

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



IMPLEMENTATION

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.

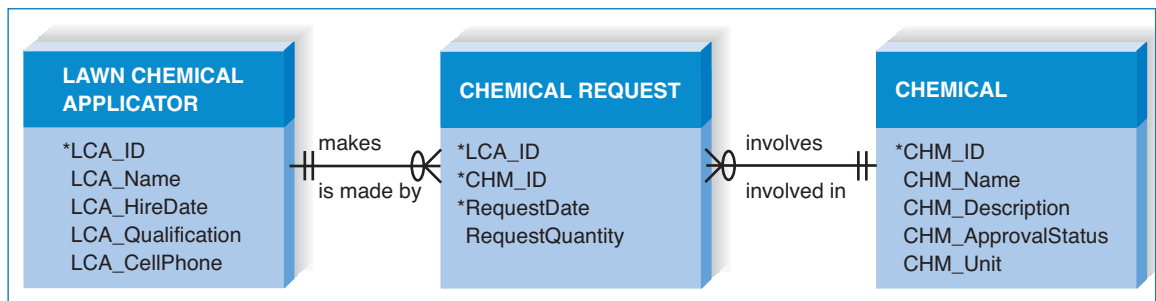


FIGURE 6-1
Chemical Request ERD

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

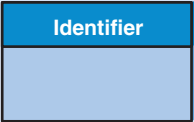

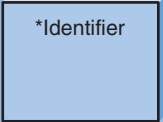
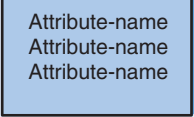
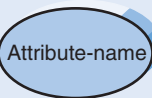
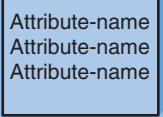
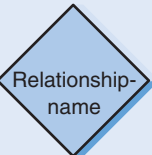
	IDEF1X	Chen	Crow's Foot
<p>An ENTITY</p> <ul style="list-style-type: none"> ✓ is a person, place, or thing. ✓ has a singular name spelled in all capital letters. ✓ has an identifier. ✓ should contain more than one instance of data. 	<p>ENTITY-NAME</p> 	<p>ENTITY-NAME</p> 	<p>ENTITY-NAME</p> 
<p>An ATTRIBUTE</p> <ul style="list-style-type: none"> ✓ 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. 	<p>ENTITY-NAME</p> 		<p>ENTITY-NAME</p> 
<p>A RELATIONSHIP</p> <ul style="list-style-type: none"> ✓ 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. 	<p><u>Relationship-name</u></p>		<p><u>Relationship-name</u></p>

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

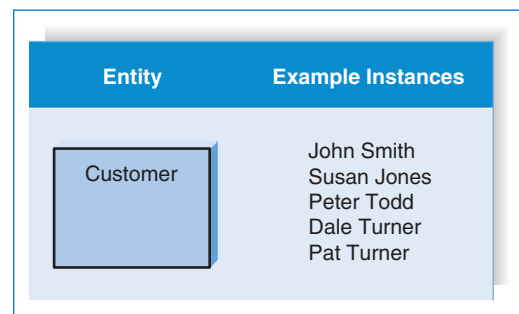


FIGURE 6-3
Entities and Instances

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.

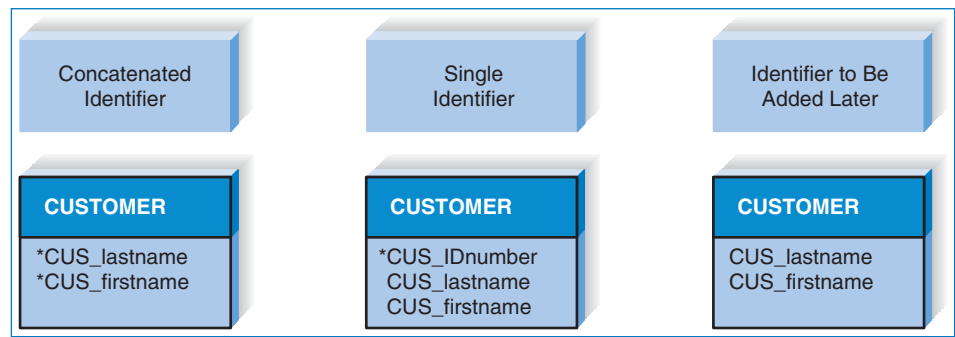


FIGURE 6-4
Choices for Identifiers

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.

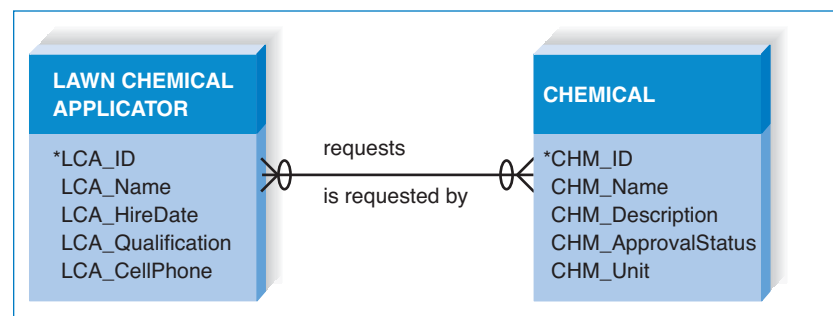


FIGURE 6-5
M:N Relationship

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.

QUESTIONS:

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

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

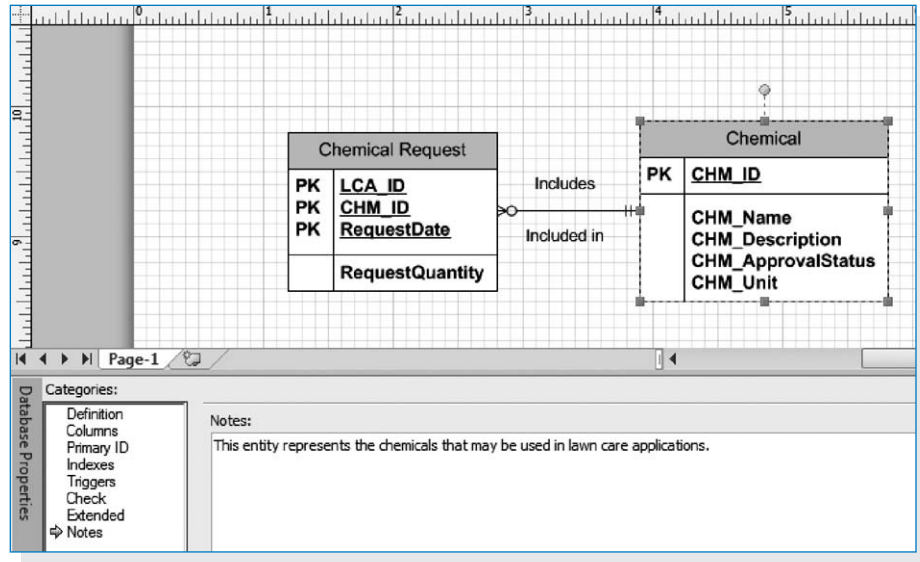


FIGURE 6-6
Data Dictionary Entry for Chemical Entity
(in Visio 2010)

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.

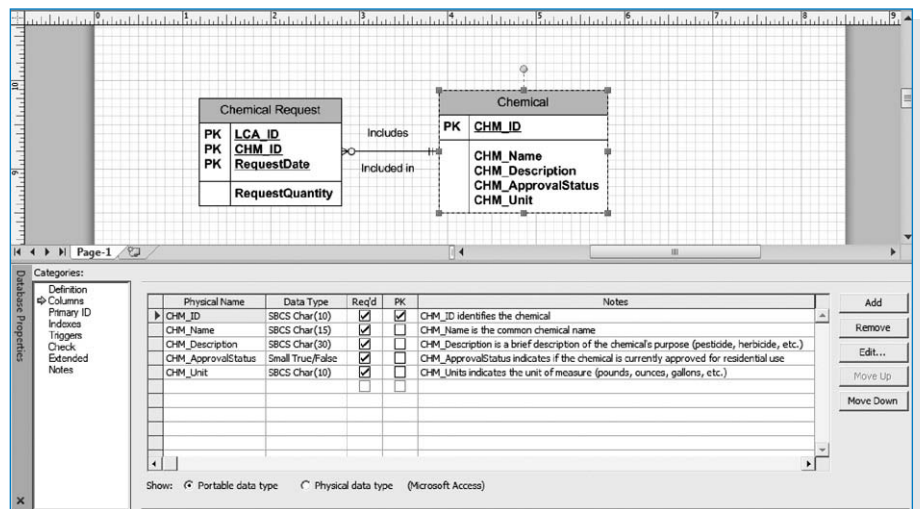


FIGURE 6-7
Data Dictionary Entry for Chemical
Attributes (in Visio 2010)

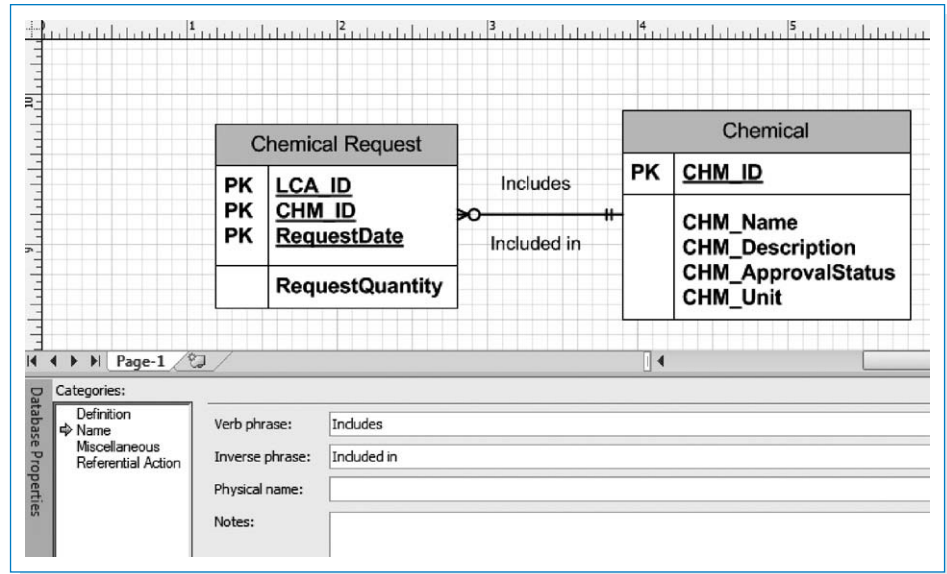


FIGURE 6-8
Data Dictionary Entry for a Relationship (in Visio 2010)

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

6-2 EVALUATE YOUR CASE TOOL

T U R N

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.

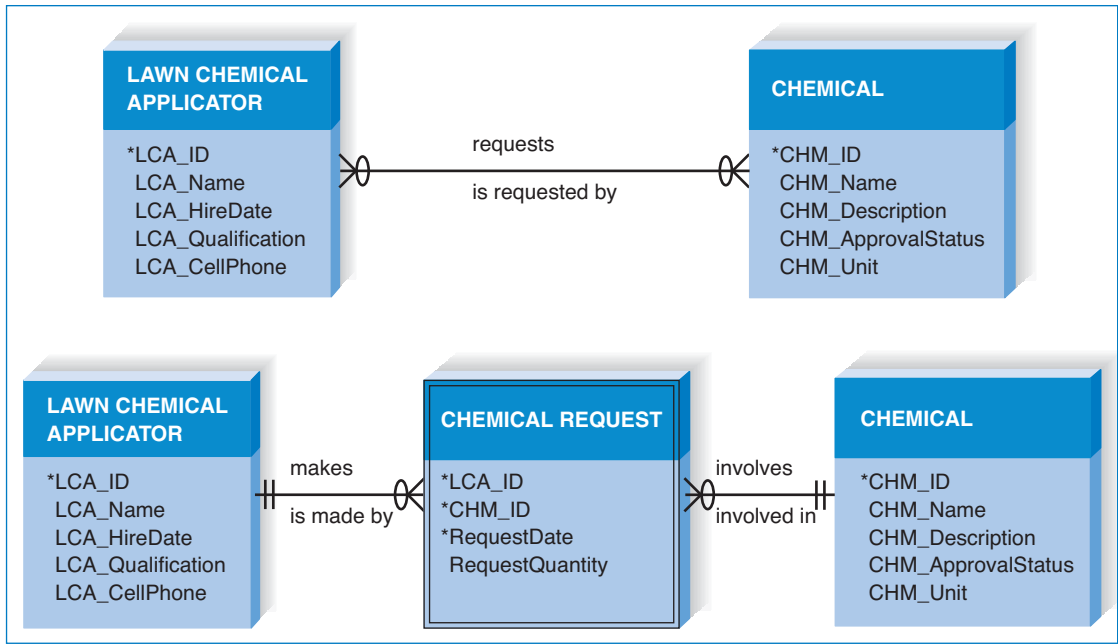


FIGURE 6-11
Resolving an M:N Relationship

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.

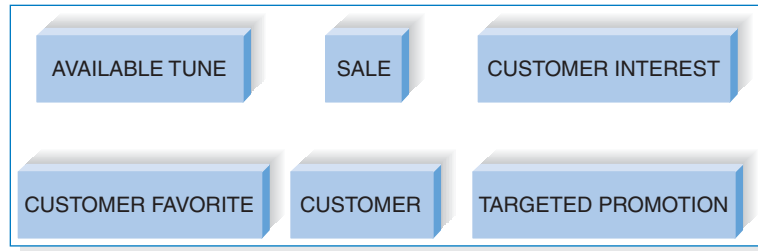


FIGURE 6-12
Entities for Tune Source ERD

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

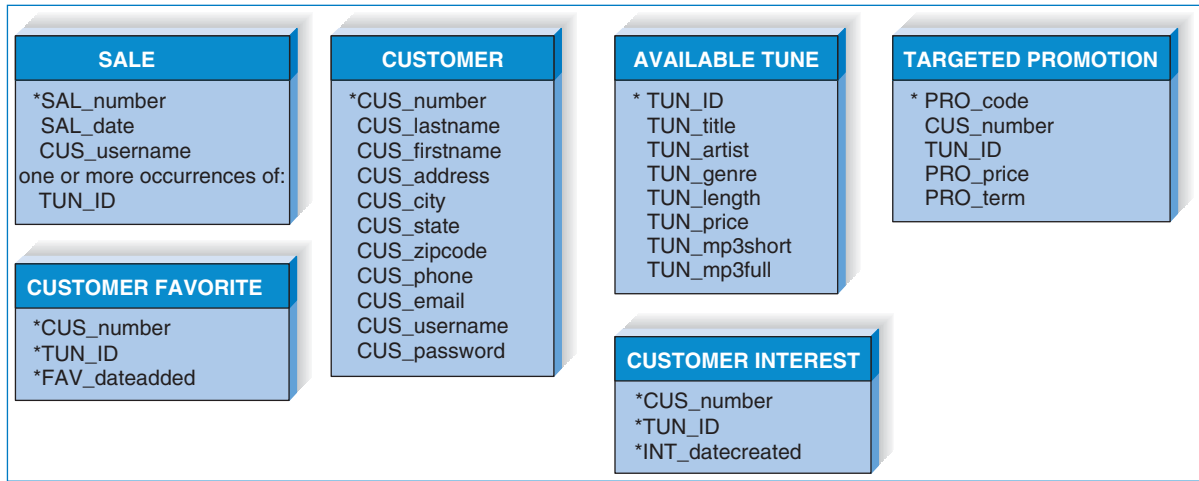


FIGURE 6-13
Attributes and Identifiers for Tune Source ERD

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

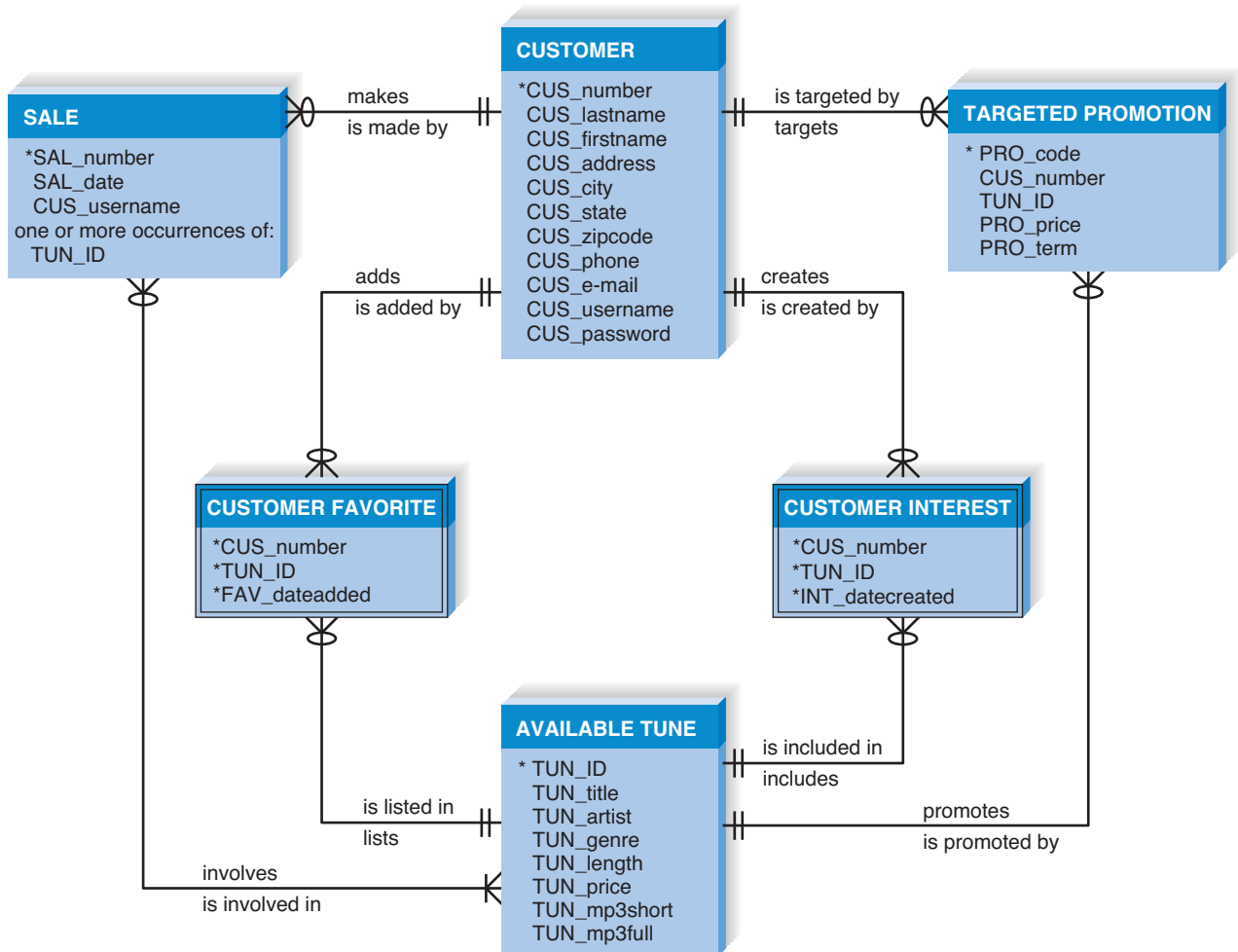


FIGURE 6-14
Relationships for Tune Source ERD

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 Wixon*

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?

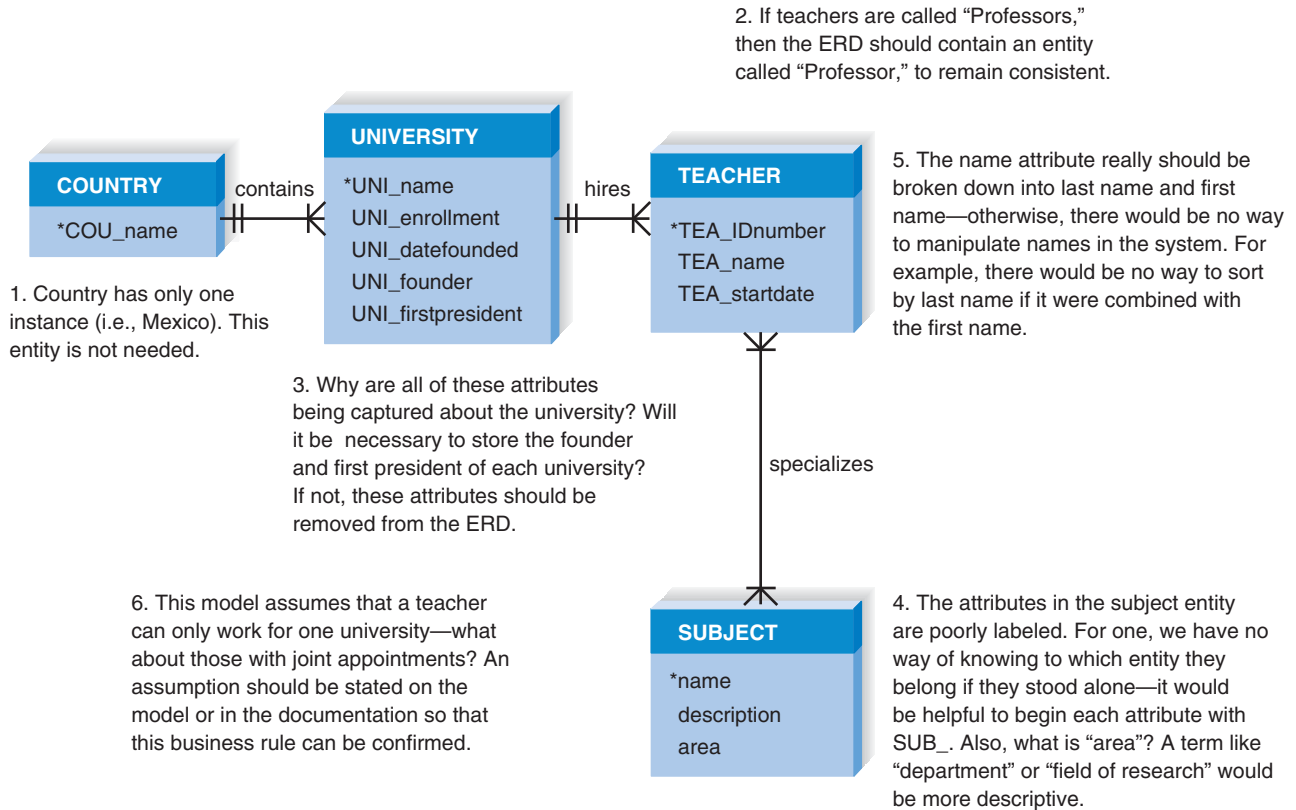


FIGURE 6-15
Data Modeling Guidelines Summary

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

6-6 INTERSECTION ENTITIES

T U R N

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

TURN

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

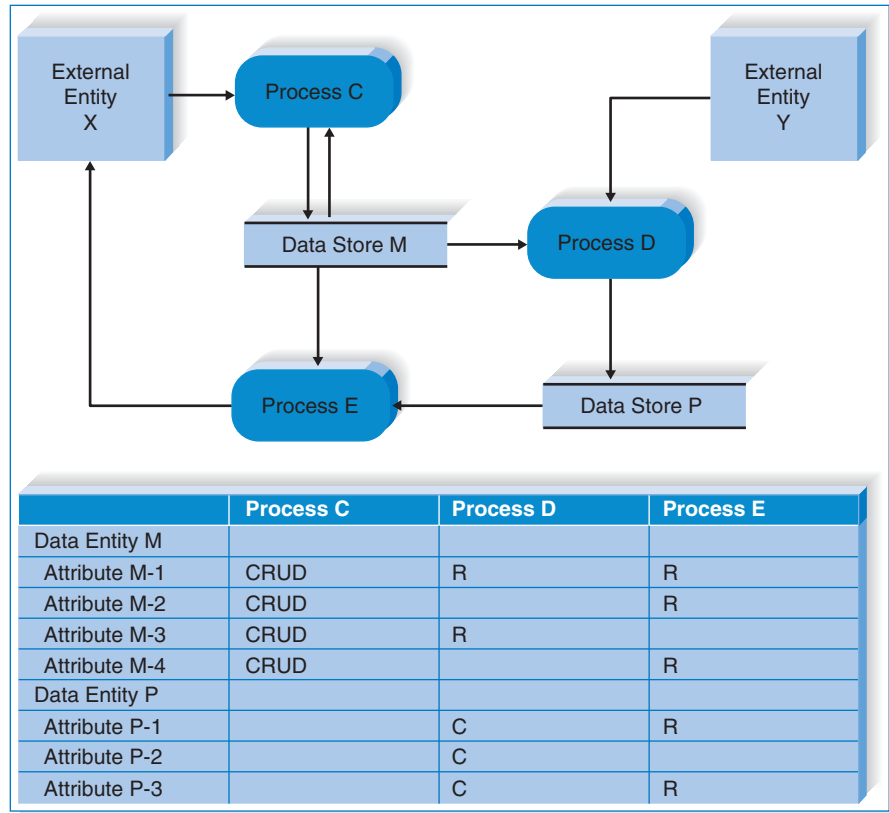


FIGURE 6-16
Partial Process Model and CRUD Matrix

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

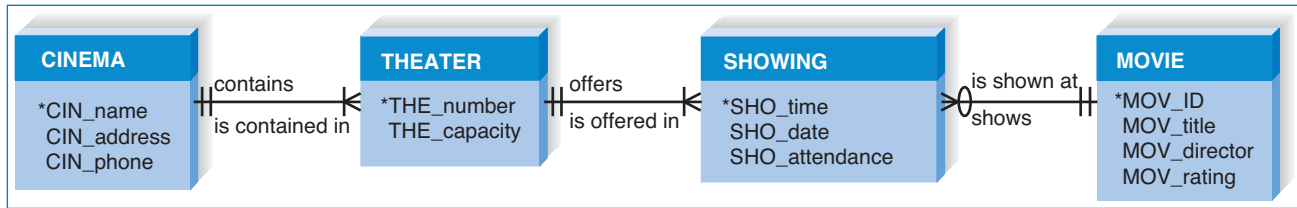
1:1 relationship	Derived attribute	Nonidentifying relationship
1:N relationship	Entity	Normalization
Assumption	Entity relationship diagram (ERD)	Parent entity
Attribute	First normal form (1NF)	Partial dependency
Balance	IDEF1X	Physical data model
Business rule	Identifier	Relationship
Cardinality	Identifying 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

- Provide three different options that are available for selecting an identifier for a student entity. What are the pros and cons of each option?
- What is the purpose of developing an identifier for an entity?
- What type of high-level business rule can be stated by an ERD? Give two examples.
- 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?
- 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?
- 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?
- 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?
- What is meant by a concatenated identifier?
- Describe, in terms a businessperson could understand, what are meant by the cardinality and modality of a relationship between two entities.
- What are metadata? Why are they important to system developers?
- What is an independent entity? What is a dependent entity? How are the two types of entities differentiated on the data model?
- Explain the distinction between identifying and nonidentifying relationships.
- What is the purpose of an intersection entity? How do you know whether one is needed in an ERD?
- Describe the three-step process of creating an intersection entity.
- Is an intersection entity dependent or independent? Explain your answer.
- What is the purpose of normalization?
- Describe the analysis that is applied to a data model in order to place it in first normal form (1NF).
- Describe the analysis that is applied to a data model in order to place it in second normal form (2NF).
- Describe the analysis that is applied to a data model in order to place it in third normal form (3NF).
- Describe how the data model and process model should be balanced against each other.
- What is a CRUD matrix? How does it relate to process models and data models?

EXERCISES

- 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)
- 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)
- 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.
- Draw an entity relationship diagram (ERD) for the following situations:
 - 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.

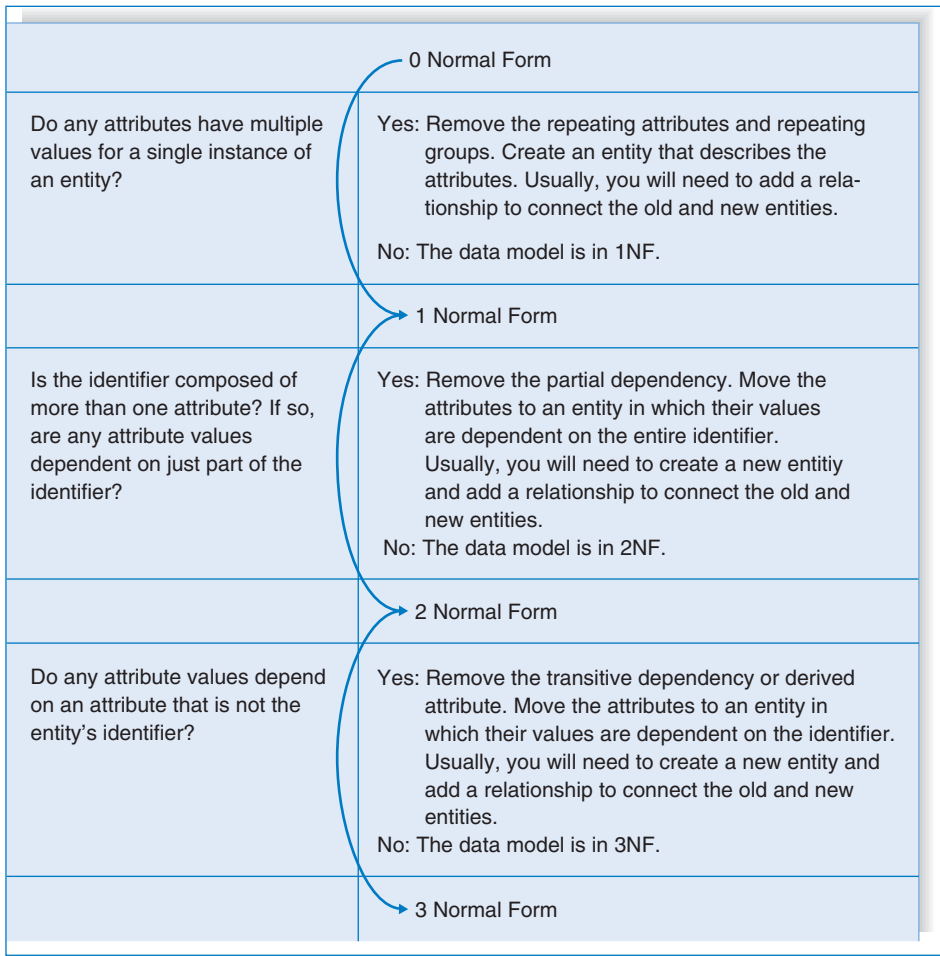


FIGURE 6A-1
Normalization Steps

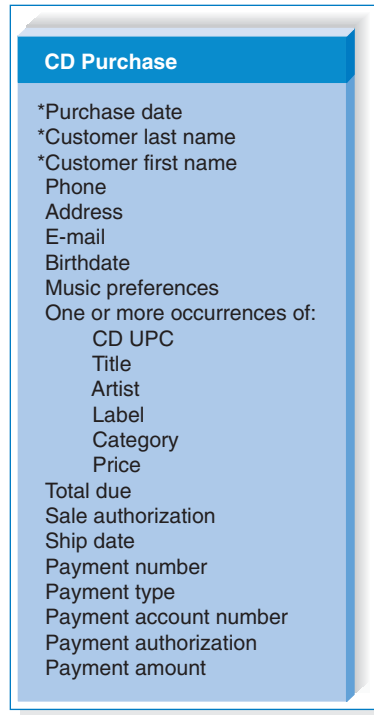


FIGURE 6A-2
Initial CD Sales System File

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

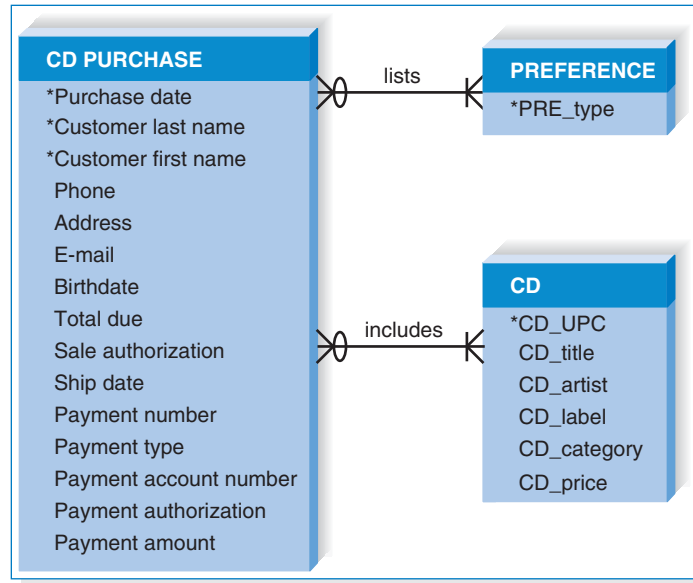


FIGURE 6A-3a
First Normal Form

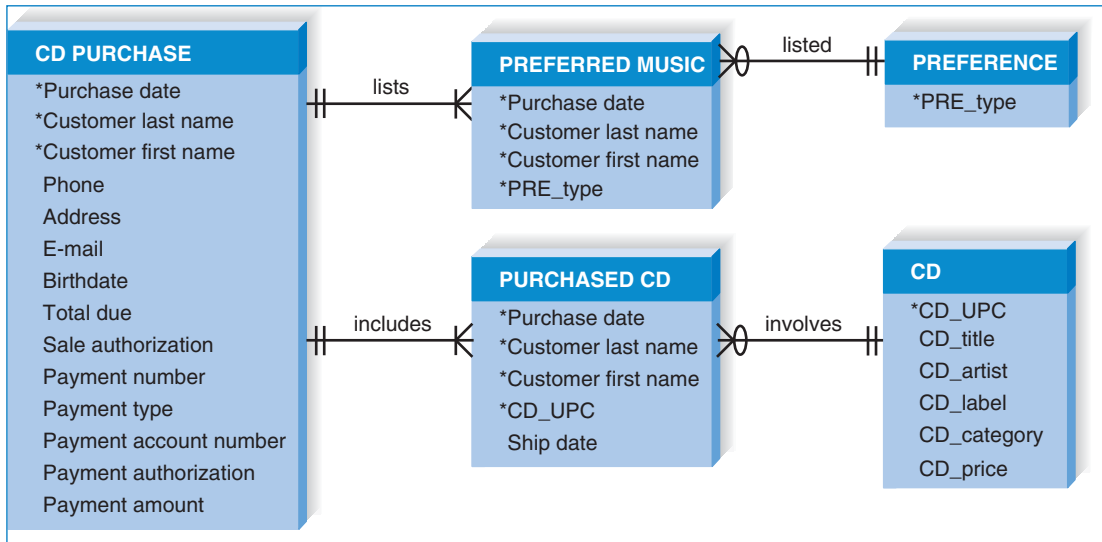


FIGURE 6A-3b
First Normal Form with M:N Relationships Resolved

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

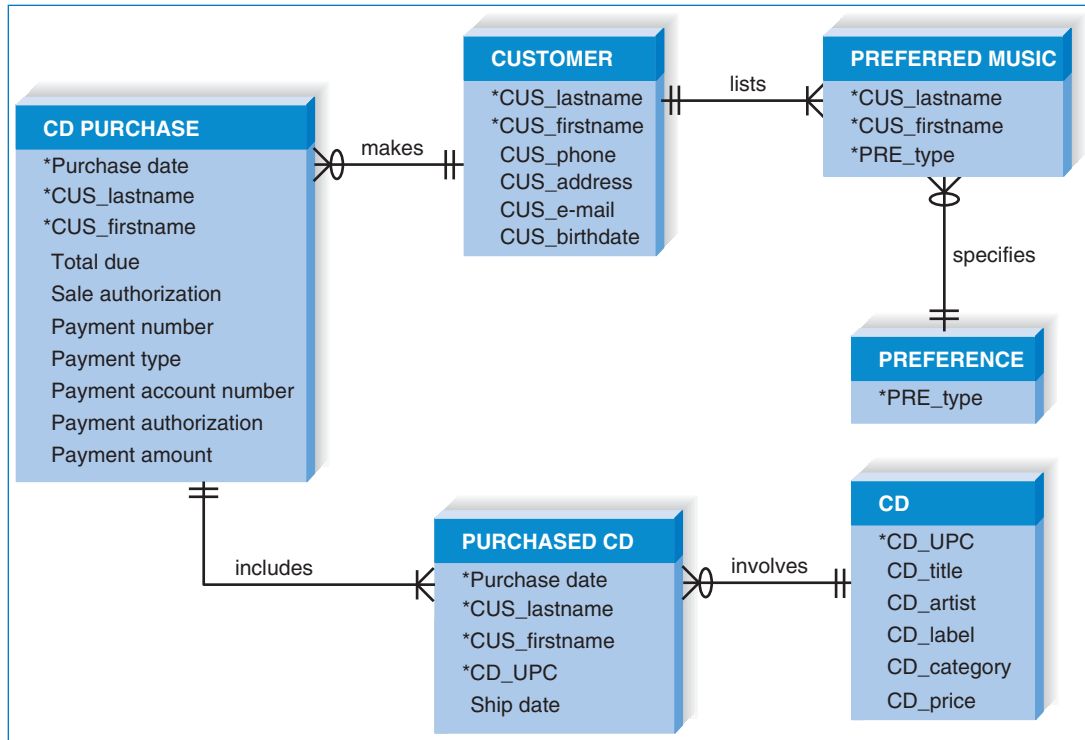


FIGURE 6A-4
Second Normal Form

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.

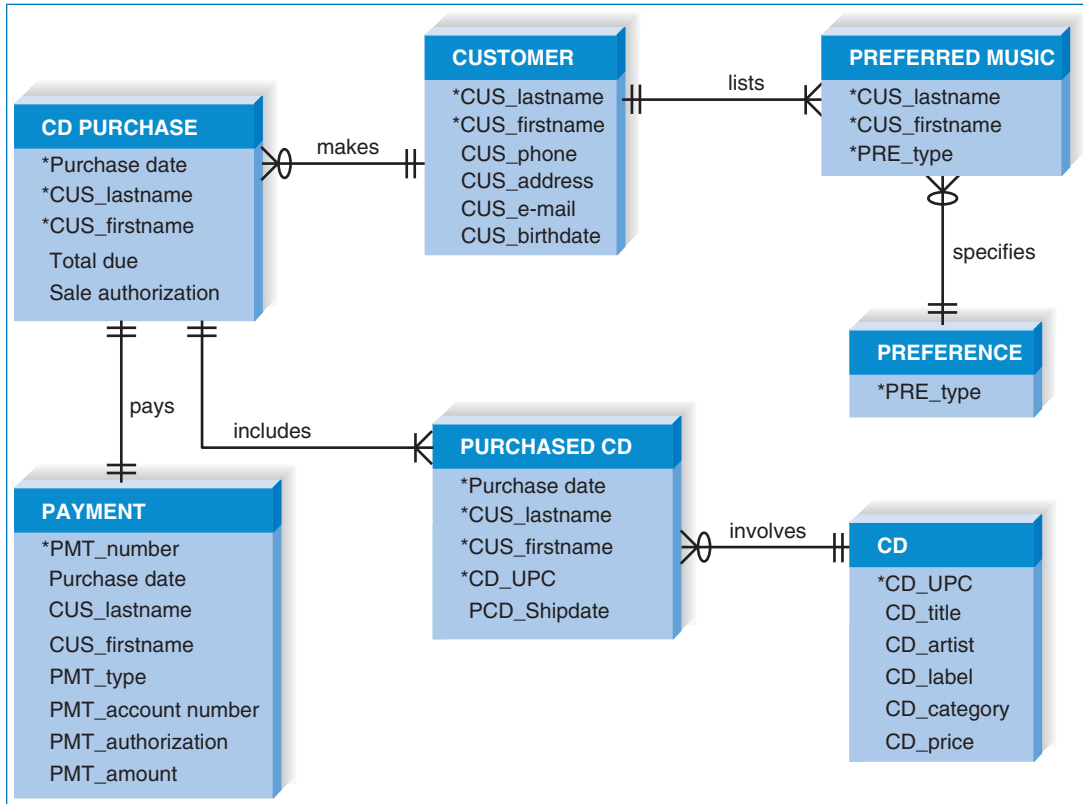


FIGURE 6A-5
Third Normal Form

YOUR

6A-1 NORMALIZING A STUDENT ACTIVITY FILE

TURN

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.