ: Prentice Hall, 1979; Tom DeMarco, *Structured Analysis and System Specification*, Englewood Cliffs, NJ: Prentice-Hall, 1979; and E. Yourdon and Larry L. Constantine, *Structured Design: Fundamentals of a Discipline of Computer Program and Systems Design*, Englewood Cliffs, NJ: Prentice-Hall, 1979.


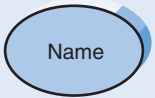


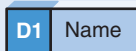
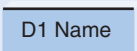
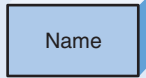
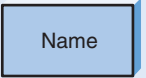
Data Flow Diagram Element	Typical Computer-Aided Software Engineering Fields	Gane and Sarson Symbol	DeMarco and Yourdon Symbol
Every <i>process</i> has a number a name (verb phase) a description at least one output data flow at least one input data flow	Label (name) Type (process) Description (what is it) Process number Process description (structured English) Notes		
Every <i>data flow</i> has a name (a noun) a description one or more connections to a process	Label (name) Type (flow) Description Alias (another name) Composition (description of data elements) Notes		
Every <i>data store</i> has a number a name (a noun) a description one or more input data flows one or more output data flows	Label (name) Type (store) Description Alias (another name) Composition (description of data elements) Notes		
Every <i>external entity</i> has a name (a noun) a description	Label (name) Type (entity) Description Alias (another name) Entity description Notes		

FIGURE 5-2
Data Flow Diagram Elements

precisely describe the steps and details of the processes; ultimately, they are used to guide the programmers who need to computerize the processes (or the writers of policy manuals for noncomputerized processes). The process descriptions become more detailed as information is learned about the process through the analysis phase. Many process descriptions are written as simple text statements about what happens. More complex processes use more formal techniques such as structured English, decision tables, or decision trees, which are discussed in a later section.

Data Flow A *data flow* is a single piece of data (e.g., quantity available) (sometimes called a data element), or a logical collection of several pieces of information (e.g., new chemical request). Every data flow should be named with a noun. The description of a data flow lists exactly what data elements the flow contains. For example, the pick-up notice data flow can list the LCA name, chemical, and quantity requested as its data elements.

Data flows are the glue that holds the processes together. One end of every data flow will always come from or go to a process, with the arrow showing the direction into or out of the process. Data flows show what inputs go into each process and what outputs each process produces. Every process must create at least

one output data flow, because if there is no output, the process does not do anything. Likewise, each process has at least one input data flow, because it is difficult, if not impossible, to produce an output with no input.

Data Store A *data store* is a collection of data that is stored in some way (which is determined later when creating the physical model). Every data store is named with a noun and is assigned an identification number and a description. Data stores form the starting point for the data model (discussed in the next chapter) and are the principal link between the process model and the data model.

Data flows coming out of a data store indicate that information is retrieved from the data store. Looking at Figure 5-1, you can see that process 2.1 (Determine Chemical Approval Status) retrieves the Chemical Approval Status data flow from the Lawn Chemicals Supply data store. Similarly, Process 2.2 (Determine Request Quantity) retrieves the Quantity Available data flow from the Lawn Chemicals Supply data store. Data flows going into a data store indicate that information is added to the data store. For example, process 2.3 adds a New Chemical Request data flow to the Chemical Requests data store. Finally, data flows going both into and out of a data store indicate that information in the data store is changed (e.g., by retrieving data from a data store, changing it, and storing it back). In Figure 5-1, we can see that process 2.4 (Update Available Chemical Quantity) retrieves the current reserved quantity from the Lawn Chemical Supply data store, modifies it, and writes the updated data back into the data store.

All data stores must have at least one input data flow (or else they never contain any data), unless they are created and maintained by another information system or on another page of the DFD. Likewise, they have at least one output data flow on some page of the DFD. (Why store data if you never use it?) In cases in which the same process both stores data and retrieves data from a data store, there is a temptation to draw one data flow with an arrow on both ends. This practice is incorrect, however. The data flow that stores data and the data flow that retrieves data should always be shown as two separate data flows.

External Entity An *external entity* is a person, organization, organization unit, or system that is external to the system, but interacts with it (e.g., customer, clearing-house, government organization, accounting system). The external entity typically corresponds to the primary actor identified in the use case. External entities provide data to the system or receive data from the system, and serve to establish the system boundaries. Every external entity has a name and a description. The key point to remember about an external entity is that it is external to the system, but may or may not be part of the organization. People who use the information from the system to perform other processes or who decide what information goes into the system are documented as external entities (e.g., managers, staff).

Using Data Flow Diagrams to Define Business Processes

Most business processes are too complex to be explained in one DFD. Most process models are therefore composed of a set of DFDs. The first DFD provides a summary of the overall system, with additional DFDs providing more and more detail about each part of the overall business process. Thus, one important principle in process modeling with DFDs is the decomposition of the business process into a hierarchy of DFDs, with each level down the hierarchy representing less scope but more detail. Figure 5-3 shows how one business process can be decomposed into several levels of DFDs.

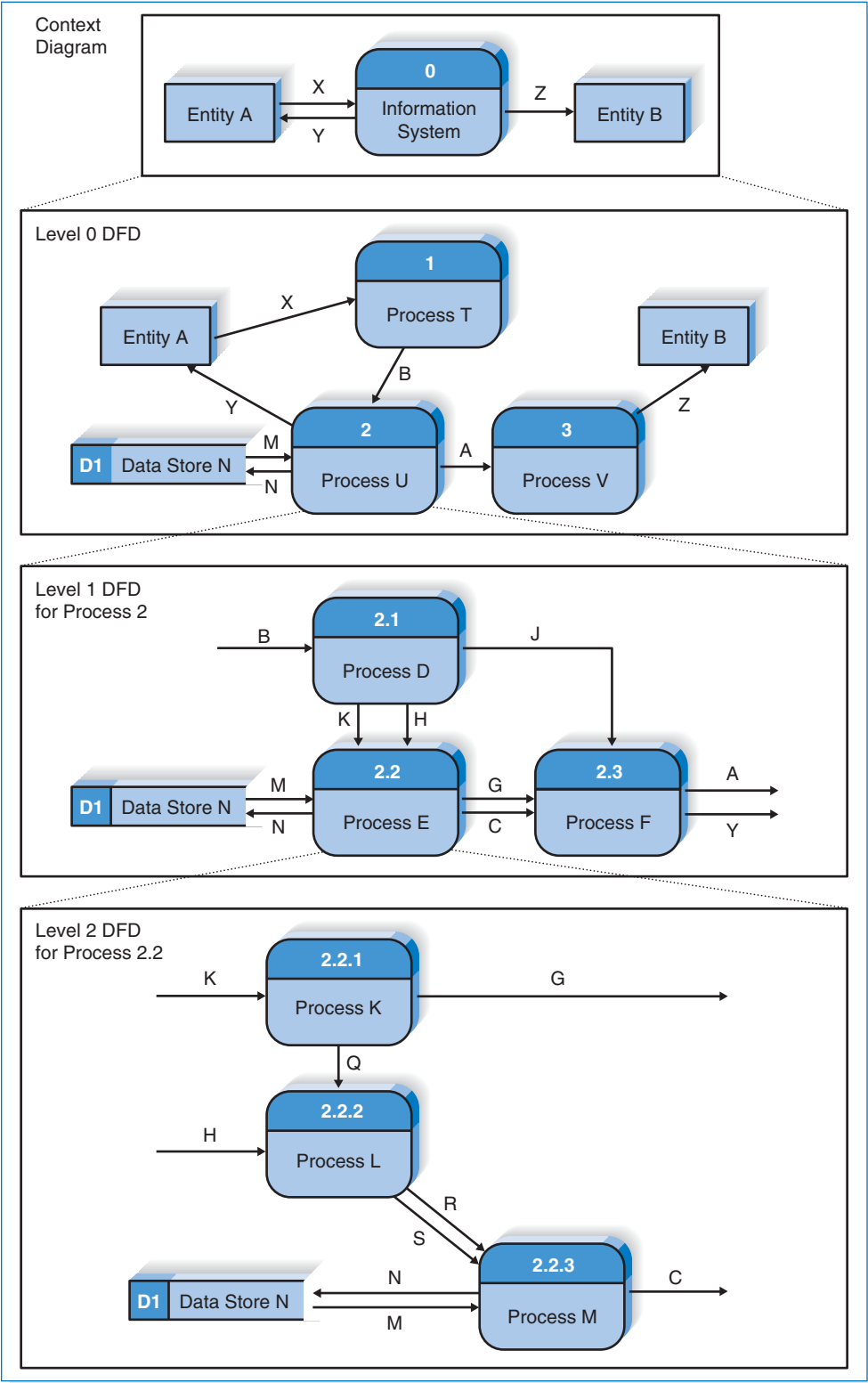


FIGURE 5-3
Relationships among Levels of Data Flow Diagrams (DFDs)

Context Diagram The first DFD in every business process model, whether a manual system or a computerized system, is the *context diagram* (see Figure 5-3). As the name suggests, the context diagram shows the entire system in context with its environment. All process models have one context diagram.

The context diagram shows the overall business process as just one process (i.e., the system itself) and shows the data flows to and from external entities. Data stores usually are not included on the context diagram, unless they are “owned” by systems or processes other than the one being documented. For example, an information system used by the university library that records who has borrowed books would likely check the registrar’s student information database to see whether a student is currently registered at the university. In this context diagram, the registrar’s student information data store could be shown on the context diagram because it is external to the library system, but used by it. Many organizations, however, would show this as an external entity called “Registrar’s Student Information System,” not as a data store.

Level 0 Diagram The next DFD is called the *level 0 diagram* or *level 0 DFD*. (See Figure 5-3.) The level 0 diagram shows all the processes at the first level of numbering (i.e., processes numbered 1 through 3), the data stores, external entities, and data flows among them. The purpose of the level 0 DFD is to show all the major high-level processes of the system and how they are interrelated. All process models have one and only one level 0 DFD.

Another key principle in creating sets of DFDs is *balancing*. Balancing means ensuring that all information presented in a DFD at one level is accurately represented in the next-level DFD. This doesn’t mean that the information is identical, but that it is shown appropriately. There is a subtle difference in meaning between these two words that will become apparent shortly, but for the moment, let’s compare the context diagram with the level 0 DFD in Figure 5-3 to see how the two are balanced. In this case, we see that the external entities (A, B) are identical between the two diagrams and that the data flows to and from the external entities in the context diagram (X, Y, Z) also appear in the level 0 DFD. The level 0 DFD replaces the context diagram’s single process (always numbered 0) with three processes (1, 2, 3), adds a data store (D1), and includes two additional data flows that were not in the context diagram (data flow B from process 1 to process 2; data flow A from process 2 to process 3).

These three processes and two data flows are contained within process 0. They were not shown on the context diagram because they are the internal components of process 0. The context diagram deliberately hides some of the system’s complexity in order to make it easier for the reader to understand. Only after the reader understands the context diagram does the analyst “open up” process 0 to display its internal operations by decomposing the context diagram into the level 0 DFD, which shows more detail about the processes and data flows inside the system.

Level 1 Diagrams In the same way that the context diagram deliberately hides some of the system’s complexity, so, too, does the level 0 DFD. The level 0 DFD shows only how the major high-level processes in the system interact. Each process on the level 0 DFD can be decomposed into a more explicit DFD, called a *level 1 diagram*, or *level 1 DFD*, which shows how it operates in greater detail. The DFD illustrated in Figure 5-1 is a level 1 DFD.

In general, all process models have as many level 1 diagrams as there are processes on the level 0 diagram; every process in the level 0 DFD would be *decomposed* into its own level 1 DFD, so the level 0 DFD in Figure 5-3 would have three level 1 DFDs (one for process 1, one for process 2, one for process 3). For simplicity, we have chosen to show only one level 1 DFD in this figure, the DFD for process 2. The processes in level 1 DFDs are numbered on the basis of the process being decomposed. In this example, we are decomposing process 2, so the processes in this level 1 DFD are numbered 2.1, 2.2, and 2.3.

Processes 2.1, 2.2, and 2.3 are the *children* of process 2, and process 2 is the *parent* of processes 2.1, 2.2, and 2.3. These three children processes wholly and completely make up process 2. The set of children and the parent are identical; they are simply different ways of looking at the same thing. When a parent process is decomposed into children, its children must completely perform all of its functions, in the same way that cutting up a pie produces a set of slices that wholly and completely make up the pie. Even though the slices may not be the same size, the set of slices is identical to the entire pie; nothing is omitted by slicing the pie.

Once again, it is very important to ensure that the level 0 and level 1 DFDs are balanced. The level 0 DFD shows that process 2 accesses data store D1, has two input data flows (B, M), and has three output data flows (A, N, and Y). A check of the level 1 DFD shows the same data store and data flows. Once again, we see that five new data flows have been added (C, G, H, J, K) at this level. These data flows are contained within process 2 and therefore are not documented in the level 0 DFD. Only when we decompose or open up process 2 via the level 1 DFD do we see that they exist.

The level 1 DFD shows more precisely which process uses the input data flow B (process 2.1) and which produces the output data flows A and Y (process 2.3). Note, however, that the level 1 DFD does not show where these data flows come from or go to. To find the source of data flow B, for example, we have to move up to the level 0 DFD, which shows data flow B coming from external entity B. Likewise, if we follow the data flow from A up to the level 0 DFD, we see that it goes to process 3, but we still do not know exactly which process within process 3 uses it (e.g., process 3.1, 3.2). To determine the exact source, we would have to examine the level 1 DFD for process 3.

This example shows one downside to the decomposition of DFDs across multiple pages. To find the exact source and destination of data flows, one often must follow the data flow across several DFDs on different pages. Several alternatives to this approach to decomposing DFDs have been proposed, but none is as commonly used as the “traditional” approach. The most common alternative is to show the source and destination of data flows to and from external entities (as well as data stores) at the lower level DFDs. The fact that most data flows are to or from data stores and external entities, rather than processes on other DFD pages, can significantly simplify the reading of multiple page DFDs. We believe this to be a better approach, so when we teach our courses, we show external entities on all DFDs, including level 1 DFDs and below.

Level 2 Diagrams The bottom of Figure 5-3 shows the next level of decomposition: a *level 2 diagram*, or *level 2 DFD*, for process 2.2. This DFD shows that process 2.2 is decomposed into three processes (2.2.1, 2.2.2, and 2.2.3). The level 1 diagram for process 2.2 shows interactions with data store D1, which we see in the level 2 DFD as occurring in process 2.2.3. Likewise, the level 2 DFD

for 2.2 shows two input data flows (H, K) and two output data flows (C, G), which we also see on the level 2 diagram, along with several new data flows (Q, R, S). The two DFDs are therefore balanced.

It is sometimes difficult to remember which DFD level is which. It may help to remember that the level numbers refer to the number of decimal points in the process numbers on the DFD. A level 0 DFD has process numbers with no decimal points (e.g., 1, 2), whereas a level 1 DFD has process numbers with one decimal point (e.g., 2.3, 5.1), a level 2 DFD has numbers with two decimal points (e.g., 1.2.5, 3.3.2), and so on.

Alternative Data Flows Suppose that a process produces two different data flows under different circumstances. For example, a quality-control process could produce a quality-approved widget or a defective widget, or our search for a chemical could find it is approved or not approved for use. How do we show these alternative paths in the DFD? The answer is that we show both data flows and use the process description to explain that they are alternatives. Nothing on the DFD itself shows that the data flows are mutually exclusive. For example, process 2.1 on the level 1 DFD produces three output data flows (H, J, K). Without reading the text description of process 2.1, we do not know whether these are produced simultaneously or whether they are mutually exclusive.

Process Descriptions

The purpose of the process descriptions is to explain what the process does and provide additional information that the DFD does not provide. As we move through the SDLC, we gradually move from the general text descriptions of requirements into more and more precise descriptions that are eventually translated into very precise programming languages. In most cases, a process is straightforward enough that the requirements definition, a use case, and a DFD with a simple text description together provide sufficient detail to support the activities in the design phase. Sometimes, however, the process is sufficiently complex that it can benefit from a more detailed process description that explains the logic which occurs inside the process. Three techniques are commonly used to describe more complex processing logic: structured English, decision trees, and decision tables. Very complex processes may use a combination of structured English and either decision trees or decision tables.

Structured English uses short sentences to describe the work that a process performs. *Decision trees* display decision logic (IF statements) as a set of nodes (questions) and branches (answers). *Decision tables* represent complex policy decisions as rules that link various conditions with actions. Since these techniques are commonly discussed in programming texts, we will not elaborate on them here. They are useful to the systems analyst in conveying the proper understanding of what goes on “inside” a process.

CREATING DATA FLOW DIAGRAMS

Data flow diagrams start with the information in the use cases and the requirements definition. Although the use cases are created by the users and project team working together, the DFDs typically are created by the project team and then reviewed by the users. Generally speaking, the set of DFDs that make up the process model

simply integrates the individual use cases (and adds in any processes in the requirements definition not selected as use cases). The project team takes the use cases and rewrites them as DFDs. However, because DFDs have formal rules about symbols and syntax that use cases do not, the project team sometimes has to revise some of the information in the use cases to make them conform to the DFD rules. The most common types of changes are to the names of the use cases that become processes and the inputs and outputs that become data flows. The second most common type of change is to combine several small inputs and outputs in the use cases into larger data flows in the DFDs (e.g., combining three separate inputs, such as “customer name,” “customer address,” and “customer phone number,” into one data flow, such as “customer information”).

Project teams usually use process modeling tools or CASE tools to draw process models. Simple tools such as Visio contain DFD symbol sets and enable easy creation and modification of diagrams. Other process modeling tools such as BPWin understand the DFD and can perform simple syntax checking to make sure that the DFD is at least somewhat correct. A full CASE tool, such as Visible Analyst Workbench, provides many capabilities in addition to process modeling (e.g., data modeling as discussed in the next chapter). CASE tools tend to be complex, and while they are valuable for large and complex projects, they often cost more than they add for simple projects. Figure 5-4 shows a sample screen from the Visible Analyst CASE tool.

Building a process model that has many levels of DFDs usually entails several steps. Some analysts prefer to begin process modeling by focusing first on the level 0 diagram. We have found it useful to first build the context diagram showing all the external entities and the data flows that originate from or terminate in them. Second, the team creates a DFD fragment for each use case that shows how the use case exchanges data flows with the external entities and data stores. Third, these DFD fragments are organized into a level 0 DFD. Fourth, the team develops level 1 DFDs, based on the steps within each use case, to better explain how they operate. In some cases, these level 1 DFDs are further decomposed into level 2 DFDs, level 3 DFDs, level 4 DFDs, and so on. Fifth, the team validates the set of DFDs to make sure that they are complete and correct.

In the following sections, process modeling is illustrated with the Holiday Travel Vehicles information system.

Creating the Context Diagram

The context diagram defines how the business process or computer system interacts with its environment—primarily the external entities. To create the context diagram, you simply draw one process symbol for the business process or system being modeled (numbered 0 and named for the process or system). You read through the use cases and add the inputs and outputs listed on the form, as well as their sources and destinations. Usually, all the inputs and outputs will come from or go to external entities such as a person, organization, or other information system. If any inputs and outputs connect directly to data stores in an external system, it is best practice to create an external entity which is named for the system that owns the data store. None of the data stores inside the process/system that are created by the process or system itself are included in the context diagram, because they are “inside” the system. Because there are sometimes so many inputs and outputs, we often combine several small data flows into larger data flows.

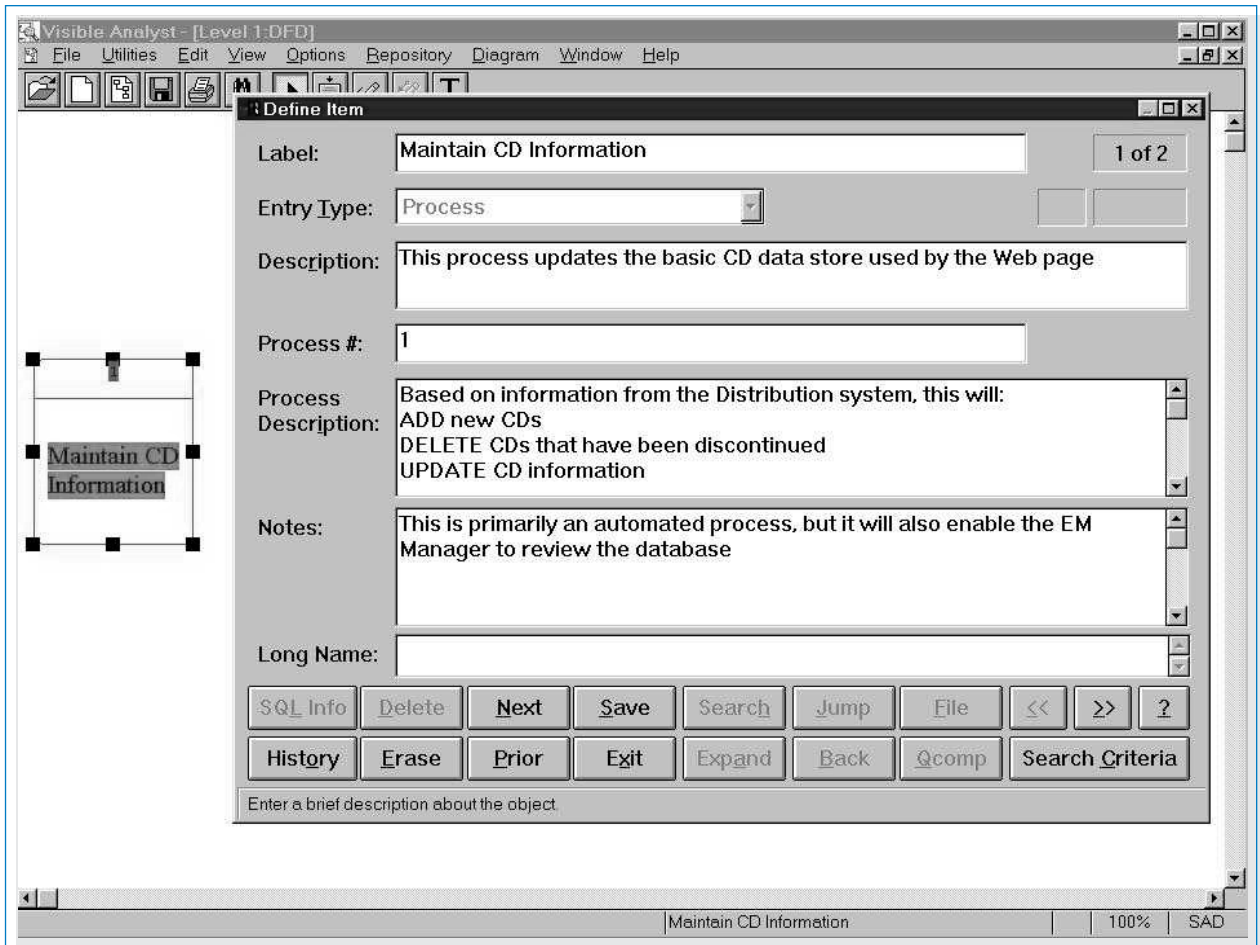


FIGURE 5-4
Entering Data Flow Diagram Processes in a Computer-Aided Software Engineering Tool

Figure 5-5 shows the context diagram for the Holiday Travel Vehicles system focusing on vehicle sales. Take a moment to review this system as described in Chapter 4 and review the use cases in Figure 4-11. You can see from the Major Inputs and Outputs sections in the Figure 4-11 use cases that the system has many interactions with the salesperson external entity. We have simplified these inflows and outflows to just three primary data flows on the context diagram. If we had included each small data flow, the context diagram would become too cluttered. The smaller data flows will become evident as we decompose the context diagram into more detailed levels. Notice that we have established three external entities to represent parts of the Holiday Travel Vehicle organization which receive information from or supply information to this system. Salespeople provide key inputs to the system, and shop work orders flow to the shop manager. The company owner or manager provides information to the system.

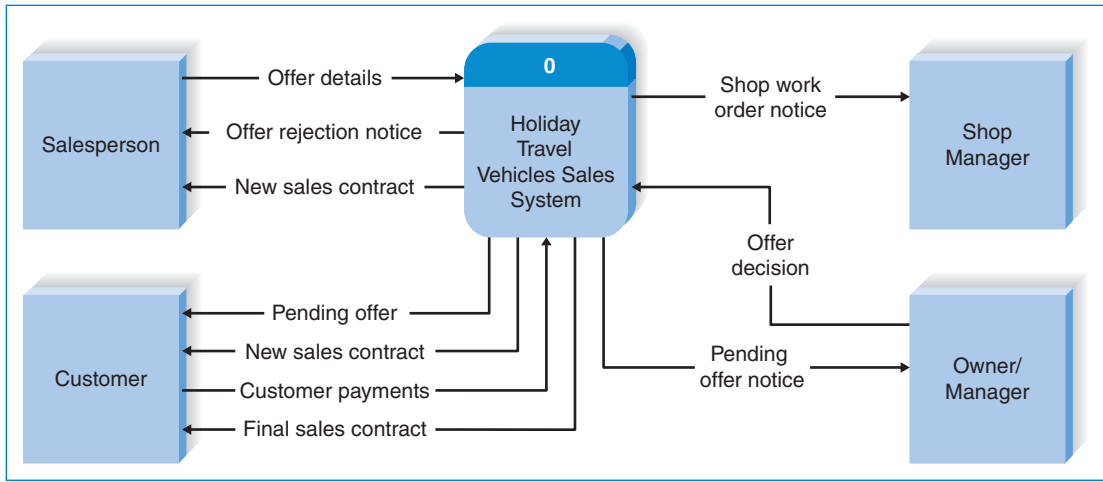


FIGURE 5-5
Holiday Travel Vehicles Sales System Context Diagram

Creating Data Flow Diagram Fragments

A *DFD fragment* is one part of a DFD that eventually will be combined with other DFD fragments to form a DFD diagram. In this step, each use case is converted into one DFD fragment. You start by taking each use case and drawing a DFD fragment, using the information given on the top of the use case: the name, ID number, and major inputs and outputs. The information about the major steps that make up each use case is ignored at this point; it will be used in a later step. Figure 5-6 shows a use case and the DFD fragment that was created from it.

Once again, some subtle, but important changes are often made in converting the use case into a DFD. The two most common changes are modifications to the process names and the addition of data flows. There were no formal rules for use case names, but there are formal rules for naming processes on the DFD. All process names must be a verb phrase—they must start with a verb and include a noun. (See Figure 5-2.) Not all of our use case names are structured in this way, so we sometimes need to change them. It is also important to have a consistent *viewpoint* when naming processes. For example, the DFD in Figure 5-6 is written from the viewpoint of the dealership, not of the customer. All the process names and descriptions are written as activities that the staff performs. It is traditional to design the processes from the viewpoint of the organization running the system, so this sometimes requires some additional changes in names.

The second common change is the addition of data flows. Use cases are written to describe how the system and user interact. Typically, they do not describe how the system obtains data, so the use case often omits data flows read from a data store. When creating DFD fragments, it is important to make sure that any information given to the user is obtained from a data store. The easiest way to do this is to first create the DFD fragment with the major inputs and outputs listed on the use case and then verify that all outputs have sufficient inputs to create them.

There are no formal rules covering the *layout* of processes, data flows, data stores, and external entities within a DFD. Most systems analysts try to put the process in the middle of the DFD fragment, with the major inputs starting from the

Use Case Name: <u>Record an offer</u>		ID: <u>UC-3</u>	Priority: <u>High</u>
Actor: <u>Salesperson</u>			
Description: <u>This use case describes how the salesperson records a customer offer on a vehicle. The offer may be a new offer or a revision of a previously rejected offer.</u>			
Trigger: <u>Customer decides to make an offer on a vehicle.</u>			
Type: <input checked="" type="checkbox"/> External <input type="checkbox"/> Temporal			
Summary			
Inputs	Source	Outputs	Destination
Vehicle ID Existing Pending Offer	Salesperson Pending Offers datastore	Offer Pending Notice Offer Summary New Pending Offer	Salesperson Customer Pending Offer datastore
Offer Type Offer ID Previous offer details	Salesperson Rejected Offers datastore Vehicle details Customer Salesperson	Pending Offer Pending Offer Notice	Customer Owner/Manager
Vehicle datastore Customer details Offer details			

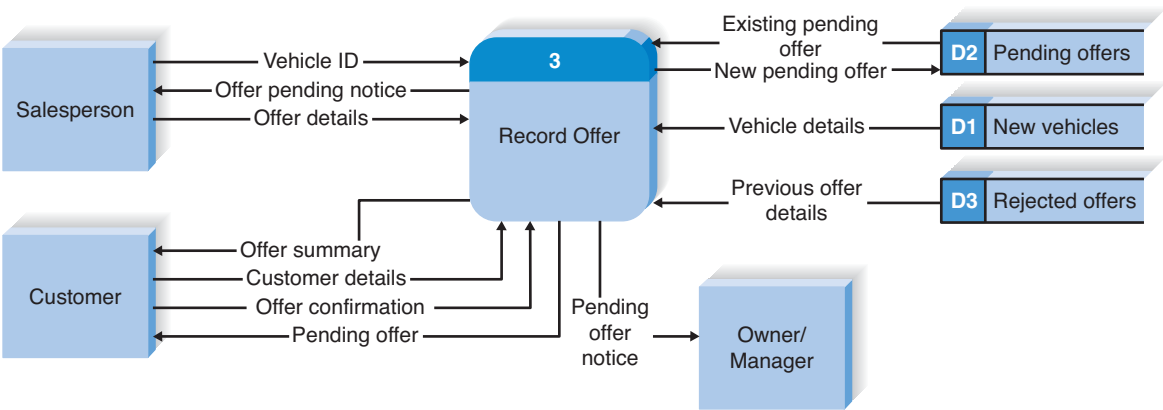
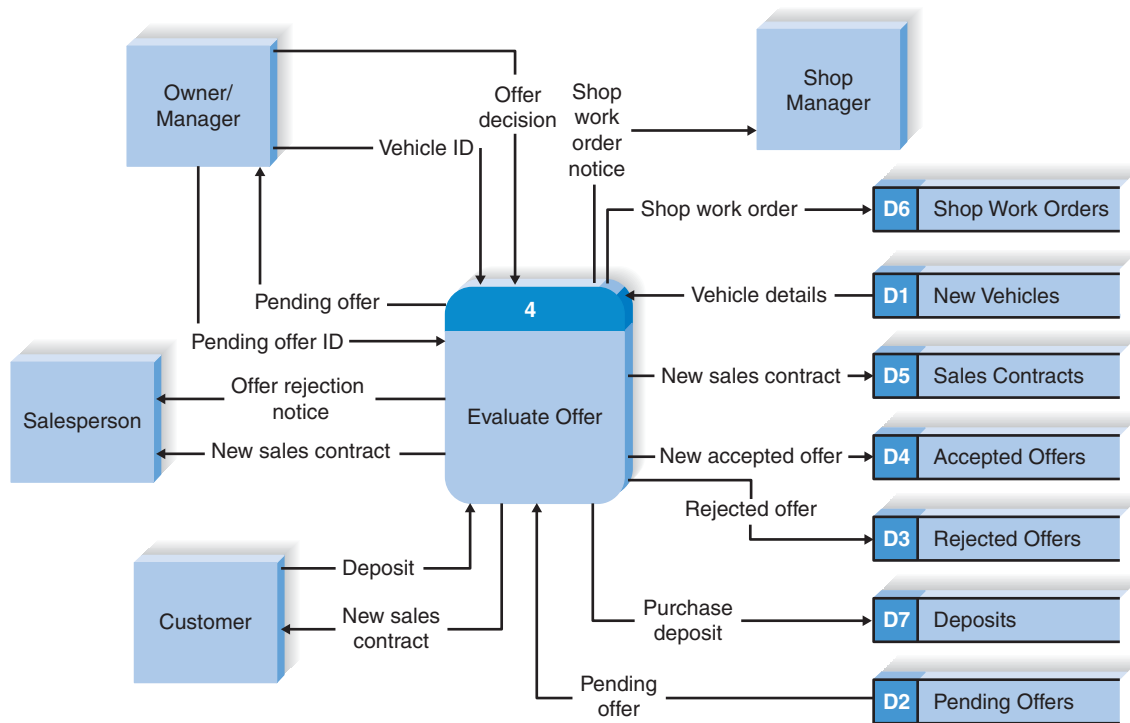


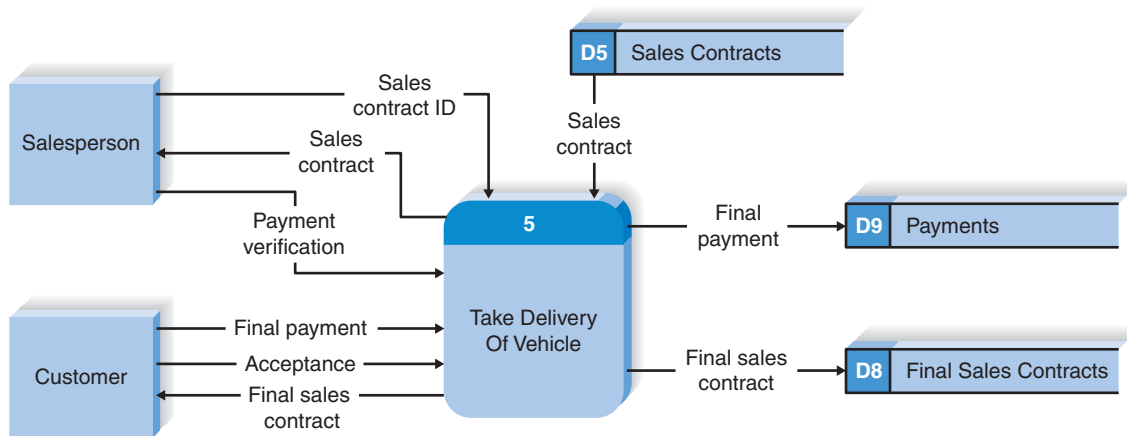
FIGURE 5-6
Holiday Travel Vehicles Process 3 (Record Offer) DFD Fragment

left side or top entering the process and outputs leaving from the right or the bottom. Data stores are often written below the process.

Take a moment and draw a DFD fragment for the two other use cases shown in Figure 4-11 (Evaluate Offer and Take Delivery of Vehicle). We have included possible ways of drawing these fragments in Figure 5-7. (Don't look until you've attempted the drawings on your own!)



(a) Process 4 DFD Fragment



(b) Process 5 DFD Fragment

FIGURE 5-7
Additional DFD Fragments for Holiday Travel Vehicles

Creating the Level 0 Data Flow Diagram

Once you have the set of DFD fragments (one for each of the major use cases), you combine them into one DFD drawing that becomes the level 0 DFD. As mentioned earlier, there are no formal layout rules for DFDs. Most systems analysts try to put the process that is first chronologically in the upper left corner of the diagram and work their way from top to bottom, left to right (e.g., Figure 5-1). Generally speaking, most analysts try to reduce the number of times that data flow lines cross or to ensure that when they do cross, they cross at right angles so that there is less confusion. (Many give one line a little “hump” to imply that one data flow jumps over the other without touching it.) Minimizing the number of data flows that cross is challenging.

Iteration is the cornerstone of good DFD design. Even experienced analysts seldom draw a DFD perfectly the first time. In most cases, they draw it once to understand the pattern of processes, data flows, data stores, and external entities and then draw it a second time on a fresh sheet of paper (or in a fresh file) to make it easier to understand and to reduce the number of data flows that cross. Often, a DFD is drawn many times before it is finished.

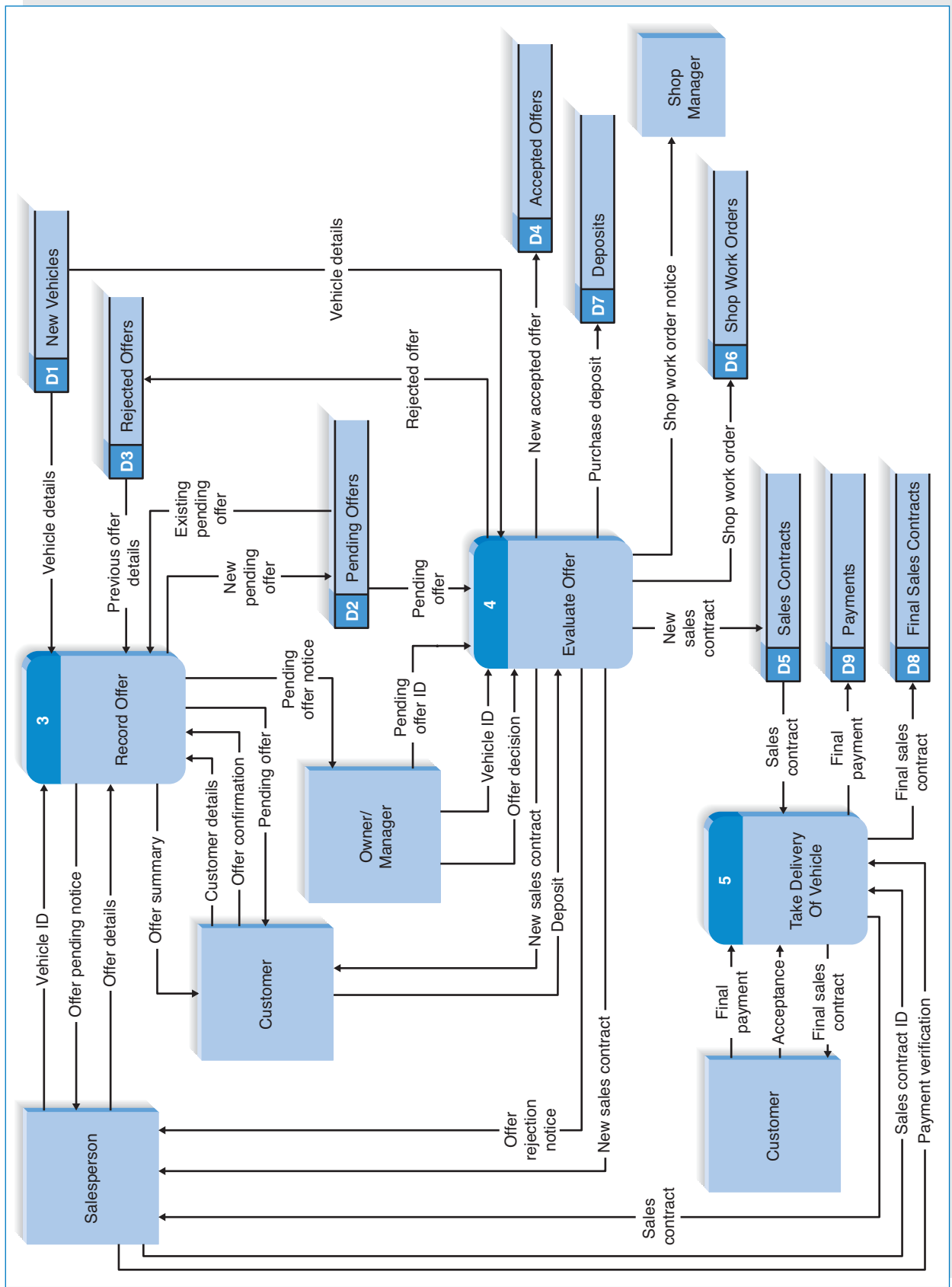
Figure 5-8 combines the DFD fragments in Figures 5-6 and 5-7. Take a moment to examine Figure 5-8 and find the DFD fragments from Figures 5-6 and 5-7 contained within it.

Creating Level 1 Data Flow Diagrams (and Below)

The team now begins to create lower-level DFDs for each process in the level 0 DFD that needs a level 1 DFD. Each one of the use cases is turned into its own DFD. The process for creating the level 1 DFDs is to take the steps as written on the use cases and convert them into a DFD in much the same way as for the level 0 DFD. Usually, each major step in the use case becomes a process on the level 1 DFD, with the inputs and outputs becoming the input and output data flows. Once again, however, sometimes subtle changes are required to go from the informal descriptions in the use case to the more formal process model, such as adding input data flows that were not included in the use case. And because the analysts are now starting to think more deeply about how the processes will be supported by an information system, they sometimes slightly change the use case steps to make the process easier to use.

In some approaches to creating DFDs, no source and destination are given on the level 1 DFD (and lower) for the inputs that come and go between external entities (or other processes outside of this process). But the source and destination of data flows for data stores and data flows that go to processes within this DFD are included (i.e., from one step to another in the same use case, such as “Confirmed Chemical Request” from process 2.3 to 2.5 in Figure 5-1). This is because the information is redundant; you can see the destination of data flows by reading the level 0 DFD.

The problem with these approaches is that in order to really understand the level 1 DFD, you must refer back to the level 0 DFD. For small systems that only have one or two level 1 DFDs, this is not a major problem. But for large systems that have many levels of DFDs, the problem grows; in order to understand the destination of a data flow on a level 3 DFD, you have to read the level 2 DFD, the level 1 DFD, and the level 0 DFD—and if the destination is to another activity, then you have to trace down in the lower-level DFDs in the other process.

**FIGURE 5-8**

Holiday Travel Vehicles Level 0 DFD

We believe that including external entities in level 1 and lower DFDs dramatically simplifies the readability of DFDs, with very little downside. In our work in several dozen projects with the U.S. Department of Defense, several other federal agencies, and the military of two other countries, we came to understand the value of this approach and converted the powers that be to our viewpoint. Because DFDs are not completely standardized, each organization uses them slightly differently. So, the ultimate decision of whether or not to include external entities on level 1 DFDs is yours—or your instructor’s! In this book, we will include them.

Ideally, we try to keep the data stores in the same general position on the page in the level 1 DFD as they were in the level 0 DFD, but this is not always possible. We try to draw input data flows arriving from the left edge of the page and output data flows leaving from the right edge. For example, see the level 1 DFD in Figure 5-1.

One of the most challenging design questions is when to decompose a level 1 DFD into lower levels. The decomposition of DFDs can be taken to almost any level, so for example, we could decompose process 1.2 on the level 1 DFD into processes 1.2.1, 1.2.2, 1.2.3, and so on in the level 2 DFD. This can be repeated to any level of detail, so one could have level 4 or even level 5 DFDs.

There is no simple answer to the “ideal” level of decomposition, because it depends on the complexity of the system or business process being modeled. In general, you decompose a process into a lower-level DFD whenever that process is sufficiently complex that additional decomposition can help explain the process. Most experts believe that there should be at least three, and no more than seven to nine, processes on every DFD, so if you begin to decompose a process and end up with only two processes on the lower-level DFD, you probably don’t need to decompose it. There seems little point in decomposing a process and creating another lower-level DFD for only two processes; you are better off simply showing two processes on the original higher level DFD. Likewise, a DFD with more than nine processes becomes difficult for users to read and understand, because it is very complex and crowded. Some of these processes should be combined and explained on a lower-level DFD.

One guideline for reaching the ideal level of decomposition is to decompose until you can provide a detailed description of the process in no more than one page of process descriptions: structured English, decision trees, or decision tables. Another helpful rule of thumb is that each lowest level process should be no more complex than what can be realized in about 25–50 lines of code.

In Figures 5-9, 5-10, and 5-12, we have provided level 1 DFDs for the vehicles sales processes we have focused on for the Holiday Travel Vehicle system. As we describe each figure, take a moment to back at the Major Steps section of their respective use cases in Figure 4-11.

Figure 5-9 depicts the level 1 DFD for the process of Recording an Order (process 3). The salesperson first checks for any existing Pending Offers for the vehicle (3.1). If a pending offer is found, the salesperson is notified and the process ends. Otherwise, the salesperson is specified if the offer is a new offer or a revision of a previous (rejected) offer (3.2). If it is a revised offer, the previous offer is retrieved from the Rejected Offers data store (3.3). If it is a new offer, information about the vehicle and the customer is obtained (3.4). The salesperson completes the specific details of the offer (3.5), and the offer must be confirmed by the customer (3.6). Once confirmed, the Pending Offer is stored, a copy is given to the customer, and the Owner/Manager is notified of the new Pending Offer (3.7). As you look at the use case for this event, you will see that the steps outlined have been captured in these

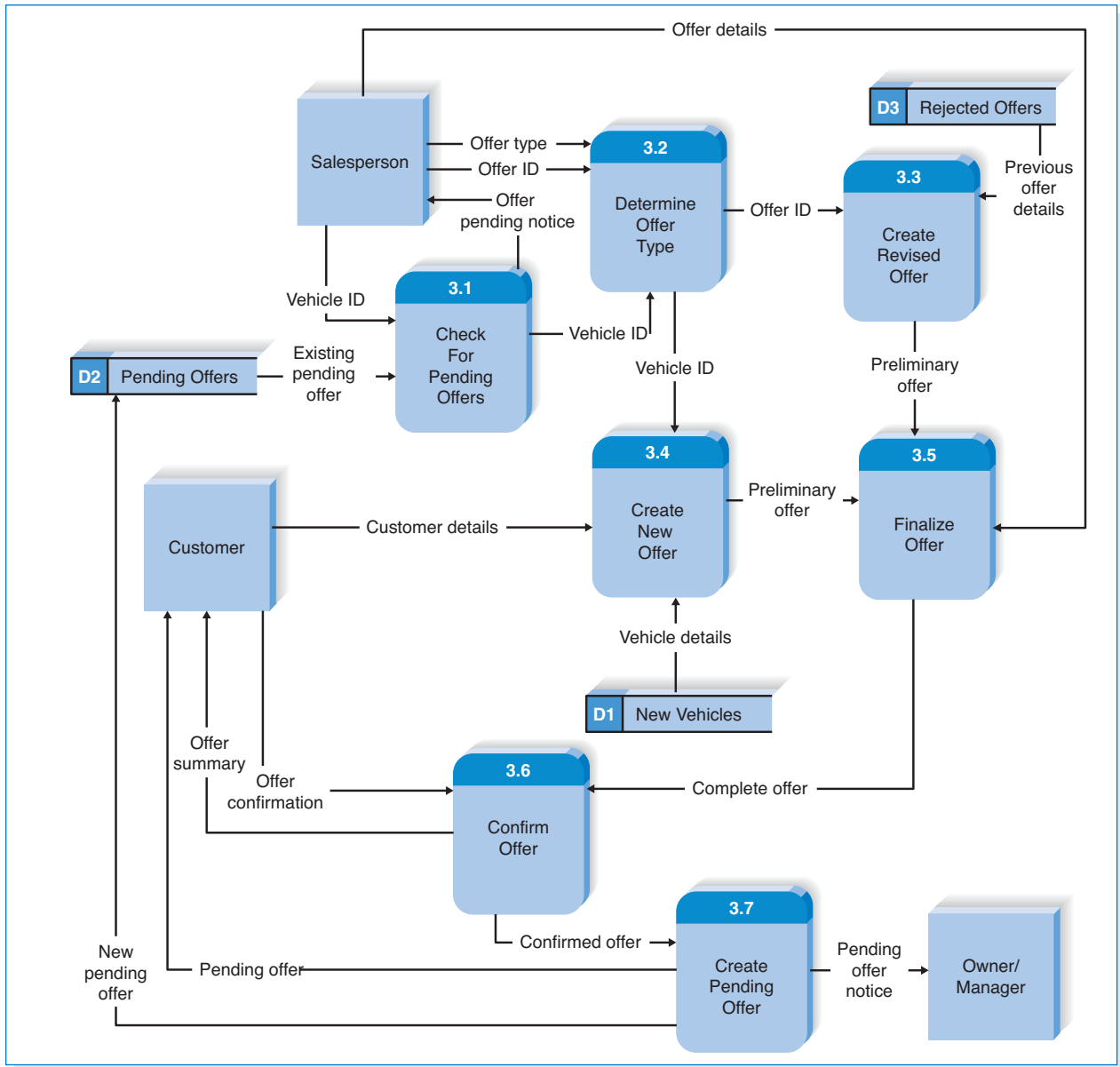
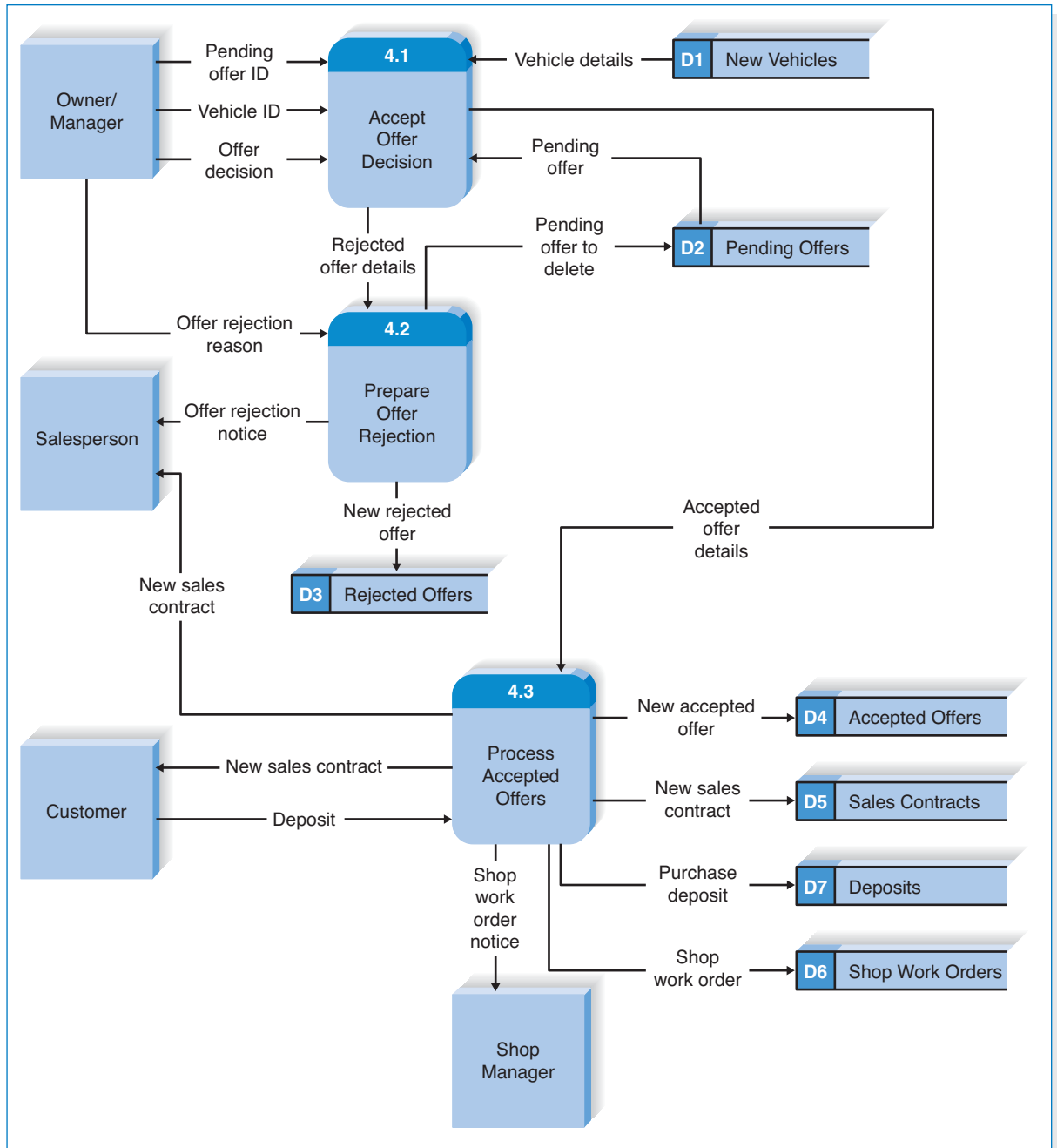


FIGURE 5-9
Holiday Travel Vehicles Process 3 (Record Offer) Level 1 DFD

seven processes in Figure 5-9. Also notice that we have attempted to organize the use case steps into processes that focus on one primary task.

Figure 5-10 describes the process of Evaluating an Offer. We have taken a slightly different approach to creating this level 1 DFD. The Owner/Manager is able to look at the Pending Offer and details about the vehicle and enters his decision to accept or reject the offer (4.1). For rejected offers, the Owner/Manager enters the reason for the rejection, the rejected offer is recorded, and the salesperson is notified of the offer rejection (4.2). For accepted offers, a variety of tasks are

**FIGURE 5-10**

Holiday Travel Vehicles Process 4 (Evaluate Offer) Level 1 DFD

performed in order to complete the accepted offer (4.3). We have purposely collected these tasks into process 4.3 so that we can demonstrate the process of “exploding” that process into a level 2 DFD.

Process 4.3 in Figure 5-10 is clearly a process that is performing many tasks. When a process has numerous inflows and outflows, it is a good candidate for

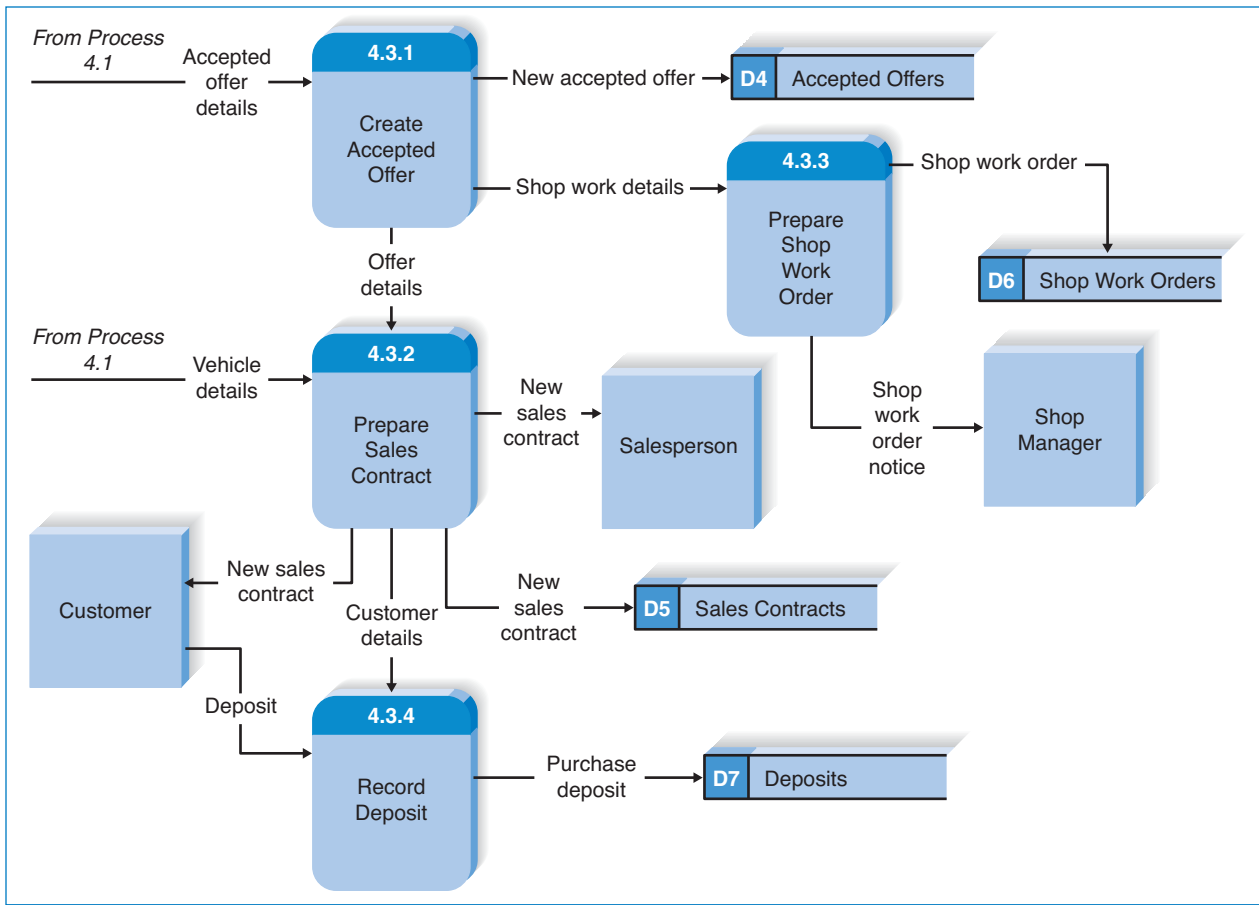
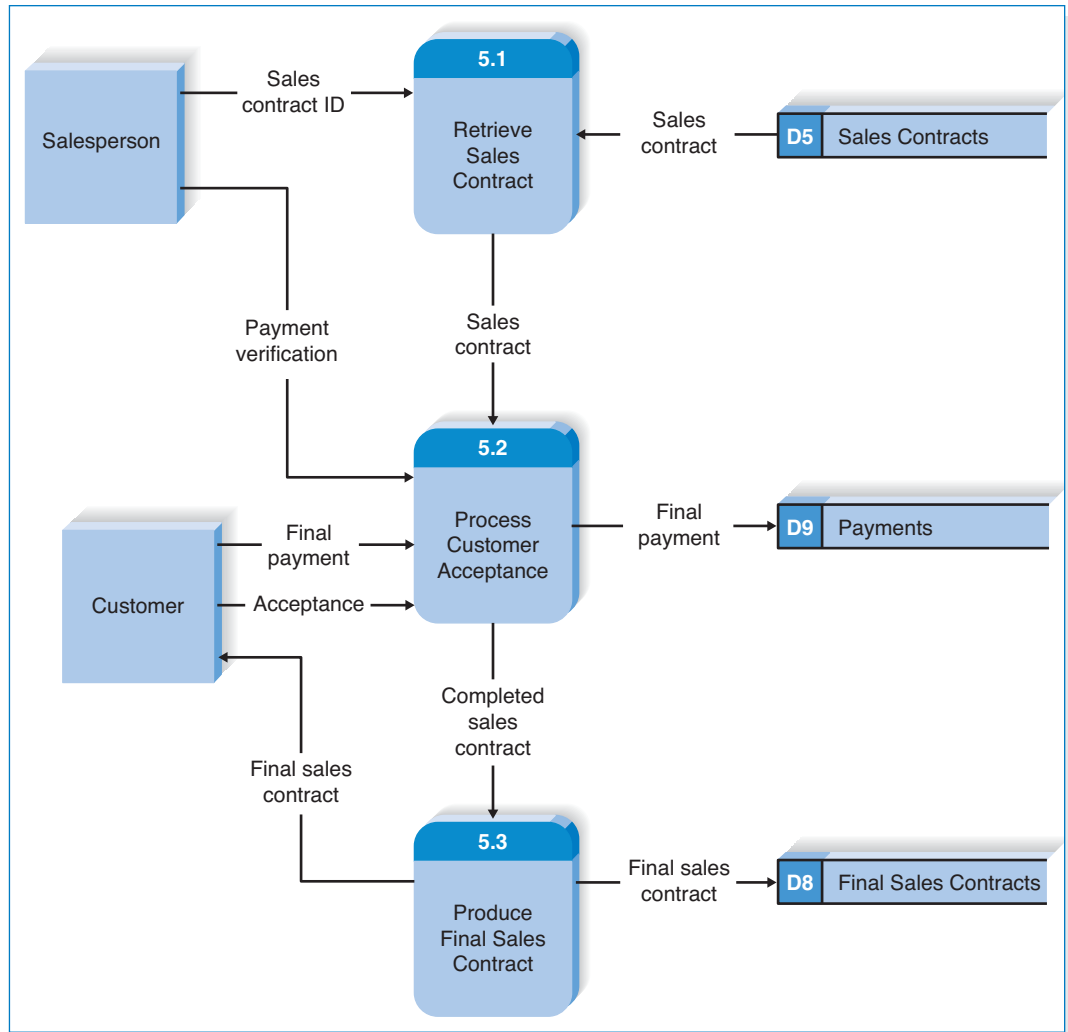


FIGURE 5-11
Holiday Travel Vehicles Process 4.3 (Process Accepted Offers) Level 2 DFD

decomposition into a lower-level DFD. As we mentioned previously, analysts often purposely “hide” details on a higher-level diagram to eliminate confusion and use a lower-level diagram to “reveal” the details. Figure 5-11 contains the level 2 DFD for process 4.3, Process Accepted Offers. As you can see by studying Figure 5-11, the parent process has been divided into four subprocesses, each focusing on one main task. Process 4.3.1 records the new Accepted Offer and provides data to two other processes. Process 4.3.3 uses the details from the accepted offer regarding options the customer wants to prepare a Shop Work Order and notify the shop manager. Process 4.3.2 uses the offer details and vehicle details to prepare the final sales contract. Finally, the customer’s deposit is recorded (4.3.4).

Figure 5-12 depicts the third of our use cases from Figure 4-11 (Take Delivery of Vehicle). The salesperson obtains the Sales Contract (5.1). The salesperson verifies receipt of the customer’s final payment, the customer signifies his/her acceptance of the vehicle, and the payment is recorded (5.2). Finally, the Final Sales Contract is recorded and a copy provided to the customer (5.3).

The process model is more likely to be drawn to the lowest level of detail for a to-be model if a traditional development process is used (i.e., not rapid

**FIGURE 5-12**

Holiday Travel Vehicles Process 5 (Take Delivery of Vehicle) Level 1 DFD

application development [RAD]; see Chapter 2) or if the system will be built by an external contractor. Without the complete level of detail, it may be hard to specify in a contract exactly what the system should do. If a RAD approach, which involves a lot of interaction with the users and, quite often, prototypes, is being used, we would be less likely to go to as low a level of detail, because the design will evolve through interaction with the users. In our experience, most systems go to only level 2 at most.

There is no requirement that all parts of the system must be decomposed to the same level of DFDs. Some parts of the system may be very complex and require many levels, whereas other parts of the system may be simpler and require fewer.

CONCEPTS

5-A U.S. ARMY AND MARINE CORPS BATTLEFIELD LOGISTICS

IN ACTION

Shortly after the Gulf War in 1991 (Desert Storm), the U.S. Department of Defense realized that there were significant problems in its battlefield logistics systems that provided supplies to the troops at the division level and below. During the Gulf War, it had proved difficult for army and marine units fighting together to share supplies back and forth because their logistics computer systems would not easily communicate. The goal of the new system was to combine the army and marine corps logistics systems into one system to enable units to share supplies under battlefield conditions.

The army and marines built separate as-is process models of their existing logistics systems that had 165 processes for the army system and 76 processes for the marines. Both process models were developed over a 3-month time period and cost several million dollars to build, even though they were not intended to be comprehensive.

I helped them develop a model for the new integrated battlefield logistics system that would be used by both services (i.e., the to-be model). The initial process model contained 1,500 processes and went down to level 6 DFDs in many places. It took 3,300 pages to print. They realized that this model was too large to be useful. The project leader decided that level 4 DFDs was as far as the model would go, with additional information contained in the process descriptions. This reduced the model to 375 processes (800 pages) and made it far more useful.

Alan Dennis

QUESTIONS:

1. What are the advantages and disadvantages to setting a limit for the maximum depth for a DFD?
2. Is a level 4 DFD an appropriate limit?

Validating the Data Flow Diagrams

Once you have created a set of DFDs, it is important to check them for quality. Figure 5-13 provides a quick checklist for identifying the most common errors. There are two fundamentally different types of problems that can occur in DFDs: *syntax errors* and *semantics errors*. “Syntax,” refers to the structure of the DFDs and whether the DFDs follow the rules of the DFD language. Syntax errors can be thought of as grammatical errors made by the analyst when he or she creates the DFD. “Semantics” refers to the meaning of the DFDs and whether they accurately describe the business process being modeled. Semantics errors can be thought of as misunderstandings by the analyst in collecting, analyzing, and reporting information about the system.

In general, syntax errors are easier to find and fix than are semantics errors, because there are clear rules that can be used to identify them (e.g., a process must have a name). Most CASE tools have syntax checkers that will detect errors within one page of a DFD in much the same way that word processors have spelling checkers and grammar checkers. Finding syntax errors that span several pages of a DFD (e.g., from a level 1 to a level 2 DFD) is slightly more challenging, particularly for consistent viewpoint, decomposition, and balance. Some CASE tools can detect balance errors, but that is about all. In most cases, analysts must carefully and painstakingly review every process, external entity, data flow, and data store on all DFDs by hand to make sure that they have a consistent viewpoint and that the decomposition and balance are appropriate.

Each data store is required to have at least one input and one output on some page of the DFD. In Figure 5-10, data store D1, New Vehicles, has only outputs and several data stores have only inputs. This situation is not necessarily an error. The analyst should check elsewhere in the DFDs to find where data is written to data

Syntax**Within DFD**

- | | |
|-----------------|--|
| Process | <ul style="list-style-type: none"> • Every process has a unique name that is an action-oriented verb phrase, a number, and a description. • Every process has at least one input data flow. • Every process has at least one output data flow. • Output data flows usually have different names than input data flows because the process changes the input into a different output in some way. |
| Data Flow | <ul style="list-style-type: none"> • There are between three and seven processes per DFD. • Every data flow has a unique name that is a noun, and a description. • Every data flow connects to at least one process. • Data flows only in one direction (no two-headed arrows). • A minimum number of data flow lines cross. |
| Data Store | <ul style="list-style-type: none"> • Every data store has a unique name that is a noun, and a description. • Every data store has at least one input data flow (which means to add new data or change existing data in the data store) on some page of the DFD. • Every data store has at least one output data flow (which means to read data from the data store) on some page of the DFD. |
| External Entity | <ul style="list-style-type: none"> • Every external entity has a unique name that is a noun, and a description. • Every external entity has at least one input or output data flow. |

Across DFDs

- | | |
|-----------------|--|
| Context diagram | <ul style="list-style-type: none"> • Every set of DFDs must have one context diagram. |
| Viewpoint | <ul style="list-style-type: none"> • There is a consistent viewpoint for the entire set of DFDs. |
| Decomposition | <ul style="list-style-type: none"> • Every process is wholly and completely described by the processes on its children DFDs. |
| Balance | <ul style="list-style-type: none"> • Every data flow, data store, and external entity on a higher level DFD is shown on the lower-level DFD that decomposes it. |

Semantics

- | | |
|----------------------------|--|
| Appropriate Representation | <ul style="list-style-type: none"> • User validation • Role-play processes |
| Consistent Decomposition | <ul style="list-style-type: none"> • Examine lowest-level DFDs |
| Consistent Terminology | <ul style="list-style-type: none"> • Examine names carefully |

FIGURE 5-13
Data Flow Diagram Quality Checklist

store D2 or read from the other data stores. All data stores should have at least one inflow and one outflow, but the flows may not be on the same diagram, so check other parts of the system. Another issue that arises is when the data store is utilized by other systems. In that case, data may be added to or used by a separate system. This is perfectly fine, but the analyst should investigate to verify that the required flows in and out of data stores exist somewhere.

In our experience, the most common syntax error that novice analysts make in creating DFDs is violating the law of conservation of data.³ The first part of the law states the following:

1. *Data at rest stays at rest until moved by a process.*

³ This law was developed by Prof. Dale Goodhue at the University of Georgia.

In other words, data cannot move without a process. Data cannot go to or come from a data store or an external entity without having a process to push it or pull it.

The second part of the law states the following:

2. *Processes cannot consume or create data.*

In other words, data only enters or leaves the system by way of the external entities. A process cannot destroy input data; all processes must have outputs. Drawing a process without an output is sometimes called a “black hole” error. Likewise, a process cannot create new data; it can transform data from one form to another, but it cannot produce output data without inputs. Drawing a process without an input is sometimes called a “miracle” error (because output data miraculously appear). There is one exception to the part of the law requiring inputs, but it is so rare that most analysts never encounter it.⁴ Figure 5-14 shows some common syntax errors.

Looking at Figure 5-14, we will discuss each error in turn. First, we can see data flow X drawn directly from Entity A to Entity B. Remember that a data flow must either originate or terminate at a process; therefore, a process is required. Second, we see data flow Z retrieved from Data Store P and sent to Entity B. Processes should exist to transform the data in some way, so we usually modify the data flow names to reflect the changes made in the process. Third, we see that Data Store P has outputs but has no inputs. This is not necessarily an error, but does deserve the analyst’s investigation. We should make sure that a process that adds data to Data Store P exists somewhere in the diagrams of the entire process model. Fourth, we can see that Process F receives data but has no outputs. This is considered a black hole since data is received but nothing is produced. Fifth, Process D is shown producing a data flow but has no inputs. This is termed a miracle process. Sixth, we see a two-headed arrow depicting Data Flow G between Process E and Process F. Data flows should not be drawn this way, but should flow only in one direction. Seventh, Data Store H receives Data Flow H as an input, but has no outputs. This issue may not be an error, but should be followed up by the analyst to ensure that the data that is stored in Data Store H is used some place in the process model; otherwise, there is no reason to store it. Finally, we see that a process should be involved between Entity A and Data Store H.

Generally speaking, semantics errors cause the most problems in system development. Semantics errors are much harder to find and fix because doing so requires a good understanding of the business process. And even then, what may be identified as an error may actually be a misunderstanding by the person reviewing the model. There are three useful checks to help ensure that models are semantically correct. (See Figure 5-13.)

The first check to ensure that the model is an appropriate representation is to ask the users to validate the model in a walk-through (i.e., the model is presented to the users, and they examine it for accuracy). A more powerful technique is for the users to role-play the process from the DFDs in the same way in which they role-played the use case. The users pretend to execute the process exactly as it is described in the DFDs. They start at the first process and attempt to perform it by using only the inputs specified and producing only the outputs specified. Then they move to the second process, and so on.

⁴ The exception is a temporal process that issues a trigger output based on an internal time clock. Whenever some predetermined period elapses, the process produces an output. The timekeeping process has no inputs because the clock is internal to the process.

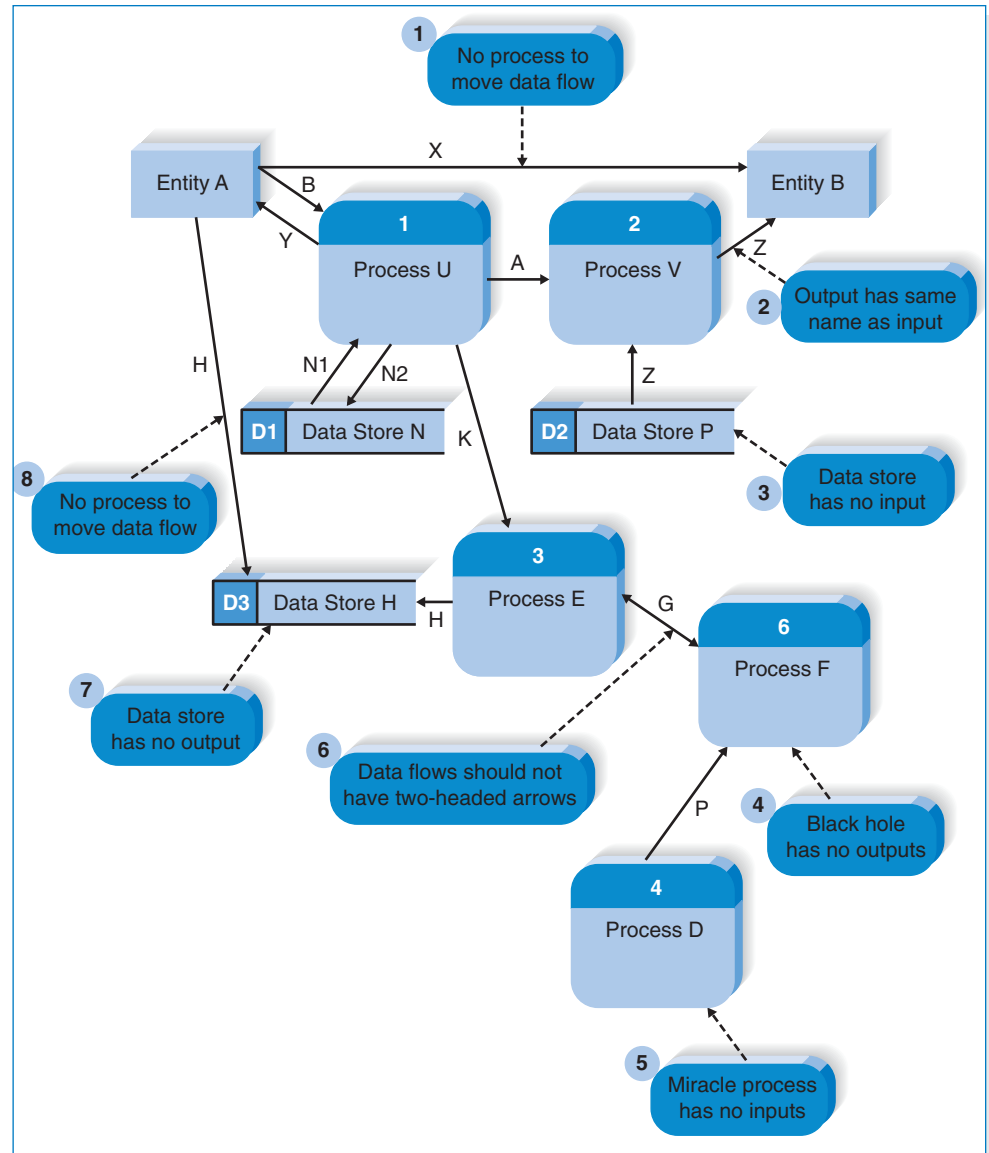


FIGURE 5-14
Some Common Errors

One of the most subtle forms of semantics error occurs when a process creates an output, but has insufficient inputs to create it. For example, in order to create water (H_2O), we need to have both hydrogen (H) and oxygen (O) present. The same is true of computer systems, in that the outputs of a process can be only combinations and transformations of its inputs. Suppose, for example, that we want to record an order; we need the customer name and mailing address and the quantities and prices for the items the customer is ordering. We need information from the customer data store (e.g., address) and information from the items data store (e.g., price). We cannot draw a process that produces an output order data flow without inputs from these two data stores. Role-playing with strict adherence to the inputs and outputs in a model is one of the best ways to catch this type of error.

A second semantics error check is to ensure consistent decomposition, which can be tested by examining the lowest-level processes in the DFDs. In most circumstances, all processes should be decomposed to the same level of detail—which is not the same as saying the same number of levels. For example, suppose that we were modeling the process of driving to work in the morning. One level of detail would be to say the following: (1) Enter car; (2) start car; (3) drive away. Another level of detail would be to say the following: (1) Unlock car; (2) sit in car; (3) buckle seat belt; and so on. Still another level would be to say the following: (1) Remove key from pocket; (2) insert key in door lock; (3) turn key; and so on. None of these is inherently better than another, but barring unusual circumstances, it is usually best to ensure that all processes at the very bottom of the model provide the same consistent level of detail.

Likewise, it is important to ensure that the terminology is consistent throughout the model. The same item may have different names in different parts of the organization, so one person's "sales order" may be another person's "customer order." Likewise, the same term may have different meanings; for example, "ship date" may mean one thing to the sales representative taking the order (e.g., promised date) and something else to the warehouse (e.g., the actual date shipped). Resolving these differences before the model is finalized is important in ensuring that everyone who reads the model or who uses the information system built from the model has a shared understanding.

APPLYING THE CONCEPTS AT TUNE SOURCE

Creating the Context Diagram

The project team began by creating the context diagram. They read through the summary area of the three major use cases in Figure 4-14 to find the major inputs and outputs.

The team noticed that the majority of data flow interactions are with the customers who are using the Web site to browse music selections and make download purchases. There will be an interaction with the payment clearinghouse entity that will handle payment verification and processing of purchases. Finally, although it is not obvious from the use cases, the marketing managers will be using sales information from the system to design and implement promotional campaigns. The team used the major inflows and outflows from the use cases and developed the context diagram shown in Figure 5-15.

Creating Data Flow Diagram Fragments

The next step was to create one DFD fragment for each use case. This was done by drawing the process in the middle of the page, making sure that the process number and name were appropriate, and connecting all the input and output data flows to it. Unlike the context diagram, the DFD fragment includes data flows to external entities and to internal data stores.

The completed DFD fragments are shown in Figure 5-16. Before looking at the figure, take a minute and draw them on your own. There are many good ways to draw these fragments. In fact, there are many "right" ways to create use cases and DFDs. Notice that on the DFD fragment for process 3 we have shown a dotted line inflow labeled "Time to determine promotions" into the process. Recall that we

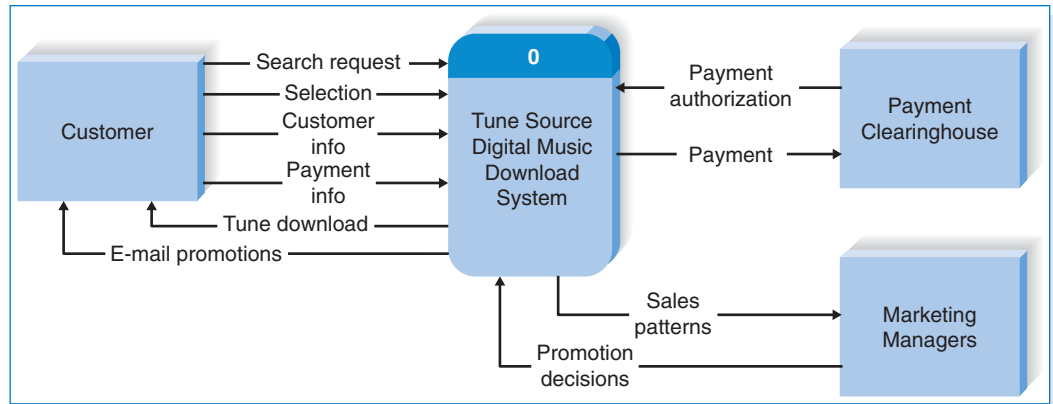


FIGURE 5-15
Tune Source Context Diagram

specified that the use case, Promote Tunes, was a temporal use case, triggered when it was time to update promotions and specials. The dotted line flow into process 3 in Figure 5-16 is sometimes referred to as a *control flow* and is commonly used to represent a time-based trigger for an event.

Creating the Level 0 Data Flow Diagram

The next step was to create the level 0 DFD by integrating the DFD fragments, which proved to be anticlimactic. The team simply took the DFD fragments and drew them together on one piece of paper. Although it sometimes is challenging to arrange all the DFD fragments on one piece of paper, it was primarily a mechanical exercise (Figure 5-17). Compare the level 0 diagram with the context diagram in Figure 5-15. Are the two DFDs balanced? Notice the additional detail contained in the level 0 diagram.

A careful review of the data stores in Figure 5-17 reveals that every one has both an inflow and an outflow, with one exception, D1:Available Tunes. D1:Available Tunes is read by two processes, but is not written to by any process shown. This violation of DFD syntax needs to be investigated by the team because it may be a serious oversight. As we will explain later, in this situation we need to create a process specifically for adding, modifying, and deleting data in D1:Available Tunes. “Administrative” processes such as this are often overlooked, as we initially focus on business requirements only, but will need to be added before the system is complete.

Creating Level 1 Data Flow Diagrams (and Below)

The next step was to create the level 1 DFDs for those processes that could benefit from them. The analysts started with the first use case (search and browse tunes) and started to draw a DFD for the individual steps it contained. The steps in the use case were straightforward, but as is common, the team had to choose names and numbers for the processes and to add input data flows from data stores not present

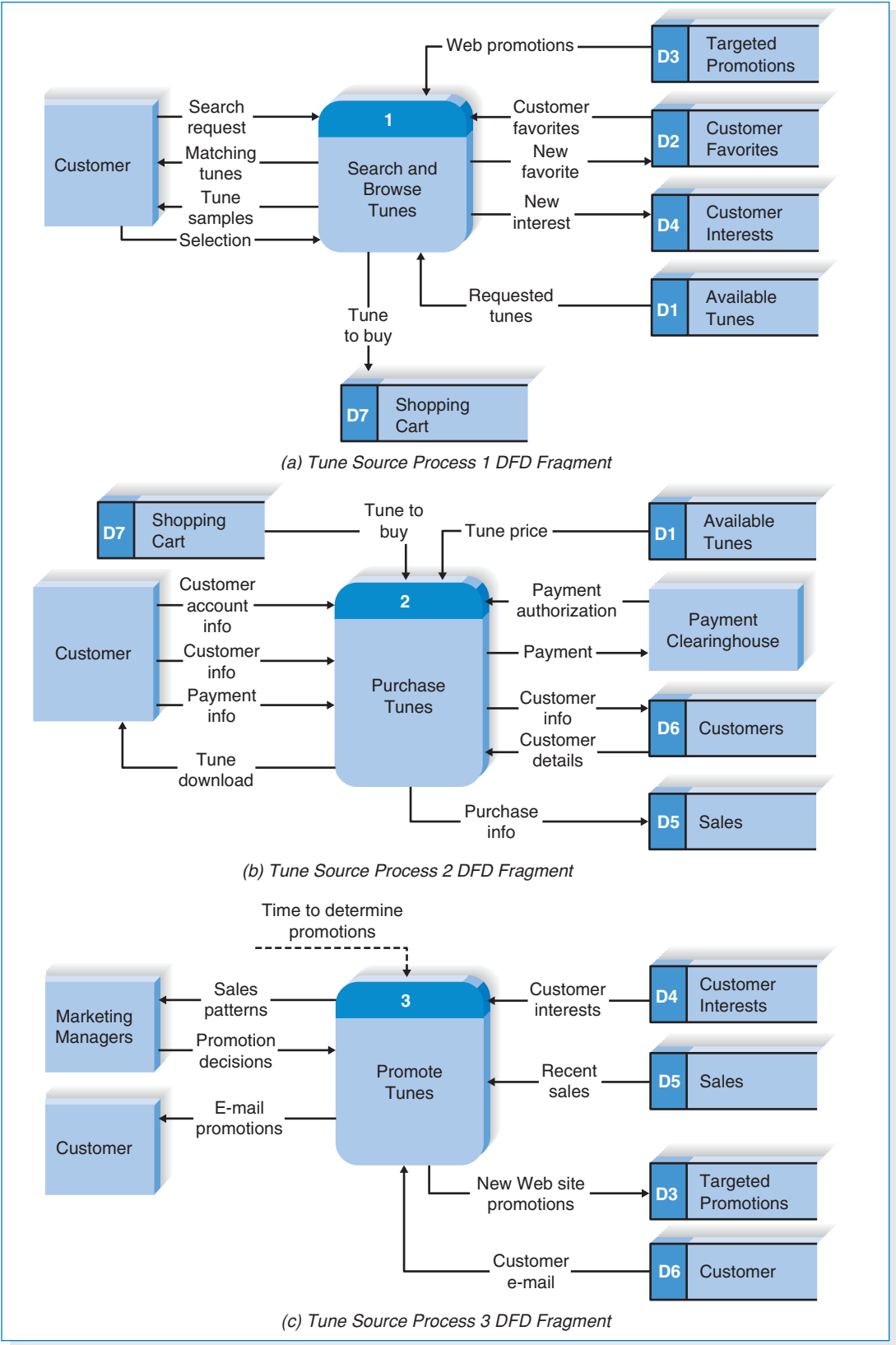


FIGURE 5-16
Tune Source DFD
Fragments

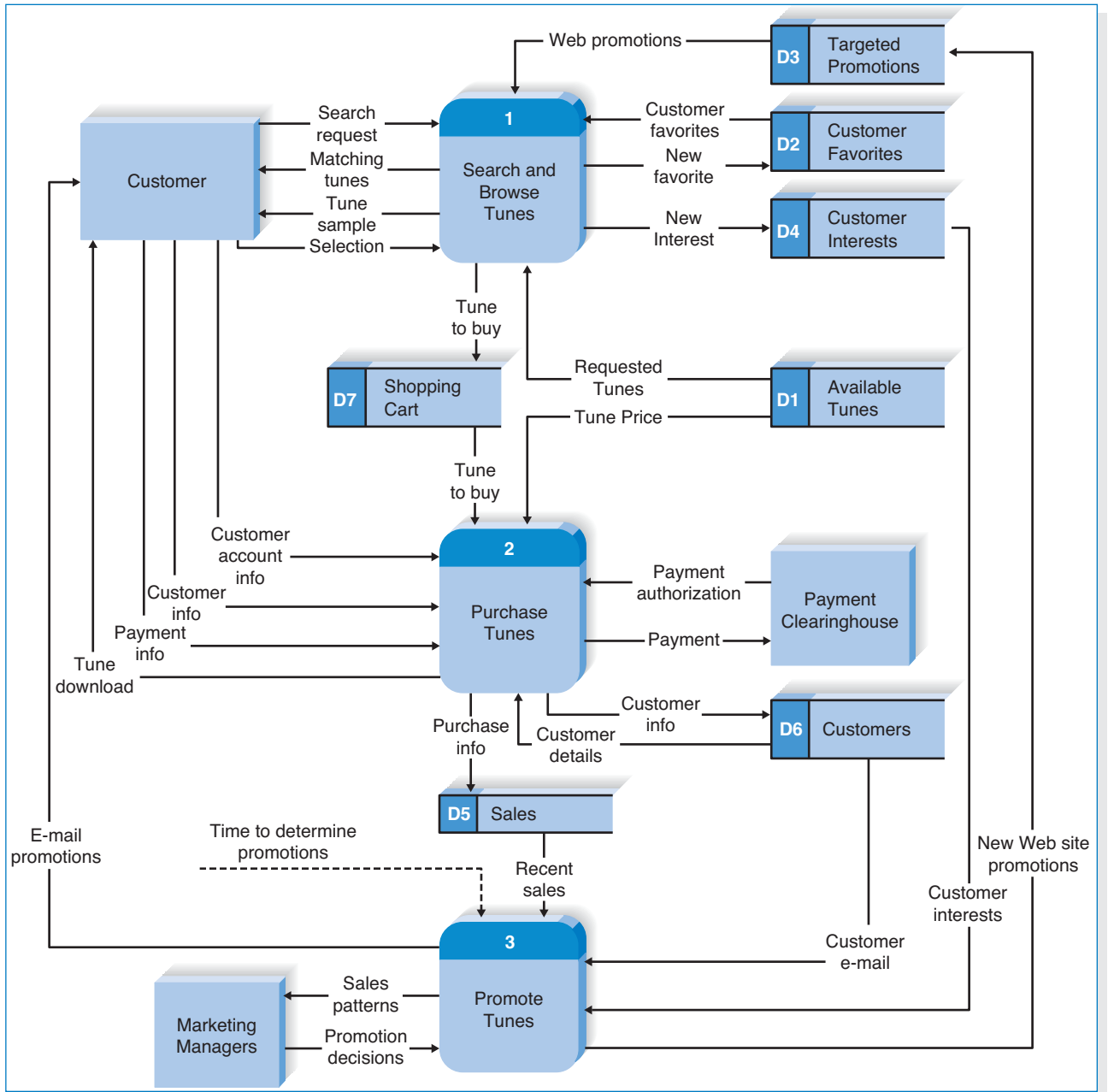


FIGURE 5-17
Tune Source Level 0 DFD

in the use case. The team also discovered the omission of a data flow, customer interests, from the use case. See Figure 5-18.

The team also developed level 1 diagrams for the other processes, using the major steps outlined in their respective use cases. Some adjustments were made from the steps as shown in the use cases, but the team followed the steps fairly closely. See Figures 5-19 and 5-20, and compare them with their use cases shown in Figure 4-14.

YOUR
TURN

5-1 CAMPUS HOUSING

Draw a context diagram, a level 0 DFD, and a set of level 1 DFDs (where needed) for the campus housing use cases that you developed for the Your Turn 4-1 box in Chapter 4.

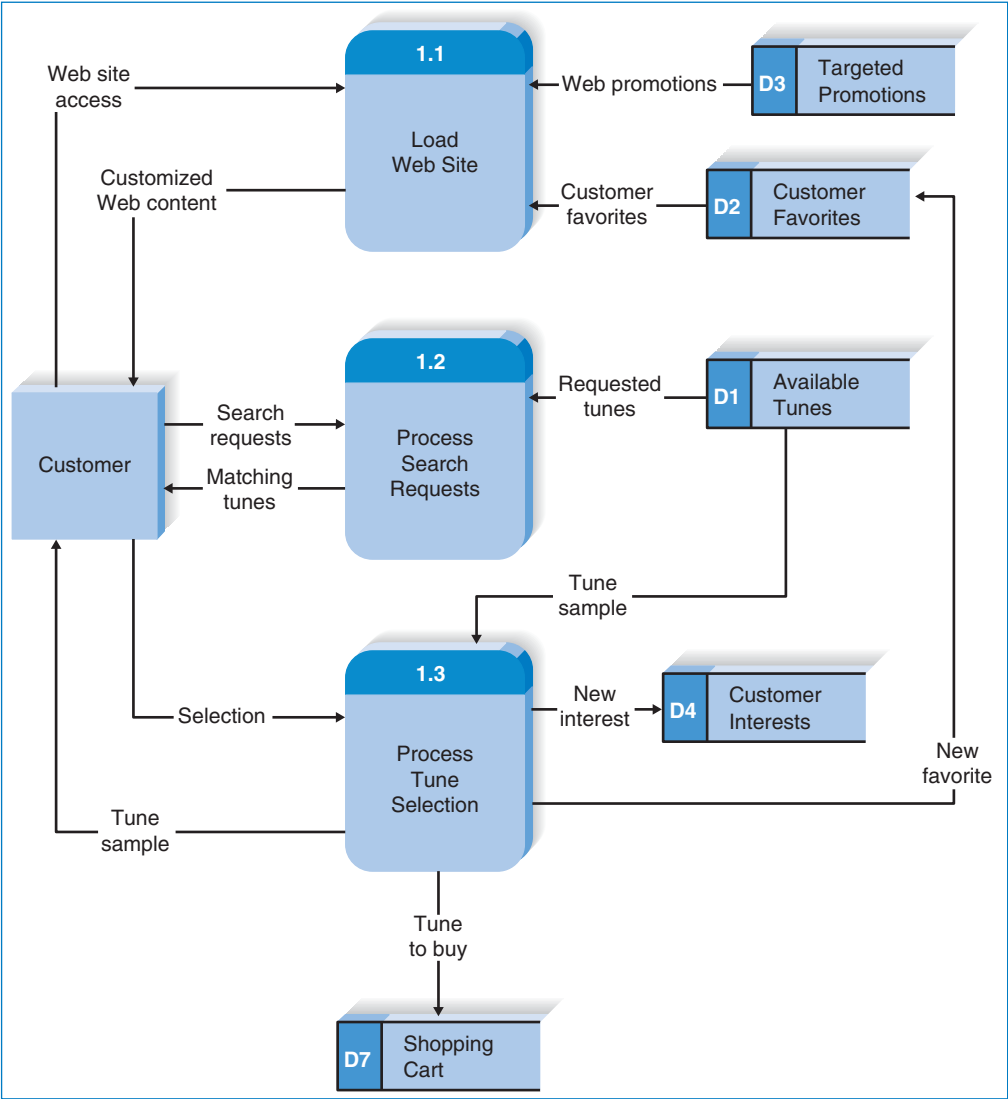


FIGURE 5-18
Level 1 DFD for Tune Source Process 1: Search and Browse Tunes

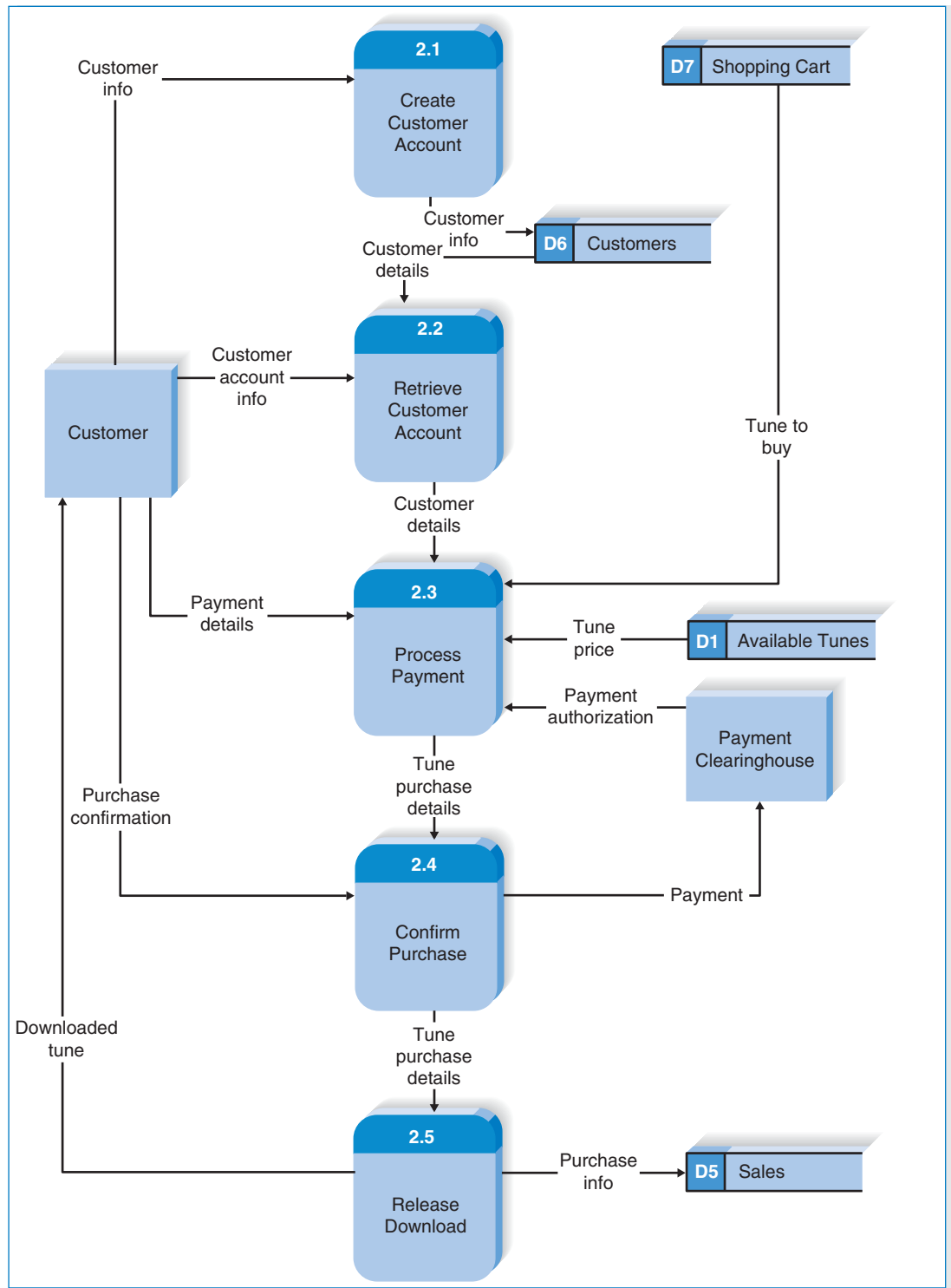
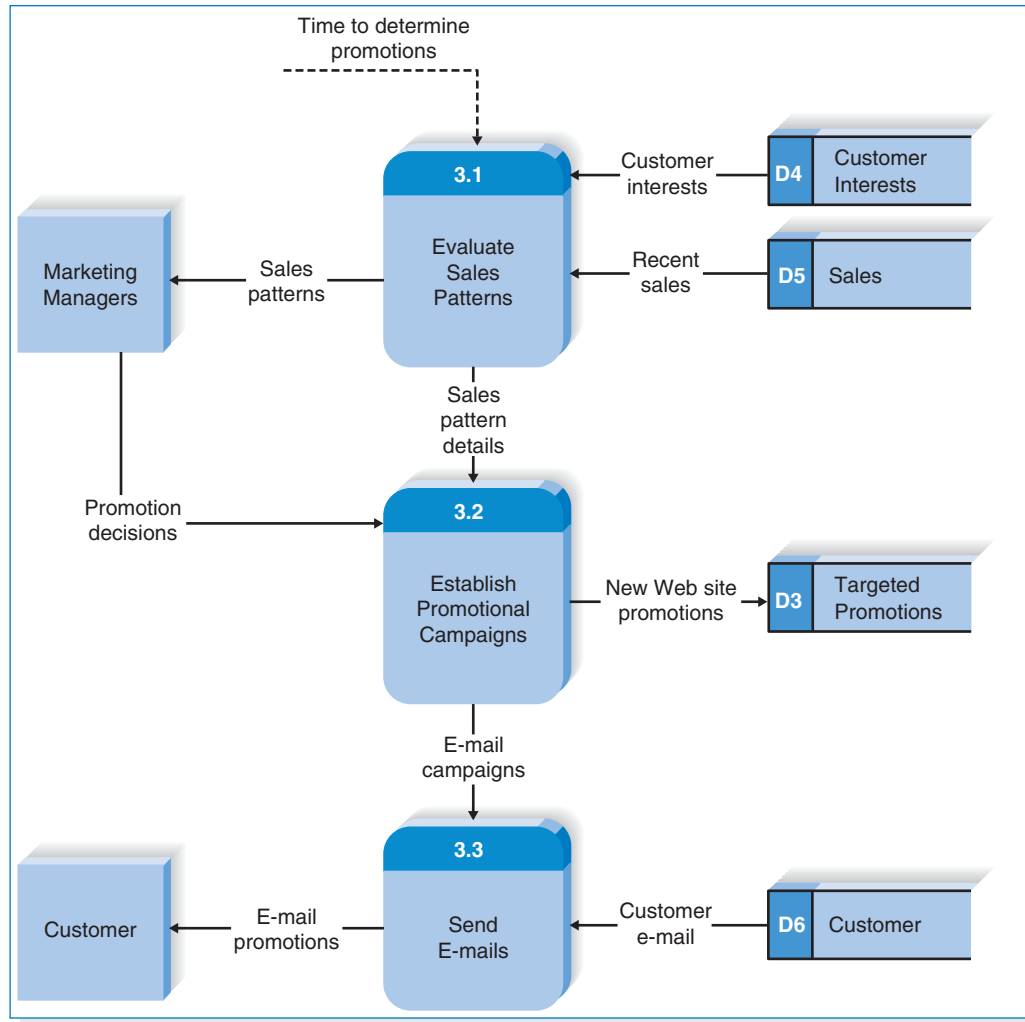


FIGURE 5-19
Level 1 DFD for Tune Source Process 2: Purchase Tunes

**FIGURE 5-20**

Level 1 DFD for Tune Source Process 3: Promote Tunes

As we specified in Figure 5-2, every data store should have one or more input data flows and one or more output data flows. A careful look at Figure 5-18, however, reveals that D1: Available Tunes has output data flows, but no input data flows. This data store is the main repository for the digital music library, so, clearly, it has a central role in our system. The team will need to be sure to create an administrative process for maintaining this data store: adding new information to it, modifying its existing contents, and removing information from it. These administrative tasks are sometimes omitted from the business-oriented tasks listed in the use cases, so it is up to the team to ensure that processes are included to add, modify, and delete the contents of data stores somewhere in the system. Simply checking the DFD syntax (all data stores must have at least one input data flow and at least one output data flow) helped discover this omission in the process model.

Although it would have been possible to decompose several of the processes on the level 1 DFDs into more detail on a level 2 diagram, the project team decided that that step was not necessary. Instead, they made sure that the process descriptions they created in the CASE repository for each of the processes was very detailed. This detail would be critical in developing the data models and designing the user interface and programs during the design phase.

Validating the Data Flow Diagrams

The final set of DFDs was validated by the project team and then by Carly and her marketing team in a final JAD meeting. There was general approval of the customer-facing processes (process 1 and process 2) and considerable discussion of the specific information that would be required to help the marketing team make its promotional campaign decisions (process 3). This information was recorded in the CASE repository so that it could be recalled during the development of the system's data model, the subject of our next chapter.

SUMMARY

Data Flow Diagram Syntax

Four symbols are used on data flow diagrams (processes, data flows, data stores, and external entities). A process is an activity that does something. Each process has a name (a verb phrase), a description, and a number that shows where it is in relation to other processes and to its children processes. Every process must have at least one output and usually has at least one input. A data flow is a piece of data or an object and has a name (a noun) and a description and either starts or ends at a process (or both). A data store is a manual or computer file, and it has a number, a name (a noun), and at least one input data flow and one output data flow (unless the data store is created by a process external to the data flow diagram [DFD]). An external entity is a person, organization, or system outside the scope of the system and has a name (a noun) and a description. Every set of DFDs starts with a context diagram and a level 0 DFD and has numerous level 1 DFDs, level 2 DFDs, and so on. Every element on the higher-level DFDs (i.e., data flows, data stores, and external entities) must appear on lower-level DFDs, or else they are not balanced.

Creating Data Flow Diagrams

The DFDs are created from the use cases. First, the team builds the context diagram that shows all the external entities and the data flows into and out of the system from them. Second, the team creates DFD fragments for each use case that show how the use case exchanges data flows with the external entities and data stores. Third, these DFD fragments are organized into a level 0 DFD. Fourth, the team develops level 1 DFDs on the basis of the steps within each use case to better explain how they operate. Fifth, the team validates the set of DFDs to make sure that they are complete and correct and contain no syntax or semantics errors. Analysts seldom create DFDs perfectly the first time, so iteration is important in ensuring that both single-page and multipage DFDs are clear and easy to read.

KEY TERMS

Action statement	Decomposition	Parent
Balancing	DFD fragment	Physical model
Bundle	External entity	Process model
Case statement	For statement	Process
Children	If statement	Semantics error
Context diagram	Iteration	Structured English
Data flow	Layout	Syntax error
Data flow diagram (DFD)	Level 0 DFD	Viewpoint
Data store	Level 1 DFD	While statement
Decision table	Level 2 DFD	
Decision tree	Logical process model	

QUESTIONS

1. What is a process model? What is a data flow diagram? Are the two related? If so, how?
2. Distinguish between logical process models and physical process models.
3. Define what is meant by a *process* in a process model. How should a process be named? What information about a process should be stored in the CASE repository?
4. Define what is meant by a *data flow* in a process model. How should a data flow be named? What information about a data flow should be stored in the CASE repository?
5. Define what is meant by a *data store* in a process model. How should a data store be named? What information about a data store should be stored in the CASE repository?
6. Define what is meant by an *external entity* in a process model. How should an external entity be named? What information about an external entity should be stored in the CASE repository?
7. Why is a process model typically composed of a set of DFDs? What is meant by decomposition of a business process?
8. Explain the relationship between a DFD context diagram and the DFD level 0 diagram.
9. Explain the relationship between a DFD level 0 diagram and DFD level 1 diagram(s).
10. Discuss how the analyst knows how to stop decomposing the process model into more and more levels of detail.
11. Suppose that a process on a DFD is numbered 4.3.2. What level diagram contains this process? What is this process's parent process?
12. Explain the use of structured English in process descriptions.
13. Why would one use a decision tree and/or decision table in a process description?
14. Explain the process of balancing a set of DFDs.
15. How are mutually exclusive data flows (i.e., alternative paths through a process) depicted in DFDs?
16. Discuss several ways to verify the correctness of a process model.
17. Identify three typical syntax errors commonly found in DFDs.
18. What is meant by a DFD semantic error? Provide an example.
19. Creating use cases when working with users is a recent development in systems analysis practice. Why is the trend today to employ use cases in user interviews or JAD sessions?
20. How can you make a DFD easier to understand? (Think first about how to make one difficult to understand.)
21. Suppose that your goal is to create a set of DFDs. How would you begin an interview with a knowledgeable user? How would you begin a JAD session?

EXERCISES

- A. Draw a level 0 data flow diagram (DFD) for the process of buying glasses in Exercise A, Chapter 4.
- B. Draw a level 0 data flow diagram (DFD) for the dentist office system in Exercise B, Chapter 4.
- C. Draw a level 0 data flow diagram (DFD) for the university system in Exercise D, Chapter 4.
- D. Draw a level 0 data flow diagram (DFD) for the real estate system in Exercise E, Chapter 4.
- E. Draw a level 0 data flow diagram (DFD) for the video store system in Exercise F, Chapter 4.
- F. Draw a level 0 data flow diagram (DFD) for the health club system in Exercise G, Chapter 4.
- G. Draw a level 0 data flow diagram (DFD) for the Picnics R Us system in Exercise H, Chapter 4.
- H. Draw a level 0 data flow diagram (DFD) for the Of-the-Month Club system in Exercise I, Chapter 4.
- I. Draw a level 0 data flow diagram (DFD) for the university library system in Exercise J, Chapter 4.

MINICASES

1. The Hatcher Company is in the process of developing a new inventory management system. One of the event handling processes in that system is Receive Supplier Shipments. The (inexperienced) systems analyst on the project has spent time in the warehouse observing this process and developed the following list of activities that are performed: getting the new order in the warehouse, unpacking the boxes, making sure that all the ordered items were actually received, putting the items on the correct shelves, dealing with the supplier to reconcile any discrepancies, adjusting the inventory quantities on hand, and passing along the shipment information to the accounts payable office. He also created the accompanying Level 1 data flow diagram for this process. Unfortunately, this DFD has numerous syntax and semantic errors. Identify the errors. Redraw the DFD to more correctly represent the Receive Supplier Shipments process.
2. Professional and Scientific Staff Management (PSSM) is a unique type of temporary staffing agency. Many organizations today hire highly skilled technical employees on a short-term, temporary basis to assist with special projects or to provide a needed technical skill. PSSM negotiates contracts with its client companies in which it agrees to provide temporary staff in specific job categories for a specified cost. For example, PSSM has a contract with an oil and gas exploration company, in which it agrees to supply geologists with at least a master's degree for \$5000 per week. PSSM has contracts with a wide range of companies and can place almost any type of professional or scientific staff members, from computer programmers to geologists to astrophysicists.

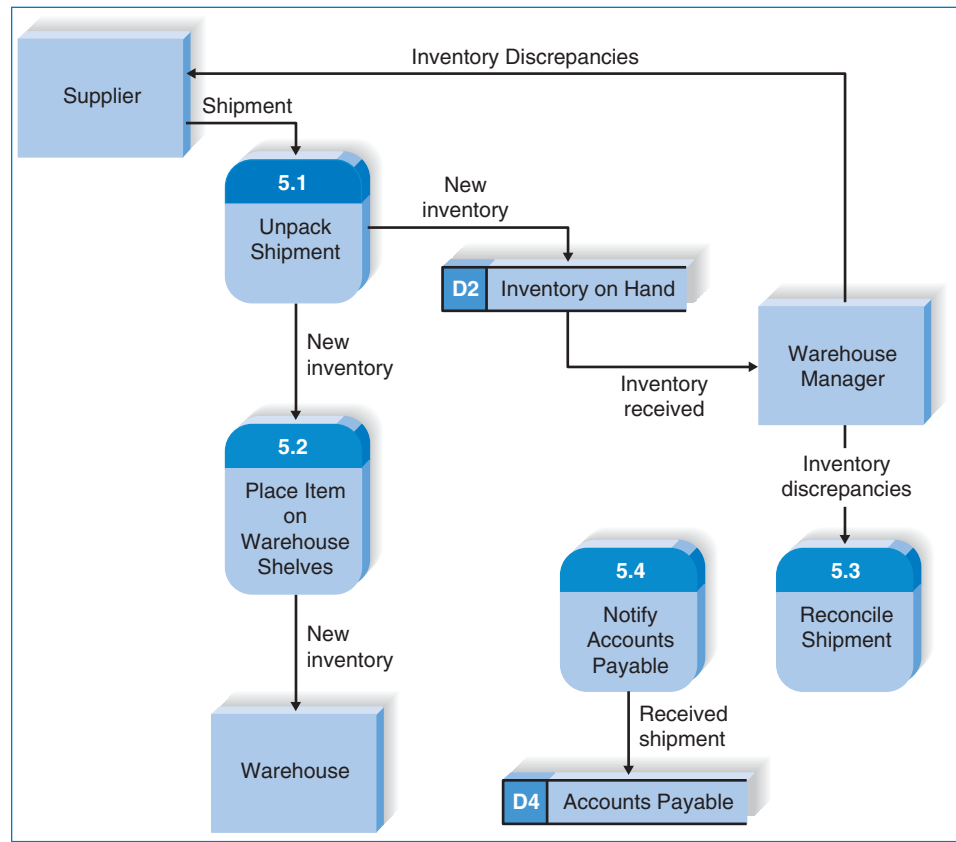
When a PSSM client company determines that it will need a temporary professional or scientific employee, it issues a staffing request against the contract it had previ-

ously negotiated with PSSM. When a staffing request is received by PSSM's contract manager, the contract number referenced on the staffing request is entered into the contract database. Using information from the database, the contract manager reviews the terms and conditions of the contract and determines whether the staffing request is valid. The staffing request is valid if the contract has not expired, the type of professional or scientific employee requested is listed on the original contract, and the requested fee falls within the negotiated fee range. If the staffing request is not valid, the contract manager sends the staffing request back to the client with a letter stating why the staffing request cannot be filed, and a copy of the letter is filed. If the staffing request is valid, the contract manager enters the staffing request into the staffing request database, as an outstanding staffing request. The staffing request is then sent to the PSSM placement department.

In the placement department, the type of staff member, experience, and qualifications requested on the staffing request are checked against the database of available professional and scientific staff. If a qualified individual is found, he or she is marked "reserved" in the staff database. If a qualified individual cannot be found in the database or is not immediately available, the placement department creates a memo that explains the inability to meet the staffing request and attaches it to the staffing request. All staffing requests are then sent to the arrangements department.

In the arrangement department, the prospective temporary employee is contacted and asked to agree to the placement. After the placement details have been worked out and agreed to, the staff member is marked "placed" in the staff database. A copy of the staffing

Hatcher Company Inventory Management System Level 1 DFD



request and a bill for the placement fee is sent to the client. Finally, the staffing request, the “unable to fill” memo (if any), and a copy of the placement fee bill is sent to the contract manager. If the staffing request was filled, the contract manager closes the open staffing request in the staffing request database. If the staffing request could not be filled, the client is notified. The staffing request, placement fee bill, and “unable to fill” memo are then filed in the contract office.

- Develop a use case for each of the major processes just described.
- Create the context diagram for the system just described.
- Create the DFD fragments for each of the four use cases outlined in part a, and then combine them into the level 0 DFD.
- Create a level 1 DFD for the most complicated use case.

This page is intentionally left blank

PLANNING

ANALYSIS

- ☒ Use requirements elicitation techniques (interview, JAD session, questionnaire, document analysis, and observation).
- ☒ Apply requirements analysis strategies as needed to discover underlying requirements.
- ☒ Develop the requirements definition.
- ☒ Develop use cases.
- ☒ Develop data flow diagrams.
- ☐ Develop entity relationship model
- ☐ Normalize entity relationship model

T A S K C H E C K L I S T

PLANNING

ANALYSIS

DESIGN

CHAPTER 6

DATA

MODELING

A data model describes the data that flow through the business processes in an organization. During the analysis phase, the data model presents the logical organization of data without indicating how the data are stored, created, or manipulated so that analysts can focus on the business without being distracted by technical details. Later, during the design phase, the data model is changed to reflect exactly how the data will be stored in databases and files. This chapter describes entity relationship diagramming, one of the most common data modeling techniques used in industry.

OBJECTIVES

- Explain the rules and style guidelines for creating entity relationship diagrams.
- Create an entity relationship diagram.
- Describe the use of a data dictionary and metadata.
- Explain how to balance entity relationship diagrams and data flow diagrams.
- Describe the process of normalization.

CHAPTER OUTLINE

Introduction

The Entity Relationship Diagram

*Reading an Entity Relationship
Diagram*

*Elements of an Entity Relationship
Diagram*

The Data Dictionary and Metadata

Creating an Entity Relationship Diagram

*Building Entity Relationship
Diagrams*

Advanced Syntax

Applying the Concepts at Tune Source

Validating an ERD

Design Guidelines

Normalization

Balancing Entity Relationship

Diagrams with Data Flow Diagrams

Summary

Appendix 6A: Normalizing the Data
Model

INTRODUCTION

During the analysis phase, analysts create process models to represent how the business system will operate. At the same time, analysts need to understand the information that is used and created by the business system (e.g., customer information, order information). In this chapter, we discuss how the data that flow through the processes are organized and presented.

A *data model* is a formal way of representing the data that are used and created by a business system; it illustrates people, places, or things about which information is captured and how they are related to each other. The data model is drawn by an iterative process in which the model becomes more detailed and less conceptual over time. During analysis, analysts draw a logical data model, which shows the logical organization of data without indicating how data are stored, created, or manipulated. Because this model is free of any implementation or technical details, the analysts can focus more easily on matching the diagram to the real business requirements of the system.

In the design phase, analysts draw a *physical data model* to reflect how the data will physically be stored in databases and files. At this point, the analysts investigate ways to store the data efficiently and to make the data easy to retrieve. The physical data model and performance tuning are discussed in Chapter 11.

Project teams usually use CASE tools to draw data models. Some of the CASE tools are data modeling packages, such as *ERwin* by Platinum Technology, that help analysts create and maintain logical and physical data models; they have a wide array of capabilities to aid modelers, and they can automatically generate many different kinds of databases from the models that are created. Other CASE tools (e.g., Oracle Designer) come bundled with database management systems (e.g., Oracle), and they are particularly good for modeling databases that will be built in their companion database products. A final option is to use a full-service CASE tool, such as Visible Analyst Workbench, in which data modeling is one of many capabilities, and the tool can be used with many different databases. A benefit of the full-service CASE tool is that it integrates the data model information with other relevant parts of the project.

In this chapter, we focus on creating a logical data model. Although there are several ways to model data, we will present one of the most commonly used techniques: entity relationship diagramming, a graphic drawing technique developed by Peter Chen¹ that shows all the data components of a business system. We will first describe how to create an entity relationship diagram (ERD) and discuss some style guidelines. Then, we will present a technique called normalization that helps analysts validate the data models that they draw. The chapter ends with a discussion of how data models balance, or interrelate, with the process models that you learned about in Chapter 5.

THE ENTITY RELATIONSHIP DIAGRAM

An *entity relationship diagram (ERD)* is a picture which shows the information that is created, stored, and used by a business system. An analyst can read an ERD to discover the individual pieces of information in a system and how they are organized

¹ P. Chen, “The Entity-Relationship Model—Toward a Unified View of Data,” *ACM Transactions on Database Systems*, 1976, 1:9–36.

and related to each other. On an ERD, similar kinds of information are listed together and placed inside boxes called entities. Lines are drawn between entities to represent relationships among the data, and special symbols are added to the diagram to communicate high-level business rules that need to be supported by the system. The ERD implies no order, although entities that are related to each other are usually placed close together.

For example, consider the Lawn Chemical Request system that was described in Chapter 5. Although this system is just a small part of the information system for a lawn care business, we will use it for our discussion on how to read an entity relationship diagram. First, go back and look at the sample DFD for the chemical request process in Figure 5-1. Although we understand how the system works from studying the data flow diagram, we have very little detailed understanding of the information itself that flows through the system. What exactly is a “new chemical request”? What pieces of data are captured in a “chemical pick-up authorization”?

Reading an Entity Relationship Diagram

The analyst can answer these questions and more by using an entity relationship diagram. We have included a partial ERD for the chemical request scenario in Figure 6-1. First, we have organized the data into three main categories: Lawn Chemical Applicator, Chemical Request, and Chemical. The Lawn Chemical Applicator data describe the employees who apply the lawn chemicals. The Chemical Request data capture information about every chemical request event, and the chemical data describe the chemicals used for lawn care.

We can also see the specific facts that describe each of the three categories. For example, a chemical is described by its ID number, name, description, approval status, and unit of measure. We can also see what can be used to uniquely identify a chemical, a chemical request, and an LCA, by looking for the asterisks next to the data elements. A unique ID has been created to identify every LCA and every chemical. A chemical request is uniquely identified by a combination of the LCA ID, the chemical ID, and the request date.

The lines connecting the three categories of information communicate the relationships that the categories share. By reading the relationship lines, the analyst understands that an LCA makes chemicals requests and chemical requests involve chemicals.

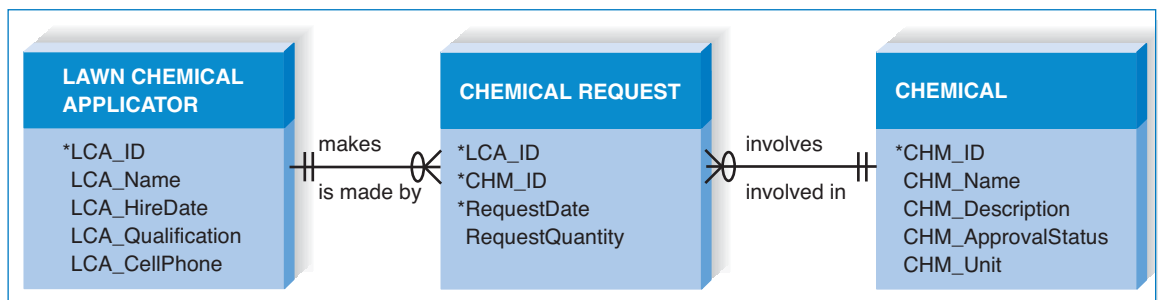


FIGURE 6-1
Chemical Request ERD

The ERD also communicates high-level business rules. Business rules are constraints or guidelines that are followed during the operation of the system; they are rules such as “A payment can be cash, check, debit card, credit card, coupon(s), or food stamps,” “A sale is paid for by one or more payments,” or “A customer may place many orders.” Over the course of a workday, people are constantly applying business rules to do their jobs, and they know the rules through training or knowing where to look them up. If a situation arises where the rules are not known, workers may have to refer to a policy guide or written procedure to determine the proper business rules.

On a data model, business rules are communicated by the kinds of relationships that the entities share. From the ERD, for example, we know from the “crow’s foot” placed on the line closest to the Chemical Request that an LCA may make many Chemical Requests. We can see by the two bars placed on the line closest to the LCA that a Chemical Request is made by exactly one LCA. Ultimately, the new system should support the business rules we just described, and it should ensure that users don’t violate the rules when performing the processes of the system. Therefore, in our example, the system should not permit a chemical request to be made that does not involve an LCA. Similarly, the system should not allow a chemical request to involve more than one LCA.

Now that you’ve seen an ERD, let’s step back and learn the ERD basics. In the following sections, we will first describe the syntax of the ERD, using the diagram in Figure 6-1. Then we will teach you how to create an ERD by using an example from Tune Source.

Elements of an Entity Relationship Diagram

There are three basic elements in the data modeling language (entities, attributes, and relationships), each of which is represented by a different graphic symbol. There are many different sets of symbols that can be used on an ERD. No one set of symbols dominates industry use, and none is necessarily better than another. We will use crow’s foot in this book. Figure 6-2 summarizes the three basic elements of ERDs and the symbols we will use.

Entity The *entity* is the basic building block for a data model. It is a person, place, event, or thing about which data is collected—for example, an employee, an order, or a product. An entity is depicted by a rectangle, and it is described by a singular noun spelled in capital letters. All entities have a name, a short description that explains what they are, and an *identifier* that is the way to locate information in the entity (which is discussed later). In Figure 6-1, the entities are Lawn Chemical Applicator, Chemical Request, and Chemical.

Entities represent something for which there exist multiple *instances*, or occurrences. For example, John Smith and Susan Jones could be instances of the customer entity (Figure 6-3). We would expect the customer entity to stand for all of the people with whom we have done business, and each of them would be an instance in the customer entity. If there is just one instance, or occurrence, of a person, place, event, or thing, then it should not be included as an entity in the data model. For example, think a little more broadly about the lawn care business’s information system. Figure 6-1 focuses on just a small part of that information system. We assumed that the company consisted of more than one Lawn Chemical Applicator, because we included an LCA entity to capture specific facts about each. If the company was owned and operated by a single person, however, there would

	IDEF1X	Chen	Crow's Foot
An ENTITY ✓ is a person, place, or thing. ✓ has a singular name spelled in all capital letters. ✓ has an identifier. ✓ should contain more than one instance of data.	ENTITY-NAME <div>Identifier</div>	ENTITY-NAME <div></div>	ENTITY-NAME <div>*Identifier</div>
An ATTRIBUTE ✓ is a property of an entity. ✓ should be used by at least one business process. ✓ is broken down to its most useful level of detail.	ENTITY-NAME <div>Attribute-name Attribute-name Attribute-name</div>	<div>Attribute-name</div>	ENTITY-NAME <div>Attribute-name Attribute-name Attribute-name</div>
A RELATIONSHIP ✓ shows the association between two entities. ✓ has a parent entity and a child entity. ✓ is described with a verb phrase. ✓ has cardinality (1 : 1, 1 : N, or M : N). ✓ has modality (null, not null). ✓ is dependent or independent.	Relationship-name	<div>Relationship-name</div>	Relationship-name

FIGURE 6-2
Data Modeling Symbol Sets

be no need to set up an LCA entity in the overall data model. There is no need to capture data in the system about something having just a single instance.

Attribute An *attribute* is some type of information that is captured about an entity. For example, last name, home address, and e-mail address are all attributes of a customer. It is easy to come up with hundreds of attributes for an entity (e.g., a customer has an eye color, a favorite hobby, a religious affiliation), but only those that actually will be used by a business process should be included in the model.

Attributes are nouns that are listed within an entity. Usually, some form of the entity name is appended to the beginning of each attribute to make it clear as

Entity	Example Instances
<div>Customer</div>	John Smith Susan Jones Peter Todd Dale Turner Pat Turner

FIGURE 6-3
Entities and Instances

to what entity it belongs (e.g., CUS_lastname, CUS_address). Without doing this, you can get confused by multiple entities that have the same attributes—for example, a customer and an employee both can have an attribute called “last-name.” CUS_lastname and EMP_lastname are much clearer ways to name attributes on the data model.

One or more attributes can serve as the identifier—the attribute(s) that can uniquely identify one instance of an entity—and the attributes that serve as the identifier are noted by an asterisk next to the attribute name. If there are no customers with the same last name, then last name can be used as the identifier of the customer entity. In this case, if we need to locate John Brown, the name Brown would be sufficient to identify the one instance of the Brown last name.

Suppose that we add a customer named Sarah Brown. Now we have a problem: Using the name Brown would not uniquely lead to one instance—it would lead to two (i.e., John Brown and Sarah Brown). You have three choices at this point, and all are acceptable solutions. First, you can use a combination of multiple fields to serve as the identifier (last name and first name). This is called a *concatenated identifier* because several fields are combined, or concatenated, to uniquely identify an instance. Second, you can find a field that is unique for each instance, like the customer ID number. Third, you can wait to assign an identifier (like a randomly generated number that the system will create) until the design phase of the SDLC (Figure 6-4). Many data modelers don’t believe that randomly generated identifiers belong on a logical data model, because they do not logically exist in the business process.

Relationship *Relationships* are associations between entities, and they are shown by lines that connect the entities together. Every relationship has a *parent entity* and a *child entity*, the parent being the first entity in the relationship, and the child being the second.

Relationships should be clearly labeled with active verbs so that the connections between entities can be understood. If one verb is given to each relationship, it is read in two directions. For example, we could write the verb *makes* alongside the relationship for the *LCA* and *Chemical Request* entities, and this would be read as “an LCA makes a chemical request” and “a chemical request is made by an LCA.” In Figure 6.1, we have included words for both directions of the relationship line; the top words are read from parent to child, and the bottom words are read from child to parent. Notice that the *LCA* entity is the parent entity in the *LCA-Chemical Request* relationship. In addition, some CASE tools require that every relationship name be unique on the ERD, so we select unique descriptive verbs for each relationship.

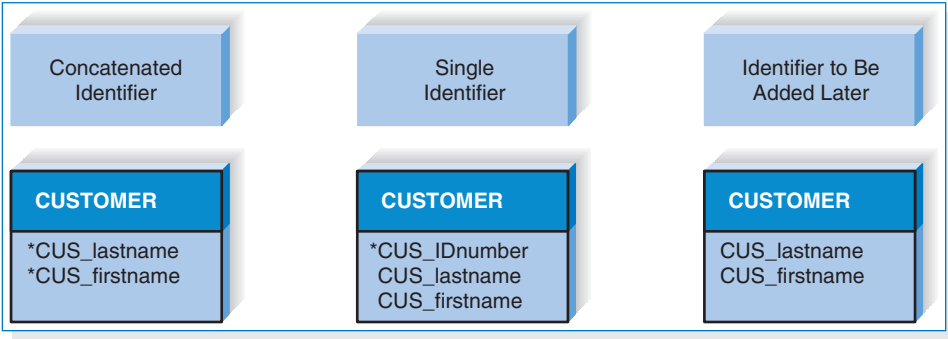


FIGURE 6-4
Choices for Identifiers

Cardinality Relationships have two properties. First, a relationship has cardinality, which is the ratio of parent instances to child instances. To determine the cardinality for a relationship, we ask ourselves: “How many instances of one entity are associated with an instance of the other?” (Remember that an instance is one occurrence of an entity, such as LCA John Brown or Chemical Orthene™.) For example, an LCA makes how many chemical requests? The cardinality for binary relationships (i.e., relationships between two entities) is 1:1, 1:N, or M:N, and we will discuss each in turn.

The 1:1 (read as “one to one”) relationship means that one instance of the parent entity is associated with one instance of the child entity. There are no examples of 1:1 relationships in Figure 6-1. So, imagine for a moment that, as a reward, a company assigns a specific reserved parking place to every employee who is honored as an “employee of the month.” One reserved parking place is assigned to each honored employee, and each honored employee is assigned one reserved parking place. If we were to draw these two entities, we would place a bar next to the Employee entity and a bar next to the Reserved Parking Place entity. The cardinality is clearly 1:1 in this case, because each honored employee is assigned exactly one reserved parking place, and a reserved parking place is assigned to exactly one employee.

More often, relationships are 1:N (read as “one to many”). In this kind of relationship, a single instance of a parent entity is associated with many instances of a child entity; however, the child entity instance is related to only one instance of the parent. For example, an LCA (parent entity) can make many Chemical Requests (child entity), but a particular Chemical Request is made by only one LCA, suggesting a 1:N relationship between LCA and Chemical Request. A character resembling a crow’s foot is placed closest to the Chemical Request entity to show the “many” end of the relationship. The parent entity is always on the “1” side of the relationship; hence, a bar is placed next to the LCA entity. Can you identify other 1:N relationships in Figure 6-1? Identify the parent and child entities for each relationship.

A third kind of relationship is the M:N (read as “many to many”) relationship. In this case, many instances of a parent entity can relate to many instances of a child entity. There are no M:N relationships shown in Figure 6-1, but take a look at Figure 6-5. This figure shows an early draft version of the Chemical Request ERD. In this version, an M:N relationship does exist between LCA and Chemical. As we can see, one LCA (parent entity) can request many Chemicals (e.g., Orthene™, Roundup™, and 2, 4-D.), and a Chemical (child entity) can be requested by many LCAs. M:N relationships are depicted on an ERD by having crow’s feet at both ends of the relationship line. As we will learn later, there are advantages to eliminating M:N relationships from an ERD, so that is why it was removed from Figure 6-1 by creating the Chemical Request entity between LCA and Chemical. The process of “resolving” an M:N relationship will be explained later in the chapter.

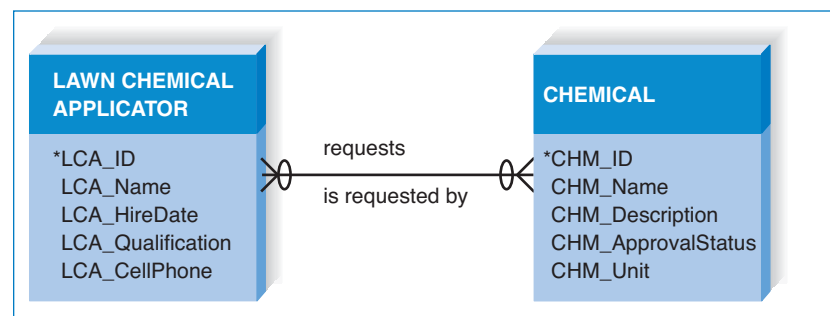


FIGURE 6-5
M:N Relationship

YOUR

TURN

6-1 UNDERSTANDING THE ELEMENTS OF AN ERD

A wealthy businessman owns a large number of paintings that he loans to museums all over the world. He is interested in setting up a system that records what he loans to whom so that he doesn't lose track of his investments. He would like to keep information about the paintings that he owns as well as the artists who painted them. He also wants to track the various museums that reserve his art, along with the actual reservations. Obviously, artists are associated with paintings, paintings are associated with reservations, and reservations are associated with museums.

2. Provide some basic attributes for each entity, and select an identifier, if possible.
3. Draw the appropriate relationships between the entities and label them.
4. What is the cardinality for each relationship? Depict this on your drawing.
5. What is the modality for each relationship? Depict this on your drawing.
6. List two business rules that are communicated by your ERD.

QUESTIONS:

1. Draw the four entities that belong on this data model.

Modality Second, relationships have a *modality* of null or not null, which refers to whether or not an instance of a child entity can exist without a related instance in the parent entity. Basically, the modality of a relationship indicates whether the child-entity instance is required to participate in the relationship. It forces you to ask questions like, Can you have a Chemical Request without a Chemical? and Can you have a Chemical without a Chemical Request? Modality is depicted by placing a zero on the relationship line next to the parent entity if nulls are allowed. A bar is placed on the relationship line next to the parent entity if nulls are not allowed.

In the two questions we just asked, the first answer is no: you need a chemical to have a chemical request. You can, however, have a chemical without having a chemical request for that chemical. The modality is “not null,” or “required,” for the first relationship in Figure 6-1. Notice, however, that a zero has been placed on the relationship line between Chemical and Chemical Request next to the Chemical Request entity. This means that chemicals can exist in our system without requiring that a chemical request exists. Said another way, instances of chemical requests are optional for a chemical. The modality is “null.”

The Data Dictionary and Metadata

As we described earlier, a CASE tool is used to help build ERDs. Every CASE tool has something called a *data dictionary*, which quite literally is where the analyst goes to define or look up information about the entities, attributes, and relationships on the ERD. Even Visio 2010, primarily known as a drawing tool, has some elementary data dictionary capabilities. Figures 6-6, 6-7, and 6-8 illustrate common data dictionary entries for an entity, an attribute, and a relationship; notice the kinds of information the data dictionary captures about each element.

The information you see in the data dictionary is called *metadata*, which, quite simply, is data about data. Metadata is anything that describes an entity, attribute, or relationship, such as entity names, attribute descriptions, and relationship

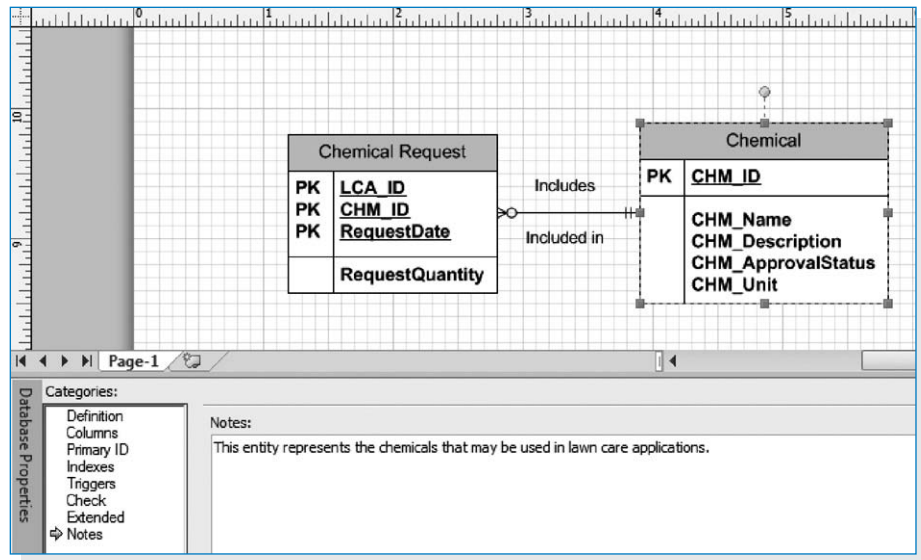


FIGURE 6-6
Data Dictionary Entry for Chemical Entity
(in Visio 2010)

cardinality, and it is captured to help designers better understand the system that they are building and to help users better understand the system that they will use. Figure 6-9 lists typical metadata that are found in the data dictionary. Notice that the metadata can describe an ERD element (like entity name) and also information that is helpful to the project team (like the user contact, the analyst contact, and special notes).

Metadata are stored in the data dictionary so that they can be shared and accessed by developers and users throughout the SDLC. The data dictionary allows you to record the standard pieces of information about your elements in one place, and it makes that information accessible to many parts of a project. For example, the data attributes in a data model also appear on the process models as elements of data stores and data flows, and on the user interface as fields on an input screen.

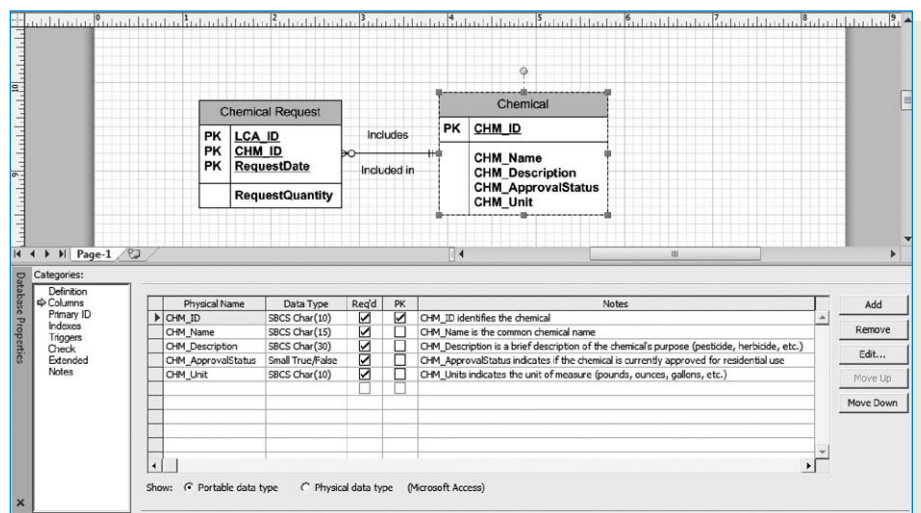


FIGURE 6-7
Data Dictionary Entry for Chemical
Attributes (in Visio 2010)

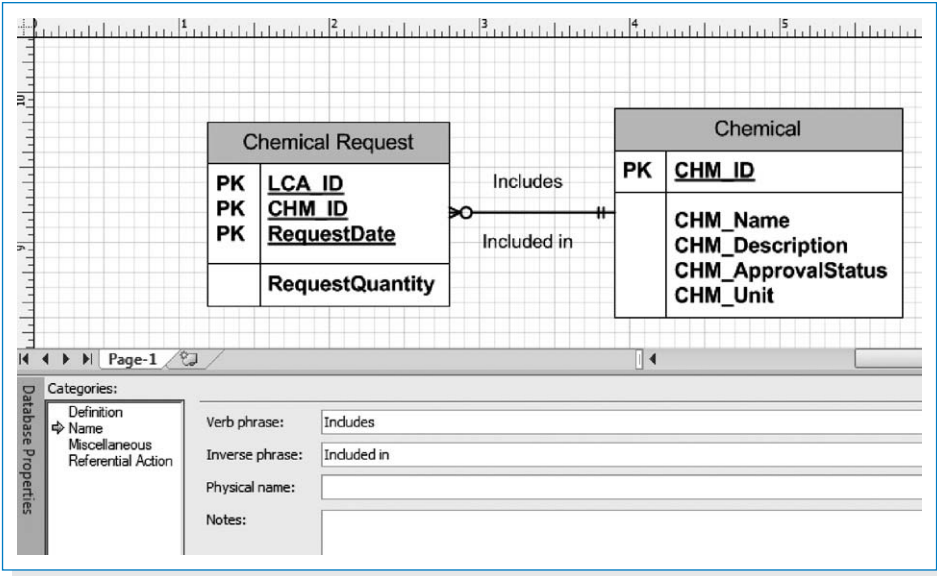


FIGURE 6-8
Data Dictionary Entry for a
Relationship (in Visio 2010)

ERD Element	Kinds of Metadata	Example
Entity	Name	Item
	Definition	Represents any item carried in inventory in the supermarket
	Special notes	Includes produce, bakery, and deli items
	User contact	Nancy Keller (x6755) heads up the item coding department
	Analyst contact	John Michaels is the analyst assigned to this entity
Attribute	Name	Item_UPC
	Definition	The standard Universal Product Code for the item based on Global Trade Item Numbers developed by GS1
	Alias	Item Bar Code
	Sample values	036000291452; 034000126453
	Acceptable values	Any 12-digit set of numerals
	Format	12 digit, numerals only
	Type	Stored as alphanumeric values
	Special notes	Values with the first digit of 2 are assigned locally, representing items packed in the store, such as meat, bakery, produce, or deli items. See Nancy Keller for more information.
Relationship	Verb phrase	Included in
	Parent entity	Item
	Child entity	Sold item
	Definition	An item is included in zero or more sold items. A sold item includes one and only one item.
	Cardinality	1:N
	Modality	Null
	Special notes	

FIGURE 6-9
Types of Metadata Captured by the Data Dictionary

When you make a change in the data dictionary, the change ripples to the relevant parts of the project that are affected.

When metadata are complete, clear, and shareable, the information can be used to integrate the different pieces of the analysis phase and ultimately lead to a much better design. It becomes much more detailed as the project evolves through the SDLC.

CREATING AN ENTITY RELATIONSHIP DIAGRAM

Drawing an ERD is an iterative process of trial and revision. It usually takes considerable practice. ERDs can become quite complex—in fact, there are systems that have ERDs containing hundreds or thousands of entities. The basic steps in building an ERD are these: (1) Identify the entities, (2) add the appropriate attributes to each entity, and then (3) draw relationships among entities to show how they are associated with one another. First, we will describe the three steps in creating ERDs, using the data model example from Figure 6-1. We will then discuss several advanced concepts of ERD's. Finally, we will present an ERD for Tune Source.

Building Entity Relationship Diagrams

Step 1: Identify the Entities As we explained, the most popular way to start an ERD is to first identify the entities for the model, and their attributes. The entities should represent the major categories of information that you need to store in your system. If you begin your data model by using a use case, look at the major inputs to the use case, the major outputs, and the information used for the use case steps. If the process models (e.g., DFDs) have been prepared, the easiest way to start is with them: The data stores on the DFDs, the external entities, and the data flows indicate the kinds of information that are captured and flow through the system.

The Chemical Request plays a key role in our chemical request system example, and so is included as a data entity. In addition, the Chemicals themselves will need to be described with data, and so will also be included as a data entity. Finally, we will need to capture information about the lawn care applicators (LCAs), since these individuals are the key actors in the system.

Step 2: Add Attributes and Assign Identifiers The information that describes each entity becomes its attributes. It is likely that you identified a few attributes if you read the chemical request system use cases and paid attention to the information flows on their DFDs. For example, an LCA has a name, and a chemical has a name

YOUR

T U R N

6-2 EVALUATE YOUR CASE TOOL

Examine the CASE tool that you will be using for your project, or find a CASE tool on the Web that you are interested in learning about. What kind of

metadata does its data dictionary capture? Does the CASE tool integrate data model information with other parts of a project? How?

FIGURE 6-10

Elements of the New Chemical Request Data Flow

Data flow name:	New Chemical Request
Data elements:	LCA ID + Chemical ID + Date of Request + Quantity

and description. Unfortunately, much of the information from the process models and use cases does not include enough detail to identify the exact attributes that should exist in each of our entities.

On a real project, there are a number of places you can go to figure out what attributes belong in your entity. For one, you can check in the CASE tool—often, an analyst will describe a process model data flow in detail when he or she enters the data flow into the CASE repository. For example, an analyst may create an entry for the chemical request information data flow like the one shown in Figure 6-10, which lists four data elements that make up the chemical request information. The elements of the data flow should be added to the ERD as attributes in your entities. A second approach is to check the requirements definition. Often, there is a section under functional requirements called data requirements. This section describes the data needs for the system that were identified while requirements were gathered. A final approach to identifying attributes is to use requirements elicitation techniques. The most effective techniques would be interviews (e.g., asking people who create and use reports about their data needs) or document analysis (e.g., examining existing forms, reports, or input screens).

Once the attributes are identified, one or more of them will become the entity's identifier. The identifier must be an attribute(s) that is able to uniquely identify a single instance of the entity. Look at Figure 6-1 and notice the identifiers that were selected for each entity.

Step 3: Identify Relationships The last step in creating ERDs is to determine how the entities are related to each other. Lines are drawn between entities that have relationships, each relationship is labeled, and cardinality and modality is assigned. The easiest approach is to begin with one entity and determine all the entities with which it shares relationships. In our example in Figure 6-1, we can see that an LCA makes chemical requests, and a chemical is included in a chemical request.

When you find a relationship to include on the model, you need to determine its cardinality and modality. For cardinality, ask how many instances of each entity participate in the relationship. You know that an LCA can make many chemical requests, but a specific chemical request is made by only one LCA. Therefore, we place a crow's foot next to the chemical request entity and a single bar closest to the LCA entity. This suggests that there is a 1:N relationship in which the LCA is the parent entity (the "1") and the chemical request is the child entity (the "many").

Next, we examine the relationship's modality. Can an LCA exist without an associated chemical request? In our example, the answer is "yes," so the modality is "null" or not required. A zero is placed next to the crow's foot near the chemical request. Now, can a chemical request exist without an associated LCA? This answer is "no," so the modality is "not null": or required, and we place a single bar next to the LCA entity.

The same type of thinking applies to determining the cardinality and modality of the relationship between chemical and chemical request. A chemical (the parent) may be included on many chemical requests (the child), so the relationship is 1:N. A chemical can exist without a chemical request, so the modality is "null";

however, a chemical request requires the existence of a chemical, so the modality is “not null.”

Again, remember that data modeling is an iterative process. Often, the assumptions you make and the decisions you make change as you learn more about the business requirements and as changes are made to the use cases and process models. But you have to start somewhere—so do the best you can with the three steps we just described and keep iterating until you have a model that works. Later in this chapter, we will show you a few ways to validate the ERDs that you draw.

Advanced Syntax

Now that we have created a data model according to the basic syntax that was presented earlier, we can move to several advanced concepts. We will explain three special types of entities and show how they can be used in our Chemical Request system.

Independent Entity An independent entity is an entity that can exist without the help of another entity, such as Lawn Chemical Applicator and Chemical. These entities all have identifiers that were created from their own attributes. Attributes from other entities were not needed to uniquely identify instances of these entities. Independent entities are drawn as rectangles with a single border line.

When a relationship includes an independent child entity, it is called a *non-identifying relationship*. This name is derived from the fact that parent entity attributes are not needed as part of the child entity’s identifier.

Dependent Entity There are situations when a child entity does require attributes from the parent entity to uniquely identify an instance. In these cases, the child entity is called a *dependent entity*, and its identifier consists of at least one attribute from the parent entity.

A good example of a dependent entity is the Chemical Request entity shown in Figure 6-1. A Chemical Request is made by a specific LCA and includes a specific chemical. We include the LCA_ID and the Chemical_ID plus the request date to fully identify each Chemical Request, Chemical Request is considered a dependent entity and is shown as a rectangle with a double border line.

When relationships have a dependent child entity, they are called *identifying relationships*. This name is derived from the fact that parent entity attributes are needed as part of the child entity’s identifier.

Intersection Entity A third kind of entity is the *intersection entity*. It exists in order to capture some information about the relationship between two other entities. Typically, intersection entities are added to a data model to store information about two entities sharing an M:N relationship. These entities are also called Associative Entities. Think back to the M:N relationship between LCA and Chemical shown in Figure 6-5. In that figure, one instance of an LCA could request many Chemicals, and a Chemical can be requested by many LCAs. A difficulty arises if we want to capture the date on which a particular chemical was requested by a specific LCA. We cannot put the date in the Chemical entity, because the Chemical is requested by many LCAs. We also cannot put the date in the LCA entity, because there are many Chemicals requested by the LCA. Therefore, we need another entity that enables us to associate a specific chemical with a specific LCA on a specific date.

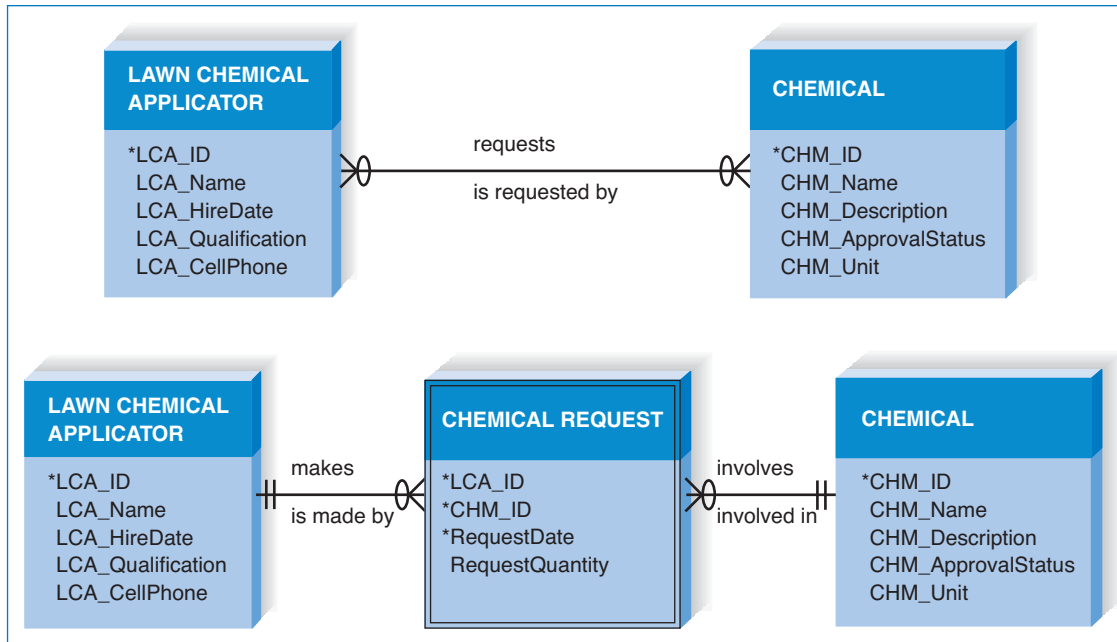


FIGURE 6-11
Resolving an M:N Relationship

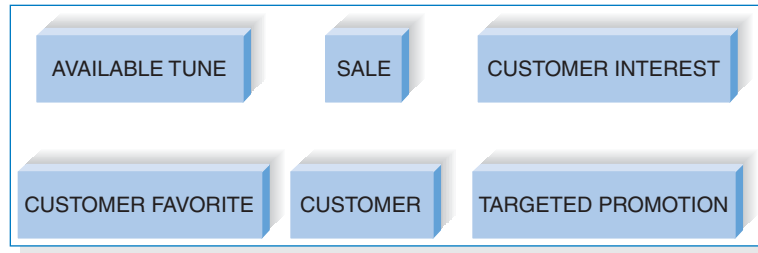
The process of adding an intersection entity is called “resolving an M:N relationship” because it eliminates the M:N relationship and its associated problems from the data model. There are three steps involved in adding an intersection entity. Step 1: Remove the M:N relationship line and insert a new entity in between the two existing ones. Step 2: Add two 1:N relationships to the model. The two original entities should serve as the parent entities for each 1:N, and the new intersection entity becomes the child entity in both relationships. Step 3: Name the intersection entity. Intersection entities are often named by a concatenation of the two entities that created it (e.g., Chemical Request), making its meaning clear. Alternatively, the entity can be given another appropriate name. Figure 6-11 shows the M:N LCA-Chemical relationship and how it was resolved with the use of an intersection entity.

Are intersection entities dependent or independent? Actually, it depends. Sometimes an intersection entity has a logical identifier that can uniquely identify its instances. For example, an intersection entity between a student and a course (a student may take many courses and a course is taken by many students) may be called a *transcript*. If transcripts have unique transcript numbers, then the entity would be considered independent. In contrast, the Chemical Request intersection entity in Figure 6-11 requires the identifiers from both LCA and Chemical for an instance to be uniquely identified. Thus, Chemical Request is a dependent entity.

Applying the Concepts at Tune Source

Let’s go through one more example of creating a data model by using the context of Tune Source. For now, review the use cases that were presented in Figure 4-14 and the final level 0 process model presented in Figure 5-17.

FIGURE 6-12
Entities for Tune Source ERD



Identify the Entities When you examine the Tune Source level 0 DFD, you see that there are six data stores: customer, sale, available tunes, customer interests, customer favorites, and targeted promotions. Each of these unique types of data likely will be represented by entities on a data model.

As a next step, you should examine the external entities and ask yourself, “Will the system need to capture information about any of these entities?” You may be tempted to include marketing managers, but there really is no need to track information about these in our system. Later, we may want to track system users, passwords, and data access privileges, but this information has to do with the *use* of the new system and would not be added until the physical data model is created in the design phase.

It is good practice to also look at the data flows on your process model and make sure that all of the information that flows through the system has been covered by your ERD. It appears that the main entities for Tune Source have been identified after an examination of the data stores and external entities. See Figure 6-12 for the beginning of our data model.

Identify the Attributes The next step is to select which attributes should be used to describe each entity. It is likely that you identified a handful of attributes if you read the Tune Source use cases and examined the DFDs. For example, an available tune has an artist, title, genre, and length, and some attributes of customer are name and contact information, which likely includes address, phone number, and e-mail address.

The two entities customer favorite and customer interest might seem similar at first glance, but they are used to capture different types of information about the customer’s music preferences. A customer favorite is a tune that the customer specifically adds to his/her Favorites list in order to monitor tunes as the Web site is searched and browsed. In a sense, it’s like a future shopping list, so we just record the customer’s ID, the tune’s ID, and the date the tune was added to the list. The customer’s Favorites are available each time the customer revisits the site to help in recalling tunes previously discovered and to (hopefully) purchase them. On the other hand, the customer interest is created automatically as the customer investigates tunes and listens to samples. Customer interests are used by the marketing department to help design promotions for the customer that will be tailored to the types of music the customer has explored. Slightly different attributes are associated with these two entities because of their different purposes in the system.

Targeted promotions are special offers that will be created for a customer on the basis of his or her interests and with regard to sales patterns. A promotion will include a sale price for a specific tune if it is purchased within a specific

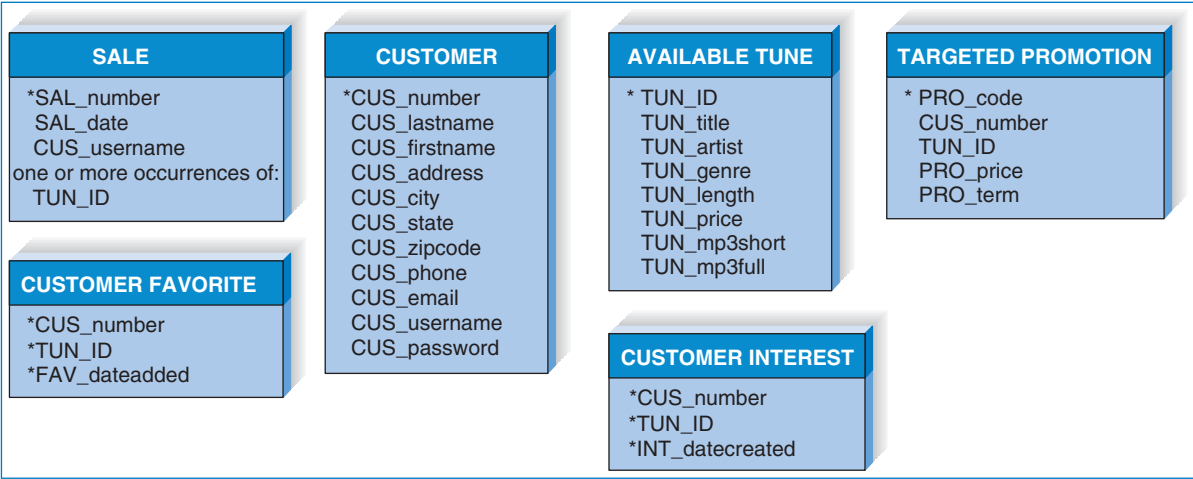


FIGURE 6-13
Attributes and Identifiers for Tune Source ERD

time frame. Attributes for the targeted promotions are listed in Figure 6-13. Finally, we also see that several attributes associated with a tune sale have been listed in the ERD.

To determine the entity identifiers, we consider the attribute or attributes that will uniquely identify each entity. We will establish a customer number for each customer in the system. Each available tune that we have will be assigned a unique tune ID. Each targeted promotion will be given a unique promotion code, and each sale will be given a unique sale number. Finally, each customer favorite and each customer interest can be uniquely identified by the customer number, tune ID, and the date created.

The customer, sale, available tune, and targeted promotion entities are independent entities; attributes from other entities are not needed to uniquely identify instances. The identifiers for customer interests and customer favorites, however, do rely on attributes from their parent entities: customer and available tune. This is because a customer favorite (or a customer interest) is uniquely identified by the customer who created it, the tune involved, and the date it was created. Therefore, since these two entities draw part of their primary keys from their parent entities, they are considered dependent entities.

Identify the Relationships The last step in creating ERDs is to determine how the entities are related to each other. Lines are drawn between entities that have relationships, and each relationship is labeled and assigned a cardinality and modality. As shown in Figure 6-14, a customer may make many sales, but a sale is made by one customer. A sale is not required for a particular customer instance, but a customer is required for a sale. A customer may be targeted by many targeted promotions, but a targeted promotion is for one customer. A targeted promotion is not required for a customer, but a customer is required for a targeted promotion. A customer creates many favorites, but a favorite is created by only one customer. A favorite is not required for a customer, but a customer is required for a favorite. An available tune may be included in many customers' favorites, but a favorite includes only one tune. A favorite is not required for an

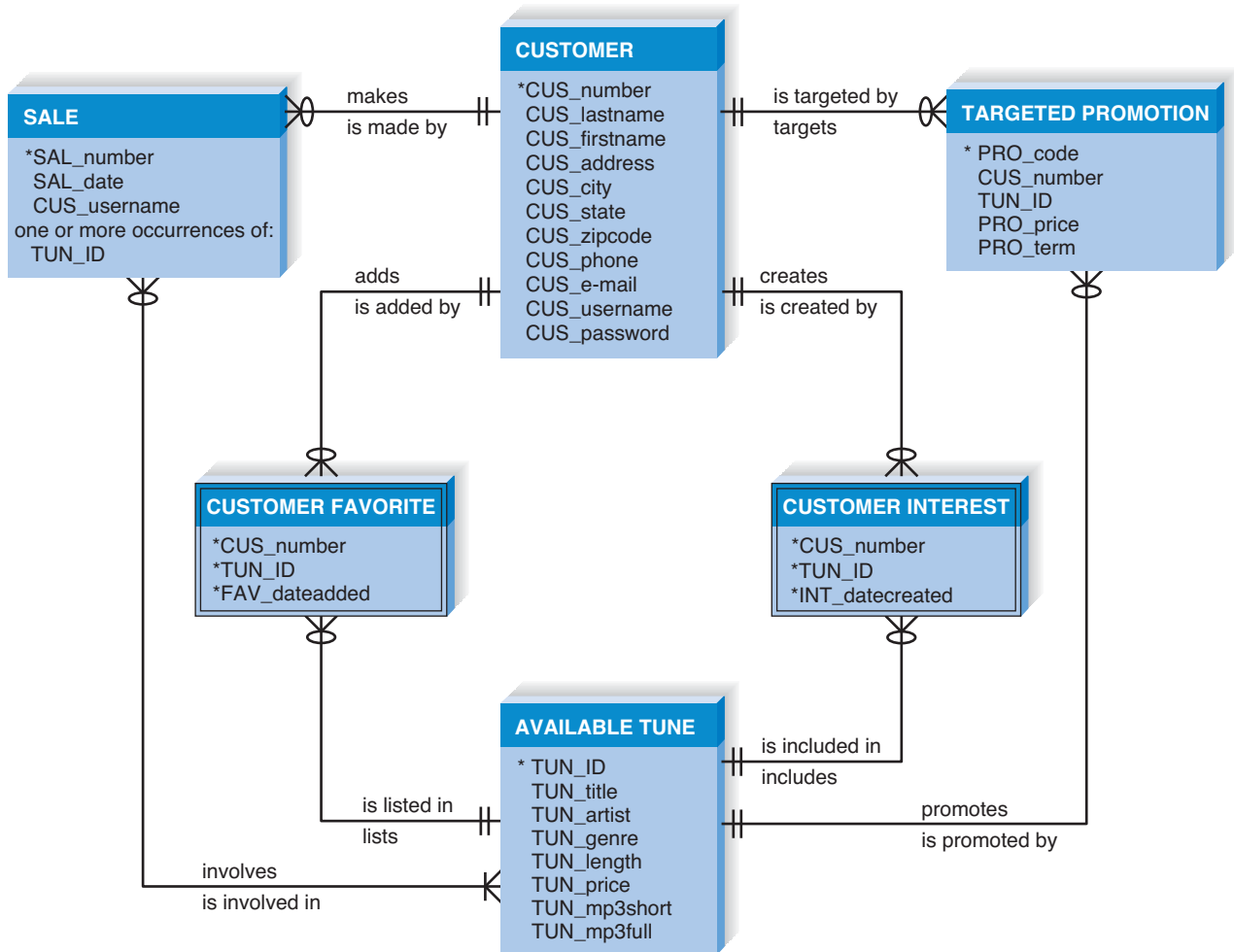


FIGURE 6-14
Relationships for Tune Source ERD

available tune, but a tune is required for a favorite. (The same relationships apply to customer–interest–available tune). We can also see that an available tune may be promoted by many targeted promotions, but a targeted promotion promotes only one tune. An available tune is not required to have a targeted promotion, but a targeted promotion must be associated with an available tune. Finally, we see that customers may place many sales, but a sale is not required for a customer. A sale belongs to one and only one customer. A sale may include many tunes, and a tune may be included on many sales. A tune is required on a sale, but a sale is not required for a tune.

The customer–customer favorite, customer favorite–available tune, customer–customer interest, and customer interest–available tune relationships are identifying relationships. All other relationships are nonidentifying relationships.

As a final step in the creation of the Tune Source ERD, we should resolve any M:N relationships in the data model. A look at Figure 6-14 shows one such relationship, between sale and available tune. See Your Turn 6-6 and resolve this relationship on your own.

VALIDATING AN ERD

As you probably guessed from the previous section, creating ERDs is pretty tough. It takes a lot of experience to draw ERDs well, and there are not many black-and-white rules to help guide you. Luckily, there are some general design guidelines that you can keep in mind as you build ERDs, and once the ERDs are drawn, you can use a technique called *normalization* to validate that your models are well formed. Another technique is to check your ERD against your process models to make sure that both models balance each other.

Design Guidelines

Design guidelines are not rules that must be followed; rather, they are “best practices” that often lead to better quality diagrams. For example, labels and naming conventions are important for creating clear ERDs. Names should not be ambiguous (e.g., name, number); instead, they should clearly communicate what the model component represents. These names should be consistent across the model and reflect the terminology used by the business. If Tune Source refers to people who order music as *customers*, the data model should include an entity called customer, not client or stakeholder.

There are no rules covering the layout of ERD components. They can be placed anywhere you like on the page, although most systems analysts try to put the

CONCEPTS

6-A THE USER’S ROLE IN DATA MODELING

IN ACTION

I have two very different stories regarding data models. First, when I worked with First American Corporation, the head of Marketing kept a data model for the marketing systems hanging on a wall in her office. I thought this was a little unusual for a high-level executive, but she explained to me that data was critical for most of the initiatives that she puts in place. Before she can approve a marketing campaign or new strategy, she likes to confirm that the data exists in the systems and that it’s accessible to her analysts. She has become very good at understanding ERDs over the years because they had been such an important communications tool for her to use with her own people and with IT.

On a very different note, here is a story I received from a friend of mine who heads up an IT department:

“We were working on a business critical, time dependent development effort, and VERY senior management decided that the way to ensure success was to have the various teams do technical design walkthroughs to senior management on a weekly basis. My team was responsible for the data architecture and database design. How could senior management, none of whom

probably had ever designed an Oracle architecture, evaluate the soundness of our work?

So, I had my staff prepare the following for the one (and only) design walkthrough our group was asked to do. First, we merged several existing data models and then duplicated each one . . . that is, every entity and relationship printed twice (imitating, if asked, the redundant architecture). Then we intricately color coded the model and printed the model out on a plotter and printed one copy of every inch of model documentation we had. On the day of the review, I simply wheeled in the documentation and stretched the plotted model across the executive boardroom table. ‘Any questions,’ I asked? ‘Very impressive,’ they replied. That was it! My designs were never questioned again.” *Barbara Wixom*

QUESTIONS:

1. From these two stories, what do you think is the user’s role in data modeling?
2. When is it appropriate to involve users in the ERD creation process?
3. How can users help analysts create better ERDs?

YOUR

6-3 CAMPUS HOUSING SYSTEM

TURN

Consider the accompanying system, which was described in Chapter 4. Use the use cases and process models that you created in Chapters 4 and 5 to help you answer the questions that follow.

The Campus Housing Service helps students find apartments. Owners of apartments fill in information forms about the rental units they have available (e.g., location, number of bedrooms, monthly rent). Students who register with the service can search the rental information to find apartments that meet their needs (e.g., a two-bedroom apartment for \$800 or less per month within 1/2 mile of campus). They then contact the apartment

owners directly to see the apartment and, possibly, rent it. Apartment owners call the service to delete their listing when they have rented their apartment(s).

QUESTIONS:

1. What entities would you include on a data model?
2. What attributes would you list for each entity? Select an identifier for each entity, if possible.
3. What relationships exist between the entities that you identified? Label the relationships appropriately, and denote the cardinality and modality of each relationship.

entities together that are related to each other. If the model becomes too complex or busy (some companies have hundreds of entities on a data model), the model can be broken down into *subject areas*. Each subject area would contain related entities and relationships, and the analyst can work with one group of entities at a time to make the modeling process less confusing.

In general, data modeling can be quite tricky, mainly because the data model is heavily based on interpretation; therefore, when business rules change, the relationships or other data model components will have to be altered. *Assumptions* are an important part of data modeling. It is important that we verify all assumptions about business rules so that our data model is correct.

YOUR

6-4 INDEPENDENT ENTITIES

TURN

Locate the independent entities on Figure 6-14. How do you know which of the entities are independent? Locate the nonidentifying relationships.

How did you find them? Can you create a rule that describes the association between independent entities and nonidentifying relationships?

YOUR

6-5 DEPENDENT ENTITIES

TURN

Locate the dependent entities on Figure 6-14. Locate the identifying relationships. How did you find them? Can you create a rule that describes the

association between dependent entities and identifying relationships?

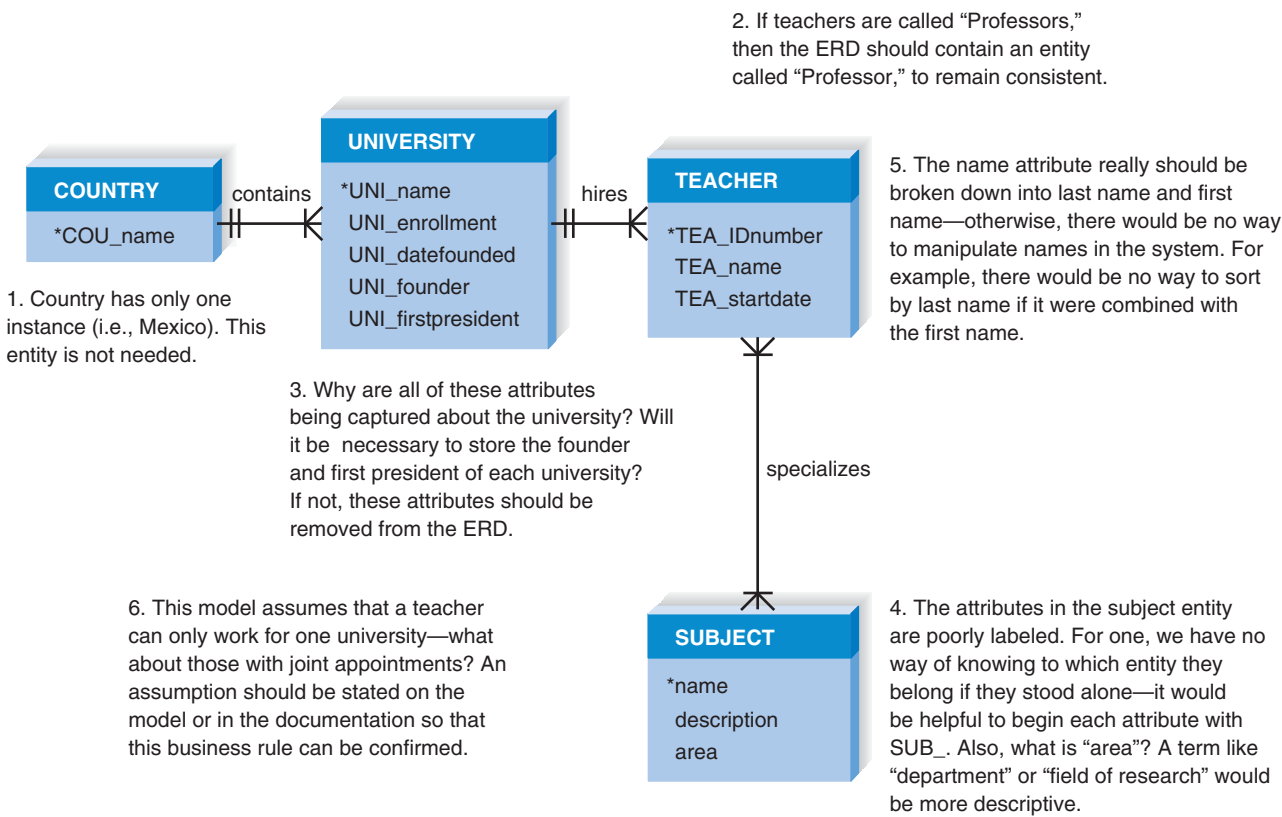


FIGURE 6-15
Data Modeling Guidelines Summary

Therefore, when you model data, don't panic or become overwhelmed by details. Rather, add components to the diagram slowly, knowing that they will be changed and rearranged many times. Make assumptions along the way and then confirm these assumptions with the business users. Work iteratively and constantly challenge the data model with business rules and exceptions to see whether the diagram is communicating the business system appropriately. Figure 6-15 summarizes the guidelines presented in this chapter to help you evaluate your data model.

**YOUR
TURN**

6-6 INTERSECTION ENTITIES

Resolve the M:N relationship between the sale and available tune that is shown in Figure 6-14. What kinds of information could you capture about this relationship? What would the new ERD look like? Would the intersection entity be considered dependent or independent?

Can you think of other kinds of M:N relationships that exist in the real world? How would you resolve these M:N relationships if you were to include them on an ERD?

CONCEPTS**6-B IMPLEMENTING AN EIM SYSTEM****IN ACTION**

A large direct health and insurance medical provider needed an enterprise information management (EIM) system to enable enterprisewide information management and to support the effective use of data for critical cross-functional decision making. In addition, the company needed to resolve issues related to data redundancy, inconsistency, and unnecessary expenditure. The company faced several information challenges: The company data resided in multiple locations, the data were developed for department-specific use, and there was limited enterprise access. In addition, data definitions

were created by individual departments and were not standardized, and data were being managed by multiple departments within the company.

Source: http://www.deloitte.com/dtt/case_study/o,1005,sid%253D26562%2526cid%253D132760,00.html

QUESTIONS:

1. What solution would you propose for this company?
2. Discuss the role that data modeling would play in a project to solve this problem.

Normalization

Once you have created your ERD, there is a technique called *normalization* that can help analysts validate the models that they have drawn. It is a process whereby a series of rules are applied to a logical data model or a file to determine how well formed it is. Normalization rules help analysts identify entities that are not represented correctly in a logical data model, or entities that can be broken out from a file. The result of the normalization process is that the data attributes are arranged to form stable, yet flexible, relations for the data model. In Appendix 6A, we describe three normalization rules that are applied regularly in practice.

Balancing Entity Relationship Diagrams with Data Flow Diagrams

All the analysis activities of the systems analyst are interrelated. For example, the requirements analysis techniques are used to determine how to draw both the

YOUR**6-7 BOAT CHARTER COMPANY****T U R N**

A charter company owns boats that are used for charter trips to islands. The company has created a computer system to track the boats it owns, including each boat's ID number, name, and seating capacity. The company also tracks information about the various islands, such as their names and populations. Every time a boat is chartered, it is important to know the date that the trip is to take place and the number of people on the trip. The company also keeps information about each captain, such as Social Security number,

name, birthdate, and contact information for next of kin. Boats travel to only one island per visit.

QUESTIONS:

1. Create a data model. Include entities, attributes, identifiers, and relationships.
2. Which entities are dependent? Which are independent?
3. [Optional] Use the steps of normalization to put your data model in 3NF. Describe how you know that it is in 3NF.

process models and data models, and the CASE repository is used to collect information that is stored and updated throughout the entire analysis phase. Now we will see how the process models and data models are interrelated.

Although the process model focuses on the processes of the business system, it contains two data components—the data flow (which is composed of data elements) and the data store. The purposes of these are to illustrate what data are used and created by the processes and where those data are kept. These components of the DFD need to *balance* with the ERD. In other words, the DFD data components need to correspond with the ERD's data stores (i.e., entities) and the data elements that comprise the data flows (i.e., attributes) depicted on the data model.

Many CASE tools offer the feature of identifying problems with balance between DFDs and ERDs; however, it is a good idea to understand how to identify problems on your own. This involves examining the data model you have created and comparing it with the process models that have been created for the system. Check your data model and see whether there are any entities you have created that do not appear as data stores on your process models. If there are, you should add them to your process models to reflect your decision to store information about that entity in your system.

Similarly, the bits of information that are contained in the data flows (these are usually defined in the CASE entry for the data flow) should match up to the attributes found in entities in the data models. For example, if the customer information data flow that goes from the *customer* entity to the *purchase tunes* process were defined as having customer name, e-mail address, and home address, then each of these pieces of information should be recorded as attributes in the *customer* entity on the data model. We must verify that all the data items included in the data stores and data flows in the process model have been included somewhere as an entity attribute in the data model. We want to ensure that the data model fully incorporates all the data identified in the process model. If it does not, then the data model is incomplete. In addition, all the data elements in the data model should appear as a part of a data store and data flow(s) in the process model. If some data elements have been omitted from the process model, then we need to investigate whether those data items are truly needed in the processing of the system. If they are needed, they must be added to the process model data stores and data flows; otherwise, they should be deleted from the data model as extraneous data items.

A useful tool to clearly depict the interrelationship between process and data models is the *CRUD matrix*. The CRUD (create, read, update, delete) matrix is a table that depicts how the system's processes use the data within the system. It is helpful to develop the CRUD matrix on the basis of the logical process and data models and then revise it later in the design phase. The matrix also provides important information for program specifications, because it shows exactly how data are created and used by the major processes in the system.

To create a CRUD matrix, a table is drawn listing all the system processes along the top, and all the data entities (and entity attributes) along the left-hand side of the table. Then, from the information presented in the process model, the analyst fills in each cell with a C, R, U, D, (or nothing) to describe the process's interaction with each data entity (and its attributes). Figure 6-16 shows a portion of a data flow diagram and the CRUD matrix that can be derived from it. As you can see, if a process reads information from a data store, but does not update it, there should be a data flow coming out of the data store only. When a process

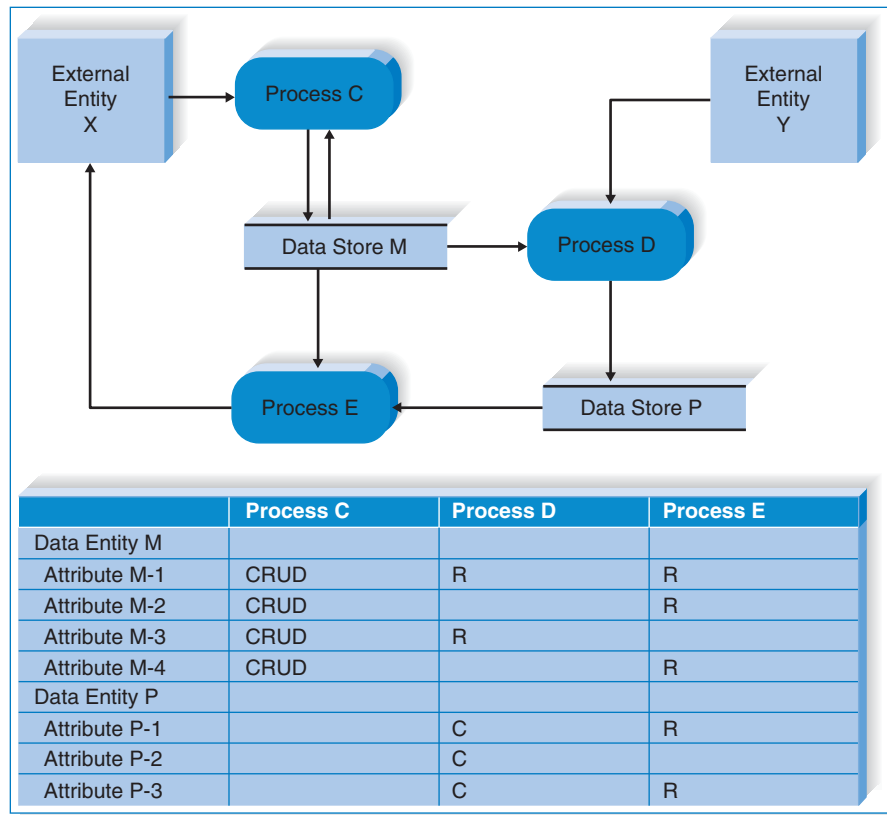


FIGURE 6-16
Partial Process Model and CRUD Matrix

updates a data store in some way, there should be a data flow going from the process to the data store.

Thinking carefully about the content of the data flows in the process models, we can identify places where attributes may have been omitted from the data stores/entities. In addition, we can verify that every attribute is created, read, updated, and deleted somewhere in the process model. If it is not read by some process, then the attribute is probably not needed. If it is not created or updated, the attribute probably needs to be added to a data flow(s) in the process model.

SUMMARY

Basic Entity Relationship Diagram Syntax

The entity relationship diagram (ERD) is the most common technique for drawing a data model, a formal way of representing the data that are used and created by a business system. There are three basic elements in the data modeling language, each of which is represented by a different graphic symbol. The entity is the basic building block for a data model. It is a person, place, or thing about which data are collected. An attribute is some type of information that is captured about an entity.

The attribute that can uniquely identify one instance of an entity is called the identifier. The third data model component is the relationship, which conveys the associations between entities. Relationships have cardinality (the ratio

of parent instances to child instances) and modality (a parent needs to exist if a child exists). Information about all of the components is captured by metadata in the data dictionary.

Creating an Entity Relationship Diagram

The basic steps in building an ERD are (1) identify the entities, (2) add the appropriate attributes to each entity, and (3) draw relationships among entities to show how they are associated with one another. There are three special types of entities that ERDs contain. Most entities are independent, because one (or more) attribute can be used to uniquely identify an instance. Entities that rely on attributes from other entities to identify an instance are dependent. An intersection entity is placed between two entities to capture information about their relationship. In general, data models are based on interpretation; therefore, it is important to clearly state assumptions that reflect business rules.

Validating an Entity Relationship Diagram

Normalization, the process whereby a series of rules is applied to the logical data model to determine how well formed it is, is described in the Chapter 6 Appendix. A logical data model is in first normal form (1NF) if it does not contain repeating attributes, which are attributes that capture multiple values for a single instance. Second normal form (2NF) requires that all entities are in 1NF and contain only attributes whose values are dependent on the whole identifier (i.e., no partial dependency). Third normal form (3NF) occurs when a model is in both 1NF and 2NF and none of the resulting attributes is dependent on nonidentifier attributes (i.e., no transitive dependency). With each violation, additional entities should be created to remove the repeating attributes or improper dependencies from the existing entities. Finally, ERDs should be balanced with the data flow diagrams (DFDs)—which were presented in Chapter 5—by making sure that data model entities and attributes correspond to data stores and data flows on the process model. The CRUD matrix is a valuable tool to use when balancing process and data models.

KEY TERMS

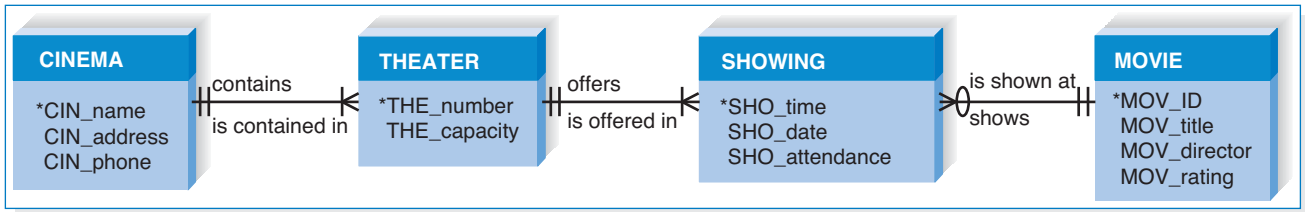
1:1 relationship	Derived attribute	Nonidentifying relationship
1:N relationship	Entity	Normalization
Assumption	Entity relationship diagram (ERD)	Parent entity
Attribute	First normal form (1NF)	Partial dependency
Balance	IDEF1X	Physical data model
Business rule	Identifier	Relationship
Cardinality	Identifying relationship	Repeating attributes
Child entity	Independent entity	Repeating groups
Concatenated identifier	Instance	Second normal form (2NF)
CRUD matrix	Intersection entity	Subject area
Data dictionary	Logical data model	Third normal form (3NF)
Data model	M:N relationship	Transcript
Dependent	Metadata	Transitive dependency
Dependent entity	Modality	

QUESTIONS

1. Provide three different options that are available for selecting an identifier for a student entity. What are the pros and cons of each option?
2. What is the purpose of developing an identifier for an entity?
3. What type of high-level business rule can be stated by an ERD? Give two examples.
4. Define what is meant by an *entity* in a data model. How should an entity be named? What information about an entity should be stored in the CASE repository?
5. Define what is meant by an *attribute* in a data model. How should an attribute be named? What information about an attribute should be stored in the CASE repository?
6. Define what is meant by a *relationship* in a data model. How should a relationship be named? What information about a relationship should be stored in the CASE repository?
7. A team of developers is considering including “warehouse” as an entity in its data model. The company for whom they are developing the system has just one warehouse location. Should “warehouse” be included? Why or why not?
8. What is meant by a concatenated identifier?
9. Describe, in terms a businessperson could understand, what are meant by the cardinality and modality of a relationship between two entities.
10. What are metadata? Why are they important to system developers?
11. What is an independent entity? What is a dependent entity? How are the two types of entities differentiated on the data model?
12. Explain the distinction between identifying and nonidentifying relationships.
13. What is the purpose of an intersection entity? How do you know whether one is needed in an ERD?
14. Describe the three-step process of creating an intersection entity.
15. Is an intersection entity dependent or independent? Explain your answer.
16. What is the purpose of normalization?
17. Describe the analysis that is applied to a data model in order to place it in first normal form (1NF).
18. Describe the analysis that is applied to a data model in order to place it in second normal form (2NF).
19. Describe the analysis that is applied to a data model in order to place it in third normal form (3NF).
20. Describe how the data model and process model should be balanced against each other.
21. What is a CRUD matrix? How does it relate to process models and data models?

EXERCISES

- A. Draw data models for the following entities:
 - Movie (title, producer, length, director, genre)
 - Ticket (price, adult or child, showtime, movie)
 - Patron (name, adult or child, age)
- B. Draw a data model for the following entities, considering the entities as representing a system for a patient billing system and including only the attributes that would be appropriate for this context:
 - Patient (age, name, hobbies, blood type, occupation, insurance carrier, address, phone)
 - Insurance carrier (name, number of patients on plan, address, contact name, phone)
 - Doctor (specialty, provider identification number, golf handicap, age, phone, name)
- C. Draw the relationships that follow. Would the relationships be identifying or nonidentifying? Why?
 - A patient must be assigned to only one doctor, and a doctor can have many patients.
 - An employee has one phone extension, and a unique phone extension is assigned to an employee.
 - A movie theater shows many different movies, and the same movie can be shown at different movie theaters around town.
- D. Draw an entity relationship diagram (ERD) for the following situations:
 1. Whenever new patients are seen for the first time, they complete a patient information form that asks their name, address, phone number, and insurance carrier, all of which are stored in the patient information file. Patients can be signed up with only one carrier, but they must be signed up



to be seen by the doctor. Each time a patient visits the doctor, an insurance claim is sent to the carrier for payment. The claim must contain information about the visit, such as the date, purpose, and cost. It would be possible for a patient to submit two claims on the same day.

2. The state of Georgia is interested in designing a database that will track its researchers. Information of interest includes researcher name, title, position; university name, location, enrollment; and research interests. Each researcher is associated with only one institution, and each researcher has several research interests.
 3. A department store has a bridal registry. This registry keeps information about the customer (usually the bride), the products that the store carries, and the products for which each customer registers. Customers typically register for a large number of products, and many customers register for the same products.
 4. Jim Smith's dealership sells Fords, Hondas, and Toyotas. The dealership keeps information about each car manufacturer with whom it deals so that employees can get in touch with manufacturers easily. The dealership also keeps information about the models of cars that it carries from each manufacturer. It keeps such information as list price, the price the dealership paid to obtain the model, and the model name and series (e.g., Honda Civic LX). The dealership also keeps information about all sales that it has made. (For instance, employees will record the buyer's name, the car the buyer bought, and the amount the buyer paid for the car.) To allow employees to contact the buyers in the future, contact information is also kept (e.g., address, phone number, e-mail).
- E. Examine the data models that you created for Exercise D. How would the respective models change (if at all) on the basis of these corresponding new assumptions?
- Two patients have the same first and last names.
 - Researchers can be associated with more than one institution.
 - The store would like to keep track of purchased items.
 - Many buyers have purchased multiple cars from Jim over time because he is such a good dealer.
- F. Visit a Web site that allows customers to order a product over the Web (e.g., Amazon.com). Create a data model that the site needs to support its business process. Include entities to show what types of information the site needs. Include attributes to represent the type of information the site uses and creates. Finally, draw relationships, making assumptions about how the entities are related.
 - G. Create metadata entries for the following data model components and, if possible, input the entries into a computer-aided software engineering (CASE) tool of your choosing:
 - Entity—product
 - Attribute—product number
 - Attribute—product type
 - Relationship—company makes many products, and any one product is made by only one company.
 - H. Describe the assumptions that are implied from the data model shown at the top of this page.
 - I. Create a data model for one of the processes in the end-of-chapter Exercises for Chapter 4. Explain how you would balance the data model and process model.
 - J. Apply the steps of normalization to validate the models you drew in Exercise D.
 - K. You have been given a file that contains fields relating to CD information. Using the steps of normalization, create a logical data model that represents this file in third normal form. The fields include the following:
 - Musical group name
 - Musicians in group
 - Date group was formed

- Group's agent
- CD title 1
- CD title 2
- CD title 3
- CD 1 length
- CD 2 length
- CD 3 length

The assumptions are as follows:

- Musicians in group contains a list of the members of the people in the musical group.
- Musical groups can have more than one CD, so both group name and CD title are needed to uniquely identify a particular CD.

MINICASES

1. West Star Marinas is a chain of 12 marinas that offer lakeside service to boaters; service and repair of boats, motors, and marine equipment; and sales of boats, motors, and other marine accessories. The systems development project team at West Star Marinas has been hard at work on a project that eventually will link all the marina's facilities into one unified, networked system.

The project team has developed a logical process model of the current system. This model has been carefully checked for syntax errors. Last week, the team invited a number of system users to role-play the various data flow diagrams, and the diagrams were refined to the users' satisfaction. Right now, the project manager feels confident that the as-is system has been adequately represented in the process model.

The director of operations for West Star is the sponsor of this project. He sat in on the role-playing of the process model and was very pleased by the thorough job the team had done in developing the model. He made it clear to you, the project manager, that he was anxious to see your team begin work on the process model for the to-be system. He was a little skeptical that it was necessary for your team to spend any time modeling the current system in the first place, but grudgingly admitted that the team really seemed to understand the business after going through that work.

The methodology that you are following, however, specifies that the team should now turn its attention to developing the logical data model for the as-is system. When you stated this to the project sponsor, he seemed confused and a little irritated. "You are going to spend even more time looking at the current system? I thought you were done with that! Why is this necessary? I want to see some progress on the way things will work in the future!"

- a. What is your response to the director of operations?
- b. Why do we perform data modeling?

- c. Is there any benefit to developing a data model of the current system at all?
 - d. How does the process model help us develop the data model?
2. The system development team at the Wilcon Company is working on developing a new customer order entry system. In the process of designing the new system, the team has identified the following data entity attributes:

Inventory Order

Order Number (identifier)

Order Date

Customer Name

Street Address

City

State

Zip

Customer Type

Initials

District Number

Region Number

1 to 22 occurrences of:

Item Name

Quantity Ordered

Item Unit

Quantity Shipped

Item Out

Quantity Received

- a. State the rule that is applied to place an entity in first normal form. Revise this data model so that it is in first normal form.
- b. State the rule that is applied to place an entity into second normal form. Revise the data model (if necessary) to place it in second normal form.
- c. State the rule that is applied to place an entity into third normal form. Revise the data model to place it in third normal form.
- d. What other guidelines and rules can you follow to validate that your data model is in good form?

APPENDIX 6A: NORMALIZING THE DATA MODEL

In this Appendix, we describe the rules of normalization that help analysts improve the quality of the data model. These rules help identify entities that are not represented correctly in the logical data model and entities that can be broken out from a file. The result of the normalization process is that the data attributes are arranged to form stable yet flexible relations for the data model. Typically, three rules of normalization are applied regularly in practice. (See Figure 6A-1.) We describe these rules and illustrate them with an example here.

First Normal Form A logical data model is in first normal form (1NF) if it does not contain attributes that have repeating values for a single instance of an entity.

Often, this problem is called repeating attributes, or repeating groups. Every attribute in an entity should have only one value per instance for the model to “pass” 1NF.

Let’s pretend that the Tune Source project team was given the layout for the CD purchase file that is used by the existing CD sales system. The team members are anxious to incorporate the data from this file into their own system, and they decide to put the file into third normal form to make the information easier to understand and, ultimately, easier for them to add to the data model for the new Digital Music Download system. See Figure 6A-2 for the file layout that the project team received.

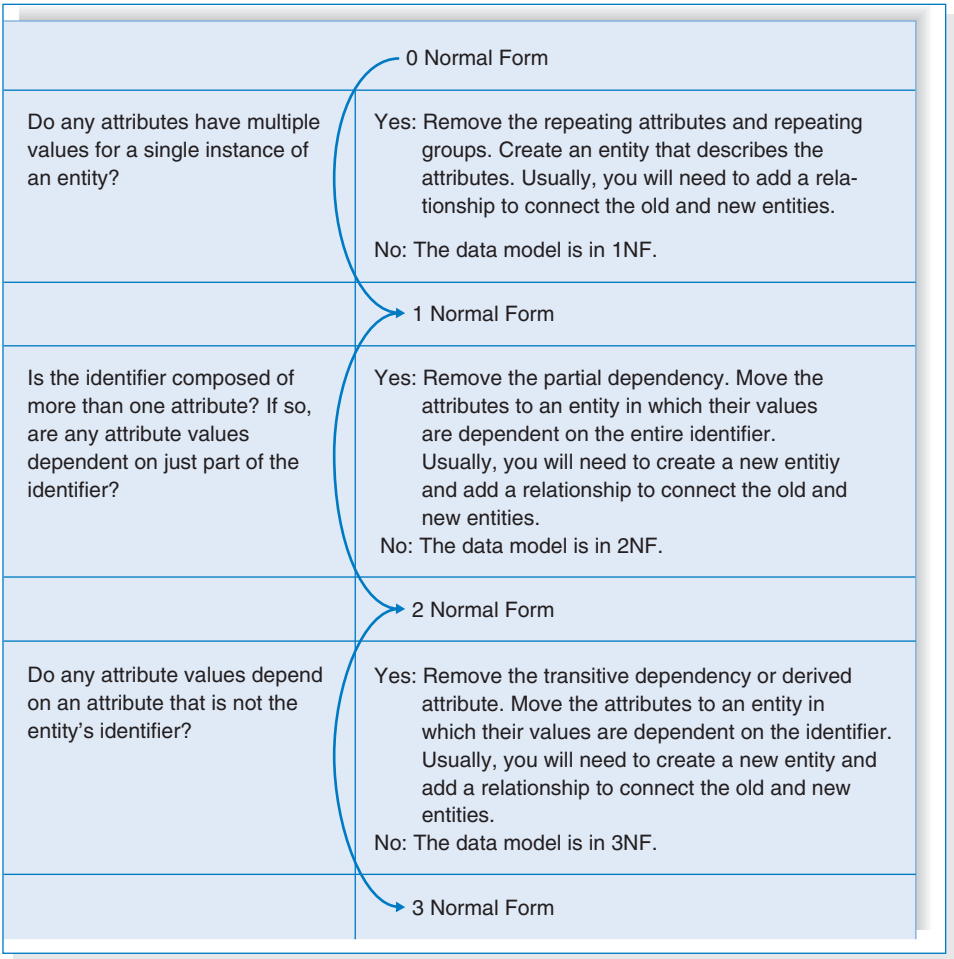


FIGURE 6A-1
Normalization Steps

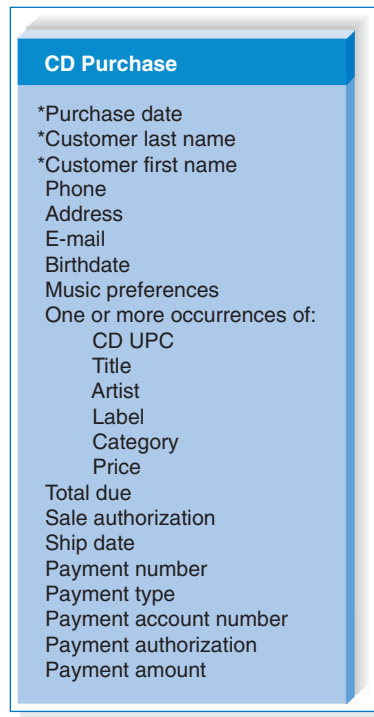


FIGURE 6A-2
Initial CD Sales System File

If you examine the file carefully, you should notice that there are two cases in which multiple values are captured for one or more attributes. The most obvious example is the multiple occurrences of CDs that are included in the purchase, a clear violation of 1NF. The repeated group of attributes about each CD included in the purchase should be removed by creating a new entity called CD and placing all of the CD attributes into it. The relationship between purchase and CD is M:N, since a purchase can include many CDs and a CD can be included in many purchases.

The second violation of 1NF may not be as readily noticed. The music preferences attribute includes the kinds of music the customer prefers (e.g., classical, rock, jazz). The fact that the attribute name is plural is a clue that many different preferences may be captured for each instance of a sale and that music preferences is a repeating attribute. This can be resolved by creating a new entity that contains preference information, and a relationship is added between CD purchase and preference. The new relationship is M:N, because a CD purchase can be associated with many music preferences and a music preference can be found on many CD purchases. See Figure 6A-3a for the current data model in 1NF.

Since we normally resolve M:N relationships as the ERD develops, we have done so now in Figure 6A-3b.

Note that a new intersection entity was inserted between CD Purchase and Preference to associate an instance of CD Purchase with specific instances of preference. Also, the intersection entity Purchased CD was inserted between CD Purchase and CD. This intersection entity associates a CD purchase instance with specific CD instances. The attribute ship date was moved to Purchased CD because the various CDs in a purchase may ship at different dates; therefore, this attribute describes a specific purchased CD, not the entire CD purchase.

Second Normal Form *Second normal form (2NF)* requires first that the data model is in 1NF and second that the data model leads to entities containing attributes that are *dependent* on the whole identifier. This means that the value of all attributes that serve as identifier can determine the value for all of the other attributes for an instance in an entity. Sometimes, nonidentifier attributes are dependent on only part of the identifier (i.e., *partial dependency*), and these attributes belong in another entity.

Figure 6A-4 shows the CD purchase data model placed in 2NF. Notice that originally, the *CD purchase* entity had three attributes that were used as identifiers: purchase date, customer last name, and customer first name. The problem was that some of the

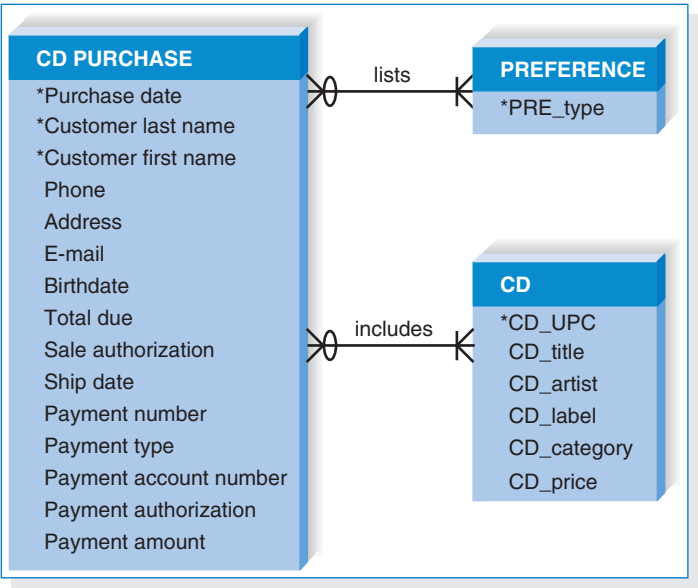


FIGURE 6A-3a
First Normal Form

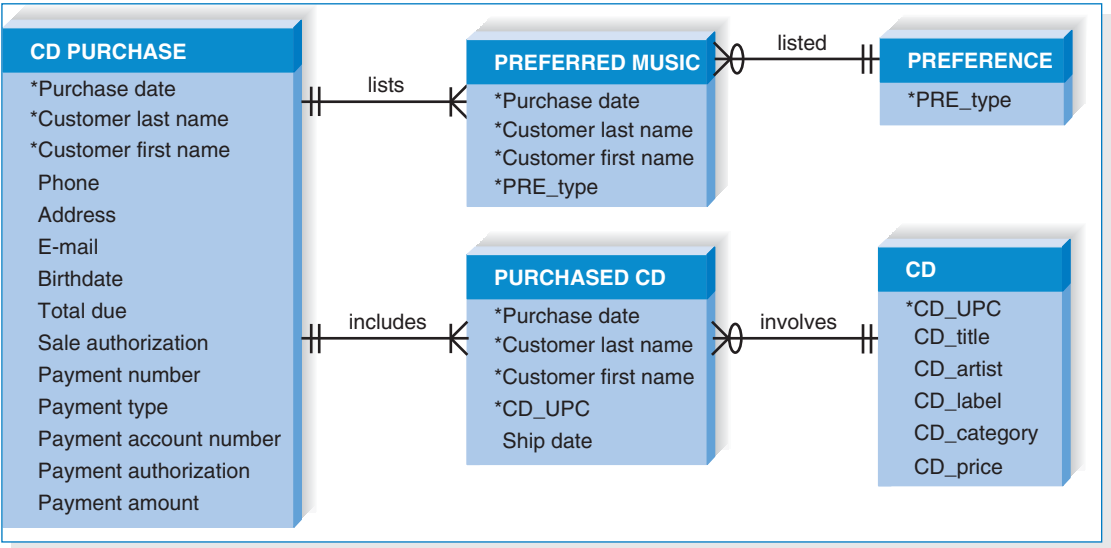


FIGURE 6A-3b
First Normal Form with M:N Relationships Resolved

attributes were dependent on the customer last name and first name, but had no dependency on purchase date. These attributes were those that describe a customer: phone, address, e-mail, and birth date. To resolve this problem, a new entity called *customer* was created, and the customer attributes were moved

into the new entity. A 1:N relationship exists between customer and CD purchase because a customer can purchase many CDs, but a CD purchase is associated with only one customer.

Remember that the customer last name and first name are still used in the *CD Purchase* entity—we know

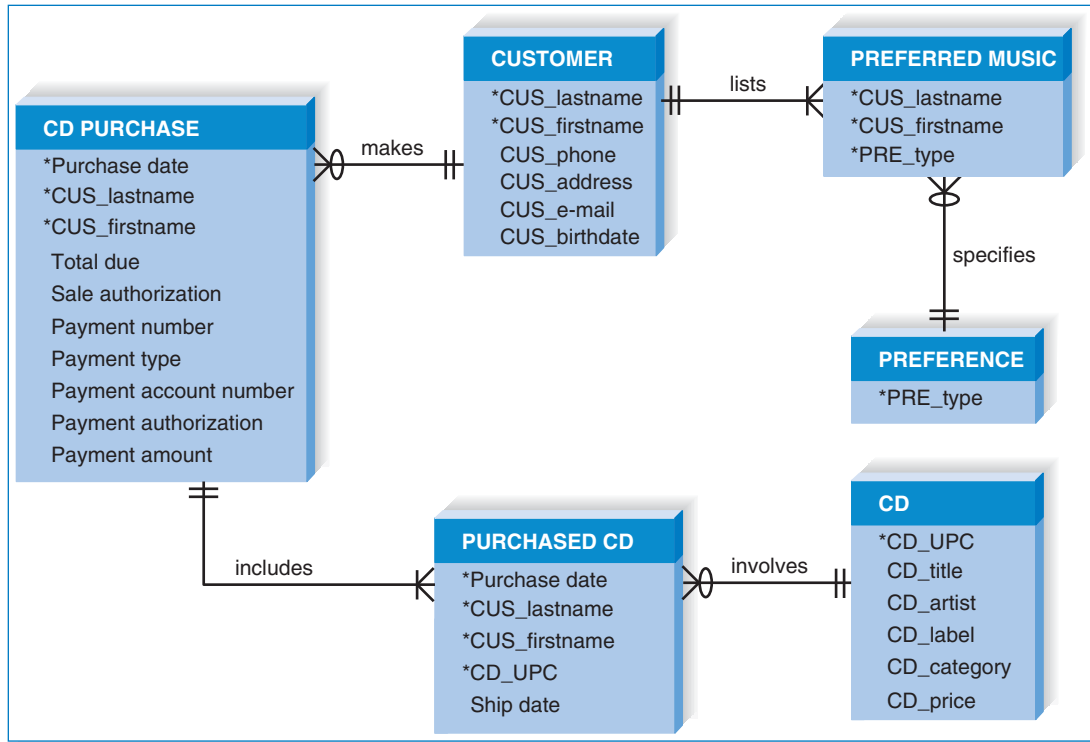


FIGURE 6A-4
Second Normal Form

this because of the identifying 1:N relationship between *customer* and *CD purchase*. The identifying relationship implies that the customer identifier (i.e., last name and first name) are used in CD Purchase as a part of its identifier.

Notice that we moved the relationship with Preferred Music to the new Customer entity. Logically, a preference should be associated with a customer, not a particular CD purchase.

Third Normal Form *Third normal form (3NF)* occurs when a model is in both 1NF and 2NF and when, in the resulting entities, none of the attributes is dependent on a nonidentifier attribute (i.e., *transitive dependency*). A violation of 3NF can be found in the CD Purchase entity in Figure 6A-4.

The problem with the CD Purchase entity is that there are attributes in the entity that depend on the payment number, not the CD purchase date and customer first and last names. The payment type, account number, authorization, and amount depend on the payment number, a nonidentifying attribute. Therefore, we create a separate *payment* entity and move the payment attributes to it. The 1:1 relationship assumes that there is one

payment for every CD purchase, and every CD purchase has one payment. Also, a payment is required for every CD purchase, and every CD purchase requires a payment.

Third normal form also addresses issues of *derived*, or calculated, *attributes*. By definition, derived attributes can be calculated from other attributes and do not need to be stored in the data model. As an example, a person's age would not be stored as an attribute if birthdate were stored, because, by knowing the birthdate and current date, we can always calculate the age. You might legitimately question whether total due should be stored as an attribute of CD purchase, since its value can be calculated by summing the prices of all the CDs included in the purchase. Like much of data modeling, there is no hard-and-fast rule about this. Many times, values such as total due are included to serve as a control value. In order to verify that no purchased CDs are omitted from the entire purchase, the total due is stored as an attribute of CD purchase, and the sum of the individual CD prices is also computed to ensure that they match. We will leave the total due in the data model and show the final ERD in 3NF in Figure 6A-5.

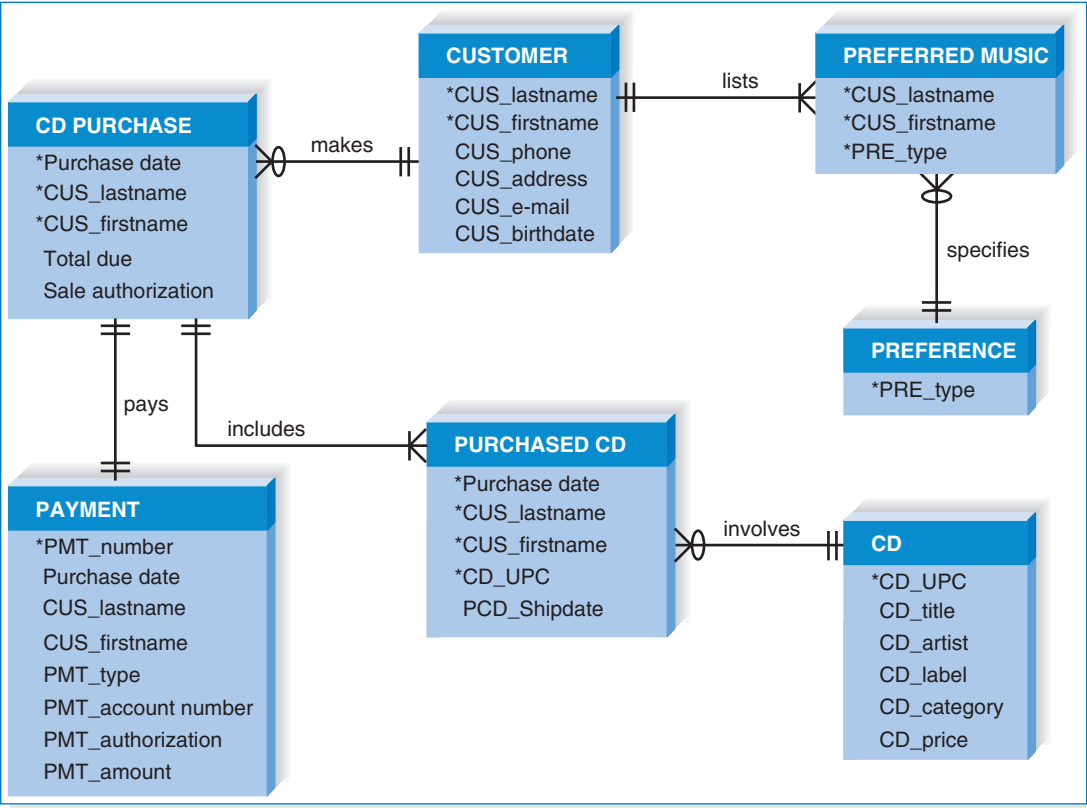


FIGURE 6A-5
Third Normal Form

YOUR

TURN

6A-1 NORMALIZING A STUDENT ACTIVITY FILE

Pretend that you have been asked to build a system that tracks student involvement in activities around campus. You have been given a file with information that needs to be imported into the system, and the file contains the following fields:

- Student Social Security number (identifier)
- Activity 1 code (identifier)
- Activity 1 description
- Activity 1 start date
- Activity 1 years with activity
- Activity 2 code
- Activity 2 description
- Activity 2 start date

- Activity 3 code
- Activity 3 description
- Activity 3 start date
- Activity 3 years with activity
- Student last name
- Student first name
- Student birthdate
- Student age
- Student advisor name
- Student advisor phone

Normalize the file. Show how the logical data model would change as you move from 1NF to 2NF to 3NF.