So far, you have been introduced to the fundamental concepts and techniques of relational database management. In Chapter 1, you learned that databases consist of related tables, and you learned the major components of a database system. Chapter 2 introduced you to the relational model, and you learned the basic ideas of functional dependencies and normalization. In Chapter 3, you learned how to use basic SQL statements to process a database.

All of this material gives you a background for understanding the nature of database management and the basic tools and techniques. However, you do not yet know how to apply all of this technology to solve a business problem. Imagine, for example, that you walk into a small business—say a bookshop—and are asked to build a database to support a frequent buyer program. How would you proceed? So far, we have assumed that the database design already exists. How would you go about creating the design of the database? Furthermore, once the database exists, what tasks need to be done to manage it over time?

The next three chapters address these important topics. We begin Chapter 4 with an overview of the database design process, and then we describe data...
modeling—a technique for representing database requirements. In Chapter 5, you will learn how to transform a data model into a relational database design. Finally, in Chapter 6, you will learn about database management. Here, you also will be introduced to many of the problems that occur when a database is concurrently processed by more than one user. Finally, Chapter 7 will conclude the book by surveying important advanced database concepts. After completing these chapters, you will have surveyed all of the basic database technology. This knowledge will give you the necessary background to learn more in the area of your interests or job requirements. For example, you will be able to learn more about SQL, or to learn to use a particular DBMS product such as Oracle or SQL Server, or to learn to publish databases using Internet technology.
The process of developing a database system consists of three major stages: requirements, design, and implementation. During the requirements stage, system users are interviewed and sample forms, reports, queries, and descriptions of update activities are obtained. These system requirements are used to create a data model, which is a representation of the content, relationships, and
constraints of the data needed to support the requirements. Often prototypes, or working demonstrations of selected portions of the future system, are created during the requirements phase. Such prototypes are used to obtain feedback from the system users.

During the design stage, the data model is transformed into a database design. Such a design consists of tables, relationships, and constraints. The design includes the table names and the names of all table columns. The design also includes the data types and properties of the columns, as well as a description of primary and foreign keys. Data constraints consist of limits on data values (e.g., Part Numbers are seven-digit numbers starting with the number three), referential integrity constraints, and business rules. An example of a business rule for a manufacturing company is that every purchased part will have a quotation from at least two suppliers.

The last stage of database development is implementation. During this stage, the database is constructed and filled with data; queries, forms, and reports are created; application programs are written; and all of these are tested. Finally, during this stage users are trained, documentation is written, and the system is installed for use.

In this chapter, we will consider the requirements stage in general and data modeling in particular. You will learn about an important tool, the entity-relationship data model, and you will learn how to apply this tool to represent the data requirements for a small business. Our goal here is to focus only on the database aspects of information systems development. The design and development of other components of an information system are outside the scope of this text; they are the subject matter of a systems development class.

THE REQUIREMENTS STAGE

Sources of user requirements are listed in Figure 4-1. As you will learn in your systems development class, the general practice is to identify the users of the new information system and to interview them. During the interviews, examples of existing forms, reports, and queries are obtained. In addition, the users are asked about the need for changes to existing forms, reports, and queries, and also about the need for new forms, reports, and queries.

Use cases are descriptions of the ways users will employ the features and functions of the new information system. A use case consists of a description of the roles users will play when utilizing the new system, together with descriptions of scenarios of activities. Inputs provided to the system and outputs generated by the system are defined.
The system requirements described in the prior section, although necessary and important as a first step, are not sufficient for designing a database. In addition, the requirements must be transformed into a data model. When writing application programs, program logic must first be documented in flowcharts or object diagrams; this is also the case with a database—data requirements must first be documented in a data model.

A number of different techniques can be used to create data models. By far the most popular is the entity-relationship (E-R) model\(^1\), so this book will focus on it. The E-R model comes in several flavors, or variations. Here we will consider two of them. We begin with the traditional E-R model, which was the first E-R version to achieve prominence. The symbols of the traditional E-R model are understood by all database professionals.

A second version of the E-R model is one that was incorporated into the Unified Modeling Language (UML)\(^2\). UML uses different symbols and notation from the traditional E-R model, and we will introduce that notation so you will be able to interpret E-R models in UML diagrams.

The important elements of the E-R model are entities, attributes, identifiers, and relationships. We will consider each of these in turn.

**Entities**

An entity is something that users want to track. Examples of entities are CUSTOMER John Doe, PURCHASE 12345, PRODUCT A4200, SALES-ORDER 1000, SALESPERSON John Smith, and SHIPMENT 123400. Notice that in this list an entity is a specific thing or an instance of something.


All of the instances of an entity of a given type are grouped into entity classes. Thus, the PRODUCT entity class is the collection of all PRODUCT entities. In this text, entity classes are printed in capital letters. It is important to understand the differences between an entity class and an entity instance. An entity class is a collection of entities and is described by the structure or format of the entities in that class. An instance of an entity class is the representation of a particular entity, such as CUSTOMER 12345; it is described by the values of attributes of the entity.

An entity class usually includes many instances of an entity. For example, the class ITEM has many instances—one for each item stored in the database. An entity class and two of its instances are shown in Figure 4-2.

When developing a data model, the developers analyze the forms, reports, queries, and other system requirements. Entities are usually the subject of one or more forms or reports, or are a major section in one or more forms or reports. For example, a form entitled PRODUCT Data Entry Form indicates the likelihood of an entity class called PRODUCT. Similarly, a report entitled CUSTOMER PURCHASE Summary indicates that most likely the business has CUSTOMER and PURCHASE entities.

Attributes

Entities have attributes that describe the entity’s characteristics. Examples of attributes include EmployeeName, DateOfHire, and JobSkillCode. In this text, attributes are printed in uppercase and lowercase letters. The E-R model assumes that all instances of a given entity class have the same attributes.

Attributes have a data type and properties that are determined from the requirements. Typical data types are character, numeric, date, currency, and the like. Properties specify whether the attribute is required, whether it has a default value, whether its value has limits, and any other constraint.

Identifiers

Entity instances have identifiers, which are attributes that name, or identify, entity instances. For example, EMPLOYEE instances could be identified by SocialSecurityNumber, by EmployeeNumber, or by EmployeeName. EMPLOYEE instances are not likely to be identified by attributes such as Salary or DateOfHire because these attributes normally are not used in a naming role. Similarly, CUSTOMER instances could be identified by

<table>
<thead>
<tr>
<th>ITEM</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>ItemNumber</td>
<td></td>
</tr>
<tr>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>Cost</td>
<td></td>
</tr>
<tr>
<td>ListPrice</td>
<td></td>
</tr>
<tr>
<td>QuantityOnHand</td>
<td></td>
</tr>
</tbody>
</table>

Entity Class

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1100</td>
<td>2000</td>
</tr>
<tr>
<td>100 amp panel</td>
<td>Door handle set</td>
</tr>
<tr>
<td>$127.50</td>
<td>$52.50</td>
</tr>
<tr>
<td>$170.00</td>
<td>$39.38</td>
</tr>
<tr>
<td>14</td>
<td>0</td>
</tr>
</tbody>
</table>

Two Entity Instances
CustomerNumber or CustomerName, and SALES-ORDER instances could be identified by OrderNumber.

The identifier of an entity instance consists of one or more of the entity’s attributes. Identifiers that consist of two or more attributes are called composite identifiers. Examples are (AreaCode, LocalNumber), (ProjectName, TaskName), and (FirstName, LastName, PhoneExtension).

An identifier may be either unique or nonunique. If it is unique, its value will identify one, and only one, entity instance. If it is nonunique, the value will identify a set of instances. EmployeeNumber is normally a unique identifier, but EmployeeName is most likely a nonunique identifier (more than one John Smith might be employed by the company, for example).

As you can tell from these definitions, identifiers are similar to keys in the relational model, but with two important differences. First, an identifier is a logical concept; it is one or more attributes that users think of as a name of the entity. Such identifiers might or might not be represented as keys in the database design. Second, primary and candidate keys must be unique, whereas identifiers might or might not be unique.

**Relationships**

Entities can be associated with one another in relationships. The E-R model contains relationship classes and relationship instances. Relationship classes are associations among entity classes, and relationship instances are associations among entity instances. According to the original specification of the E-R model, relationships can have attributes; however, in modern practice, only entities have attributes.

A relationship class can involve many entity classes. The number of entity classes in the relationship is known as the degree of the relationship. In Figure 4-3(a), the SUPPLIER-QUOTATION relationship is of degree 2 because it involves two entity classes, SUPPLIER and QUOTATION. The PARENT relationship in Figure 4-3(b) is of degree 3, because it involves three entity classes: MOTHER, FATHER, and CHILD. Relationships of degree 2, which are called binary relationships, are the most common.

**Three Types of Binary Relationships**  Figure 4-4 shows the three types of binary relationships. In a 1:1 (read “one-to-one”) relationship, a single entity instance of one type is related to a single entity instance of another type. In Figure 4-4(a), the LOCKER-ASSIGNMENT relationship associates a single EMPLOYEE with a single LOCKER. According to this diagram, no employee has more than one locker assigned, and no locker is assigned to more than one employee.

Figure 4-4(b) shows the second type of binary relationship, 1:N (read “one to N” or “one to many”). In this relationship, which is called the ITEM-QUOTE relationship, a single instance of ITEM relates to many instances of QUOTATION. According to this sketch, an item has many quotations, but a quotation has only one item.

Think of the diamond as representing the relationship. The position of the 1 indicates that the relationship has one ITEM; the position of the N indicates that it also has...
many QUOTATION entities. Thus, each instance of the relationship consists of one ITEM and many QUOTATIONS. If the 1 and the N were reversed and the relationship were written N:1, each instance of the relationship would have many ITEMS and one QUOTATION. However, this is not the case.

Figure 4-4(c) shows the third type of binary relationship, N:M (read “N to M” or “many to many”). This relationship is named ITEM-SOURCE, and it relates instances of ITEM to instances of SUPPLIER. An item can be supplied by many suppliers, and a supplier can supply many items.

The numbers inside the relationship diamond show the maximum number of entities that can occur on one side of the relationship. Such constraints are called the relationship's maximum cardinality. The relationship in Figure 4-4(b), for example, is said to have a maximum cardinality of 1:N. However, the cardinalities are not restricted to the values shown here. It is possible, for example, for the maximum cardinality to be other than 1 and N. The relationship between BASKETBALL-TEAM and PLAYER, for example, could be 1:5, indicating that a basketball team has at most five players.

**Recursive Relationships** It is possible for an entity to have a relationship to itself. Figure 4-4(d) shows a CUSTOMER entity in which one customer can refer many other customers. As with binary relationships, recursive relationships can be 1:1, 1:N (shown in Figure 4-4[d]), and N:M. We will discuss each of these three types further in Chapter 5.

**Entity-Relationship Diagrams**

The sketches in Figure 4-4 are called entity-relationship diagrams. Such diagrams are standardized, but only loosely. According to this standard, entity classes are shown by rectangles, relationships are shown by diamonds, and the maximum cardinality of the relationship is shown inside the diamond. The name of the entity is shown inside the rectangle, and the name of the relationship is shown near the diamond. In some cases, the attributes of the entity are listed in the entity rectangle. You will see examples of this later in this chapter.
As stated, the maximum cardinality indicates the maximum number of entities that can be involved in a relationship. The diagrams do not indicate the minimum. For example, Figure 4-4(b) shows that a quotation is related, at maximum, to one item, but it does not show whether a quotation must be related to an item.

Minimum cardinality can be shown in several different ways. One way, illustrated in Figure 4-5, is to place a hash mark across the relationship line to indicate that an entity must exist in the relationship, and to place an oval across the relationship line to indicate that an entity might or might not be in the relationship. Accordingly, Figure 4-5 shows that an ITEM must have a relationship with at least one SUPPLIER, but that a SUPPLIER is not required to have a relationship with an ITEM. The complete relationship restrictions are that an ITEM has a minimum cardinality of one and a maximum cardinality of many SUPPLIER entities. A SUPPLIER has a minimum cardinality of zero and a maximum cardinality of many ITEM entities.

Weak and ID-Dependent Entities

The E-R model defines a special type of entity called a weak entity. Such entities are those that cannot exist in a database unless another type of entity also exists in the database. An entity that is not weak is called a strong entity.

To understand weak entities, consider a human resource database with EMPLOYEE and DEPENDENT entity classes. Suppose the business has a rule that an EMPLOYEE instance can exist without having a relationship to any DEPENDENT entity, but a DEPENDENT entity cannot exist without having a relationship to a particular EMPLOYEE entity. In such a case, DEPENDENT is a weak entity. This means that DEPENDENT data can be stored in the database only if the DEPENDENT data have a relationship with an EMPLOYEE entity.

As shown in Figure 4-6(a), weak entities are signified by rounding the corners of the entity rectangle. In addition, the relationship on which the entity depends for its existence is shown in a diamond with rounded corners. Alternatively, in some E-R diagrams (not shown here) weak entities are depicted by using a double line for the boundary of the weak entity rectangle and double diamonds for the relationship on which the entity depends.

The E-R model includes a special type of weak entity called an ID-dependent entity. Such an entity is one in which the identifier of one entity includes the identifier of another entity. Consider the entities BUILDING and APARTMENT. Suppose the identifier of BUILDING is BuildingName, and the identifier of APARTMENT is the composite identifier (BuildingName, ApartmentNumber). Because the identifier of APARTMENT contains the identifier of BUILDING (BuildingName), APARTMENT is ID-dependent on BUILDING. Contrast Figure 4-6(b) with Figure 4-6(a). Another way to think of this is that logically and physically, an APARTMENT simply cannot exist unless a BUILDING exists.
ID-dependent entities are common. Another example is the entity VERSION in the relationship between PRODUCT and VERSION, where PRODUCT is a software product and VERSION is a release of that software product. The identifier of PRODUCT is ProductName, and the identifier of VERSION is (ProductName, ReleaseNumber). A third example is EDITION in the relationship between TEXTBOOK and EDITION. The identifier of TEXTBOOK is Title, and the identifier of EDITION is (Title, EditionNumber).

Unfortunately, an ambiguity is hidden in the definition of weak entity, and this ambiguity is interpreted differently by different database designers (as well as different textbook authors). The ambiguity is this: In a strict sense, if a weak entity is defined as any entity whose presence in the database depends on another entity, then any entity that participates in a relationship having a minimum cardinality of one to a second entity is a weak entity. Thus, in an academic database, if a STUDENT must have an ADVISER, then STUDENT is a weak entity, because a STUDENT entity cannot be stored without an ADVISER.

This interpretation seems too broad to some people. A STUDENT is not physically dependent on an ADVISER (unlike an APARTMENT to a BUILDING), and a STUDENT is not logically dependent on an ADVISER (despite how it might appear to either the student or the adviser); therefore, STUDENT should be considered a strong entity.

To avoid such situations, some people interpret the definition of weak entity more narrowly. To be a weak entity, an entity must logically depend on another entity. According to this definition, DEPENDENT and APARTMENT are weak entities, but STUDENT is not. A DEPENDENT cannot be a dependent unless it has someone to depend on, and an APARTMENT cannot exist without a BUILDING in which to be located. However, a STUDENT can logically exist without an ADVISER, even if a business rule requires it.

To illustrate this interpretation, consider several examples. Suppose a data model includes the relationship between an ORDER and a SALESPERSON, as shown in Figure 4-7(a). Although we might say that an ORDER must have a SALESPERSON, it does not necessarily require one for its existence. (The ORDER could be a cash sale in which the salesperson is not recorded.) Hence, the minimum cardinality of one arises from a business rule, not from logical necessity. Thus, ORDER requires a SALESPERSON but is not existence-dependent on it, and ORDER is thus a strong entity.

Now consider the relationship of PATIENT and PRESCRIPTION in Figure 4-7(b). Here, a PRESCRIPTION cannot logically exist without a PATIENT. Hence, not only is the minimum cardinality one, but also the PRESCRIPTION is existence-dependent on PATIENT. PRESCRIPTION is thus a weak entity.

Finally, consider ASSIGNMENT in Figure 4-7(c), where the identifier of ASSIGNMENT contains the identifier of PROJECT. Here, not only does ASSIGNMENT

---

**FIGURE 4-7**

Examples of Required Entities

![Diagram](image-url)

(a) ORDER is a strong entity

(b) PRESCRIPTION is a weak entity

(c) ASSIGNMENT is ID-dependent
have a minimum cardinality of one, and not only is ASSIGNMENT existence-dependent on PROJECT, but ASSIGNMENT is also ID-dependent on PROJECT, because its identifier includes the key of another entity. Thus, ASSIGNMENT is a weak entity.

This text defines weak entities as those that logically depend on another entity. Hence, not all entities that have a minimum cardinality of one in relationship to another entity are weak. Only those that are logically dependent are weak. This definition implies that all ID-dependent entities are weak. In addition, every weak entity has a minimum cardinality of one on the entity on which it depends, but every entity that has a minimum cardinality of one is not necessarily weak.

**THE E-R MODEL IN UML**

The Unified Modeling Language (UML) is a set of diagrams and structures for modeling and designing object-oriented programs (OOP) and applications. UML is not a methodology for developing OOP programs but rather is a set of tools that support the development of such programs. UML has received prominence via the Object Management Group, an organization that has been developing OOP models, technologies, and standards since the 1980s. UML also has begun to receive widespread use among OOP practitioners. UML is the basis of the object-oriented design tools from Rational Systems, recently purchased by IBM.

Because UML concerns program and application development, it is a subject for a course on systems development and is of limited concern to us. However, you may encounter UML-style E-R diagrams, so you should be familiar with their style. Realize that when it comes to database design, they are treated just the same as traditional E-R diagrams.

**UML Entities and Relationships**

Figure 4-8 shows the UML representation of the designs in Figure 4-4. Each entity is represented by an entity class, which is shown as a rectangle with three segments. The top segment shows the name of the entity and other data that pertain to OOP and is not important for our purposes. The second segment lists the names of the attributes in the entity, and the third documents constraints and lists methods (program procedures) that belong to the entity. Here, we will show constraints but will not be concerned with the OOP methods.

Relationships are shown with a line between the two entities. Cardinalities are represented in the format \( x.y \), where \( x \) is the minimum required and \( y \) is the maximum allowed. Thus, \( 0..1 \) means that no entity is required and at most one is allowed. An asterisk represents an unlimited number. Thus, \( 1..* \) means that one is required and an unlimited number is allowed. Examine Figures 4-8(a) through (c) for examples of 1:1, 1:N, and N:M maximum-cardinality relationships.

Figure 4-8 illustrates a subtle but important point. With E-R diagrams, whether traditional or UML style, foreign keys are never shown. The relationship is represented only by the diamond or line between the entities. The placement of foreign keys is considered a database design task that is done only when the tables are designed, and not before. You will see one reason for this in Chapter 5, when we show the relational representation of N:M relationships.

**UML Representation of Weak Entities**

Figure 4-9 shows the UML representation of weak entities. A filled-in diamond is placed on the line going to the weak entity’s parents (the entity on which the weak entity depends). In Figure 4-9(a), PRESCRIPTION is the weak entity and PATIENT is the parent entity. Every weak entity has a parent, and thus the cardinality on the parent side of the weak relationship is always 1..1. Because this is so, the cardinality on the parent entity is shown simply as 1. Figure 4-9(a) shows a weak entity that is not ID-dependent. This is denoted by the expression <nonidentifying> on the PATIENT-PRESCRIPTION
relationship. Figure 4-9(b) shows a weak entity that is ID-dependent. That is denoted with the label <identifying>.

With this background, you should be able to interpret UML-style E-R diagrams. However, for the rest of this text we will use the more traditional, and common, notation.

**DEVELOPING AN EXAMPLE E-R DIAGRAM**

The best way to gain proficiency with data modeling is to do it. In this section, we will examine a set of documents used by a small business and create a data model from those documents. After you have read this section, you should practice creating data models with one or more of the projects at the end of the chapter.

**Heather Sweeney Designs**

Heather Sweeney is an interior designer who specializes in home kitchen design. She offers a variety of seminars at home shows, kitchen and appliance stores, and other public locations. The seminars are free; she offers them as a way of building her customer base. She earns revenue by selling books and videos that instruct people on kitchen design. She also offers custom-design consulting services.

After someone attends a seminar, Heather wants to leave no stone unturned in attempting to sell that person one of her products or services. Accordingly, she would like
to develop a database to keep track of customers, the seminars they have attended, the contacts she has made with them, and the purchases they have made. She wants to use this database to continue to contact her customers and offer them products and services.

The Seminar Customer List

Figure 4-10 shows the seminar customer list that Heather or her assistant fills out at seminars. It includes basic data about the seminar as well as the name, phone, and e-mail address of all the attendees at the seminar. If we examine this list in terms of a data model, two potential entities are found: SEMINAR and CUSTOMER.
From the form in Figure 4-10, we can conclude that a SEMINAR relates to many CUSTOMERS, and we can make the initial E-R diagram shown in Figure 4-11(a). However, from this single document a number of facts cannot be determined. The missing facts are denoted with question marks.

Having missing facts is typical during the data-modeling process. We examine documents and conduct user interviews, then create a data model with the data that we have. We also note where data are missing and supply that data later as we learn more. Thus, there is no need to stop data modeling when something is unknown; we just note that it is unknown and keep going with the goal of supplying missing information at some later point.

Suppose we talk with Heather and determine that customers can attend as many seminars as they would like, but that she would like to be able to record customers even if they have not been to a seminar. (“Honey, I’ll take a customer wherever I can find one!” was her actual response.) Also, she never offers a seminar to fewer than 10 attendees. Given this information, we can fill out more of the E-R diagram as shown in Figure 4-11(b).

Before continuing, consider the minimum cardinality of the relationship from SEMINAR to CUSTOMER in Figure 4-11(b). The notation says that a seminar must have at least 10 customers, which is what we were told. However, this means that we cannot add a new SEMINAR to the database unless it already has 10 customers. This is incorrect. When Heather first schedules a seminar, it probably has no customers at all, but she still would like to record it in the database. Therefore, even though she has a

---

**FIGURE 4-11**
Initial E-R Diagram for Heather Sweeney Designs

(a) First Version of SEMINAR and CUSTOMER E-R Diagram

(b) Second Version of SEMINAR and CUSTOMER E-R Diagram

(c) Third Version of SEMINAR and CUSTOMER E-R Diagram
business policy of requiring at least 10 customers at a seminar, we cannot place this limit as a constraint in the data model.

In Figure 4-11(b), neither of the entities has an identifier. For SEMINAR, the composites (Date, Time, Location) and (Date, Time, Title) are probably unique and could be the identifier. However, identifiers will become table keys during database design, and these will be large character keys. A surrogate key is probably a better idea here, so we will add one. Also, looking at our data and thinking about the nature of e-mail addresses, we can reasonably suppose that EmailAddress can be the identifier of CUSTOMER. All of these decisions are shown for the E-R diagram in Figure 4-11(c).

### The Customer Form Letter

Figure 4-12 shows a form letter that Sweeney Designs uses. Eventually, Heather would like to send messages like this via e-mail, as well. Accordingly, we will represent this form letter with an entity called CONTACT, which could be a letter, an e-mail, or some other form of

---

**FIGURE 4-12**

Customer Form Letter

Heather Sweeney Designs
122450 Rockaway Road
Dallas, Texas 54331
555-123-4565

Ms. Nancy Jacobs
1400 West Palm Drive
Fort Worth, Texas 45887

Dear Ms. Jacobs:

Thank you for attending my seminar “Kitchen on a Budget” at the San Antonio Convention Center. I hope that you found the seminar topic interesting and helpful for your design projects.

As a seminar attendee, you are entitled to a 15 percent discount on all of my video and book products. I am enclosing a product catalog and I would also like to invite you to visit our Web site at www.Sweeney.com.

Also, as I mentioned at the seminar, I do provide customized design services to help you create that just-perfect kitchen. In fact, I have a number of clients in the Fort Worth area. Just give me a call at my personal phone number of 555-122-4873 if you’d like to schedule an appointment.

Thanks again and I look forward to hearing from you!

Best regards,

Heather Sweeney
customer contact. She uses several different form letters (and, in the future, e-mails); Heather refers to each one by a number. Thus, she has form letter 1, form letter 2, and so forth. For now, we will represent the attributes of CONTACT as ContactNumber and Type, where Type can be either Form Letter or, in the future Email or some other type.

Reading the form letter, we see that it refers to a seminar and a customer. Therefore, we will add it to the E-R diagram with relationships to both of these entities as shown in Figure 4-13(a). A seminar can result in many contacts and a customer may receive many contacts, so the maximum cardinality of these relationships is N. However, neither a customer nor a seminar need generate a contact, so the minimum cardinality of these relationships is zero.

Working from CONTACT back to SEMINAR and CUSTOMER, we can determine that the contact is for a single CUSTOMER and refers to a single SEMINAR, so the maximum cardinality in that direction is one. Also, some of the form letters refer to seminars and some do not, so the minimum cardinality back to SEMINAR is zero. However, a contact must have a customer, so the minimum cardinality of that relationship is one. These cardinalities are shown in Figure 4-13(a).

Now, however, consider the identifier of CONTACT, which is shown as unknown in Figure 4-13(a). What could be the identifier? None of the attributes by themselves suffice because many contacts will have the same values for ContactNumber, ContactType, or Date. Reflect on this for a minute, and you will begin to realize that some attribute of CUSTOMER has to be part of CONTACT. That realization is a signal that something is wrong. In a data model, the same attribute should not logically need to be part of two different entities.

Could it be that CONTACT is a weak entity? Can a CONTACT logically exist without a SEMINAR? Yes, because not all CONTACTs refer to a SEMINAR. Can a CONTACT logically exist without a CUSTOMER? The answer to that question has to be no. Whom would we be contacting without a CUSTOMER? Aha! That’s it: CONTACT is a weak entity, depending on CUSTOMER; in fact, it is an ID-dependent entity because the identifier of CONTACT includes the identifier of CUSTOMER.

Figure 4-13(b) shows the data model with CONTACT as a weak entity on CUSTOMER. After further interviews with Heather, it was determined that she never
Figure 4-13: Heather Sweeney Designs Data Model with Contact

(a) Version without CONTACT as Weak Entity

(b) Version with CONTACT as Weak Entity

(c) Version with Amended CUSTOMER
contacts a customer more than once on the same day, so Date can be the identifier of CONTACT (as ID-dependent within CUSTOMER).

This E-R diagram has one other problem. The contact letter has the customer's address, but the CUSTOMER entity has no address attributes. Consequently, they need to be added as shown in Figure 4-13(c). This adjustment is typical; as more forms and reports are obtained, new attributes and other changes will need to be made to the data model.

The Sales Invoice

The sales invoice that Heather uses to sell books and videos is shown in Figure 4-14. The sales invoice itself will need to be an entity, and because the sales invoice has customer data, it will have a relationship back to CUSTOMER. (Note that we do not duplicate the customer

![FIGURE 4-14 Sales Invoice](image-url)
data, because we can obtain data items via the relationship; if data items are missing, we add them to CUSTOMER.) Because Heather runs her computer with lax security, she decided that she did not want to record credit card numbers in her computer database. Instead, she records only the PaymentType value in the database and files the credit card receipts in a (locked) physical file with a notation that relates them back to an invoice number.

Figure 4-15(a) shows a first cut at the data model with INVOICE. This diagram is missing data about the line items on the order. Because there are multiple line items, the line item data cannot be stored in INVOICE. Instead, a weak entity, LINE-ITEM, must be defined. The need for a weak entity is typical for documents that contain a group of repeating data; if the repeating group is not logically independent, then it must be made into a weak entity. Figure 4-15(b) shows the adjusted design.

As shown, LINE-ITEM has no identifier. Because it belongs to an identifying relationship from INVOICE, it needs an attribute that can be used to identify a particular LINE-ITEM within an INVOICE. We could use Description, but that would imply that the same item may not appear on a given invoice more than once, and this might be too constraining. Another option is to add a LineNumber attribute that identifies the line on which an item appears. We will add a LineItem attribute to our design.

We need to make one more correction to this data model. Heather sells standard products—her books and videos have standardized names and prices. She does not want the person who fills out an order to be able to use nonstandard names or prices. Accordingly, we need to add a PRODUCT entity and relate it to LINE-ITEM as shown in Figure 4-15(c).

Observe that UnitPrice is an attribute of both PRODUCT and LINE-ITEM. This was done so that Heather can update UnitPrice without impacting the recorded orders. At the
time a sale is made, UnitPrice in LINE-ITEM is set equal to UnitPrice in PRODUCT. The LINE-ITEM UnitPrice never changes. However, as time passes and Heather changes prices for her products, she can update UnitPrice in PRODUCT. If UnitPrice were not copied into LINE-ITEM, when the PRODUCT price changes, the price in already-stored LINE-ITEMs would change as well, and Heather does not want this to occur. This means that although two attributes are named UnitPrice, they are different attributes used for different purposes.

Also note in Figure 4-15(c) that based on interviews with Heather, we have added ProductNumber and QuantityOnHand to PRODUCT. These attributes do not appear in any of the documents, but they are known by Heather and are important to her.

**Attribute Specifications**

The data model in Figure 4-15(c) shows entities, attributes, and entity relationships, but it does not document details about attributes. To do that, the development team needs to create a table like that shown in Figure 4-16. Here, the data format and properties of the attributes of each entity are documented. These attributes and properties are used to create the tables in the database design, as you will see in the next chapter.
FIGURE 4-15

Final Data Model for Heather Sweeney Designs

(c) Finished Data Model

FIGURE 4-16

Attribute Specifications

<table>
<thead>
<tr>
<th>Entity Name</th>
<th>Attribute Name</th>
<th>Data Format (Length)</th>
<th>Required?</th>
<th>Default Value</th>
<th>Input Mask</th>
<th>Remarkable Constraints</th>
</tr>
</thead>
<tbody>
<tr>
<td>SEMINAR</td>
<td>SeminarID</td>
<td>AutoNumber</td>
<td>Yes</td>
<td>DBMS supplied</td>
<td>mm/dd/yyyy</td>
<td>Surrogate</td>
</tr>
<tr>
<td>SEMINAR</td>
<td>Date</td>
<td>Date</td>
<td>Yes</td>
<td>None</td>
<td>hh(AM/PM)</td>
<td></td>
</tr>
<tr>
<td>SEMINAR</td>
<td>Time</td>
<td>Text(4)</td>
<td>No</td>
<td>None</td>
<td>hh(AM/PM)</td>
<td></td>
</tr>
<tr>
<td>SEMINAR</td>
<td>Location</td>
<td>Text(7)</td>
<td>Yes</td>
<td>None</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SEMINAR</td>
<td>Title</td>
<td>Text(35)</td>
<td>Yes</td>
<td>'Intro'</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CUSTOMER</td>
<td>Name</td>
<td>Text(35)</td>
<td>Yes</td>
<td>None</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CUSTOMER</td>
<td>Phone</td>
<td>Text(10)</td>
<td>No</td>
<td>None</td>
<td>(nnn)nnn-nnn</td>
<td></td>
</tr>
<tr>
<td>CUSTOMER</td>
<td>Email Address</td>
<td>Text(50)</td>
<td>No</td>
<td>None</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CUSTOMER</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Business Rules

While creating the data model, the development team needs to be on the lookout for business rules that constrain data values and the processing of the database. We encountered such a business rule with regard to CONTACT, when Heather stated that no more than one form letter or e-mail per day is to be sent to a customer.

In more complicated data models, many such business rules would exist. These rules are generally too specific or too complicated to be enforced by the DBMS. Rather, application programs or other forms of procedural logic need to be developed to enforce such rules. You will learn more about this when we discuss stored procedures and triggers in the next chapter.

Validating the Data Model

After the data model has been completed, it needs to be validated. The most common way to do this is to show it to the users and obtain their feedback. However, a large, complicated data model is off-putting to many users, so often the data model needs to be broken into sections and validated piece by piece or expressed in some other terms that are more understandable.

As mentioned earlier in this chapter, prototypes are sometimes constructed for users to review. This is because prototypes are easier for users to understand and evaluate than data models. Prototypes can be developed that show the consequences of data model design decisions without requiring the users to learn E-R modeling. For example, showing a form with room for only one customer is a way of indicating that the maximum cardinality of a relationship is one. If the users respond to such a form with the question “But where do I put the second customer?” you know that the maximum cardinality is greater than one.

It is relatively easy to create mock-ups of forms and reports using Microsoft Access wizards. Such mock-ups are often developed even in situations where Access is not going to be used as the operational DBMS. The mock-ups are used to demonstrate the consequences of data modeling decisions.

Finally, the data model needs to be evaluated against all use cases. For each use case, the development team needs to verify that all of the data and relationships necessary to support the use case are present and accurately represented in the data model.

Data model validation is exceedingly important. It is far easier and cheaper to correct errors at this stage than it is to correct them after the database has been designed and implemented. Changing a cardinality in a data model is a simple adjustment to a document; changing the cardinality later might require the construction of new tables, new relationships, new queries, new forms, new reports, and so forth. So, every minute spent validating a data model will pay great dividends down the line.

SUMMARY

The process of developing a database system consists of three stages: requirements, design, and implementation. During the requirements stage, users are interviewed, systems requirements are documented, and a data model is constructed. Often prototypes of selected portions of the future system are created. During the design phase, the data model is transformed into a relational database design. During the implementation stage, the database is constructed and filled with data, and queries, forms, reports, and application programs are created.

In addition to a data model, the development team also must determine data-item data types, properties, and limits on data values. Business rules that constrain database activity also need be documented.

The entity-relationship model is the most popular tool used to develop a data model. With the E-R model, entities—which are identifiable things of importance to the
users—are defined. All of the entities of a given type form an entity class. A particular entity is called an instance. Attributes describe the characteristics of entities, and one or more attributes identify an entity. Identifiers can be unique or nonunique.

Relationships are associations among entities. The E-R model explicitly defines relationships. Each relationship has a name, and there are relationship classes as well as relationship instances. According to the original specification of the E-R model, relationships may have attributes; however, this is not common in contemporary data models.

The degree of a relationship is the number of entities participating in the relationship. Most relationships are binary. The three types of binary relationships are 1:1, 1:N, and N:M. A recursive relationship occurs when an entity has a relationship to itself.

In traditional E-R diagrams such as the traditional E-R model, entities are shown in rectangles, and relationships are shown in diamonds. The maximum cardinality of a relationship is shown inside the diamond. The minimum cardinality is indicated by a hash mark or an oval.

A weak entity is one whose existence depends on another entity; an entity that is not weak is called a strong entity. Weak entities are shown in rectangles with rounded corners, and the relationship on which the entity depends is indicated by a diamond with rounded corners. In this text, we further define a weak entity as an entity that logically depends on another entity. An entity can have a minimum cardinality of one in a relationship with another entity and not necessarily be a weak entity.

Unified Modeling Language (UML), an object-oriented program (OOP) design tool, has defined a new style of E-R diagrams. You should be familiar with diagrams of that style, but you also should realize that when creating a database design, no fundamental difference exists between the traditional style and the UML style.

In addition to E-R diagrams, a data model includes attribute specifications such as those shown in Figure 4-16. Also, while creating a data model, the development team needs to document business rules that constrain database activity.

After E-R models are completed, they must be evaluated. The development team can show the data model, or portions of the data model, directly to the users for evaluation. This requires the users to learn how to interpret an E-R diagram. Sometimes, instead of the data model, prototypes that demonstrate the consequences of the data model are shown to the users. Such prototypes are easier for users to understand.

**REVIEW QUESTIONS**

4.1 Name the three stages in the process of developing database systems. Summarize the tasks in each.
4.2 What is a data model and what is its purpose?
4.3 What is a prototype and what is its purpose?
4.4 What is a use case and what is its purpose?
4.5 Give an example of a data constraint.
4.6 Give an example of a business rule that would need to be documented in a database development project.
4.7 Define the term *entity* and give an example.
4.8 Explain the difference between an entity class and an entity instance.
4.9 Define the term *attribute* and give examples for the entity you created in question 4.7.
4.10 Which attribute defined in your answer to question 4.9 identifies the entity?
4.11 Explain what a composite identifier is and give an example.
4.12 Define the term *relationship* and give an example.
4.13 Explain the difference between a relationship class and a relationship instance.
4.14 Define the term *degree of relationship*. Give an example, other than the one in this text, of a relationship greater than degree 2.

4.15 List and give an example of the three types of binary relationships. Draw an E-R diagram for each.

4.16 Give an example of a recursive relationship other than the one shown in this chapter.

4.17 Define the terms *maximum cardinality* and *minimum cardinality*.

4.18 Name and sketch the traditional symbols used in E-R diagrams for (a) an entity, (b) a relationship, and (c) a weak entity and its relationship.

4.19 Give an example E-R diagram for the entities DEPARTMENT and EMPLOYEE, which have a 1:N relationship. Assume that a DEPARTMENT does not need to have an EMPLOYEE, but that every EMPLOYEE does have a DEPARTMENT.

4.20 Show example attributes for DEPARTMENT and EMPLOYEE (from question 4.19). Use traditional symbols.

4.21 Define the term *weak entity* and give an example other than one in this text.

4.22 Explain the ambiguity in the definition of the term *weak entity*. Explain how this book interprets this term. Give examples, other than those in this text, of each type of weak entity.

4.23 Define the term *ID-dependent entity* and give an example other than one in this text.

4.24 List important attribute properties that need to be developed as part of a data model.

4.25 Create example properties for five of the attributes from your work for question 4.20.

4.26 Give an example of a business rule for your work for question 4.19.

4.27 Describe why it is important to evaluate a data model.

4.28 Summarize one technique for evaluating a data model, and explain how that technique could be used to evaluate the data model in Figure 4-15.

---

**EXERCISES**

4.29 Suppose that Heather Sweeney wants to include records of her consulting services in her database. Extend the data model in Figure 4-15(c) to include CONSULTING-PROJECT and DAILY-PROJECT-HOURS entities. CONSULTING-PROJECT contains data about a particular project for one of her customers, and DAILY-PROJECT-HOURS contains data about the hours spent and a description of the work accomplished on a particular day for a particular project. Use strong and/or weak entities as appropriate. Specify minimum and maximum cardinalities. Use traditional E-R model diagrams.

4.30 Extend your work for question 4.29 to include supplies that Heather uses on a project. Assume that she wants to track the description, price, and amount used of each supply. Supplies are used on multiple days of a project.

4.31 Develop a model of the boxcars on a railway train. Use recursive relationships.

4.32 Develop a model of a genealogical diagram. Model only biological parents; do not model stepparents. Use more than one relationship if necessary.

4.33 Develop a model of a genealogical diagram. Model all parents, including stepparents. Use more than one relationship if necessary.
GARDEN GLORY PROJECT QUESTIONS

Garden Glory wants to expand its database applications beyond the recording of property services. The company still wants to maintain data on owners, properties, employees, and services, but it wants to include other data as well. Specifically, Garden Glory wants to track equipment, how it is used during services, and equipment repairs. In addition, employees need to be trained before using certain equipment, and management wants to be able to determine who has obtained training on which equipment.

With regard to properties, Garden Glory has determined that most of the properties it services are too large and complex to be described in one record. The company wants the database to allow for many subproperty descriptions of a property. Thus, a particular property might have subproperty descriptions of Front Garden, Back Garden, Second-Level Courtyard, and so on. For better accounting to the customers, services are to be related to the subproperties rather than to the overall property.

Develop an E-R data model that will meet Garden Glory’s new requirements. Use traditional-style diagrams, and specify entities and relationships only. For brevity, do not specify attributes. Justify the decisions you make regarding minimum and maximum cardinality. Describe how you would go about validating this model.

JAMES RIVER JEWELRY PROJECT QUESTIONS

James River Jewelry wants to expand its database applications beyond the recording of purchases for awards. The company still wants to maintain data on customers, purchases, and awards, but it wants to include other data, as well. Specifically, James River wants to record artists and styles and to keep track of which customers are interested in which artists and styles.

Also, most of the jewelry is sold on consignment, so the company does not pay the artist of a piece of jewelry until it is sold. Typically, the company pays artists 60 percent of the sales price, but the terms are negotiated separately for each item. For some items, the artists earn a larger percentage, and for others they earn less. Artists and James River personnel agree on the initial sales price at the time the item is brought to the shop. When an item has been in the shop for some time, James River may reduce the price; sometimes it renegotiates the sales percentage.

Develop an E-R data model that will meet James River’s new requirements. Use traditional-style diagrams, and specify entities and relationships only. For brevity, do not specify attributes. Justify the decisions you make regarding minimum and maximum cardinality. Describe how you would go about validating this model.

MID-WESTERN UNIVERSITY CHEMISTRY DEPARTMENT PROJECT QUESTIONS

The chemistry department wants to expand the data it keeps about its NMR magnets and their use. They still want to track principal investigators, grants, magnets, users, and appointments as described at the end of Chapter 3, but they want to track other data, as well.

Specifically, the department conducts several different training courses for each magnet. They want to keep track of which users have taken which training, and when they took the training. They also want to track who conducted the training. (The courses are taught by the users themselves.)

Create an E-R model for the chemistry department’s database. Include entities and relationships for the tables listed at the end of Chapter 3, as well as entities and relationships for the new requirements. Specify attributes that you think are applicable for the new entities.