Databases are used to represent facts in an organized manner. The facts of interest are some of those that are part of the enterprise under consideration. Thus the contents of any database represent facts about real objects that are of interest to someone. The first step in building a system to represent information involves determining the meaning and organization of the information that is part of the enterprise. This chapter discusses tools and techniques that are used to describe the organization and meaning of data in the form of a data model.

The importance of data modeling cannot be overstated. Limitations on the ability to represent facts are often created in this stage. These limitations can cause serious problems much later in the development process. Fixing such errors at a later stage can be extremely costly and time-consuming.

The process of producing a data model allows us to see the options provided by possible representations and to predict the effects of our decisions.
Consider our example of the BigHit Video enterprise. To conduct its business, BigHit Video must keep track of its videotapes, rentals, customers, orders, employees, purchases, and stores. Clearly, the success of this enterprise depends on the quality of its information management. The most obvious crucial activity is the management of rentals: knowing which customer has which video, how much each customer owes, and when the videos are due. Nevertheless, many other activities, and associated information resources, are just as important to the overall success of the enterprise.

To develop an effective information management application for BigHit Video, we must understand what information must be stored and how that information is likely to be used.

Discovering and Specifying Requirements

No software development is successful unless the resulting product is useful. For an information system to be useful, it must record and manipulate information that is important to its users. The information must be accurate and complete. A developer must convert the ideas and opinions of the users into the working information system. Unfortunately, the requirements for success are almost never clear at the beginning of the development efforts. For this reason, the process of determining the requirements is often termed “discovery.”

Discovery of application requirements usually involves interviews with members of the organization and the collection and analysis of current documents and computer systems. From this information, developers will describe the objects and operations that must be included in the final system. An important part of discovery is determining the vocabulary that is used to describe the objects and operations. This terminology must be incorporated into the design by the developers. Adopting the user’s vocabulary shows a respect for the enterprise that often makes it much easier to persuade users to adopt the final system. In addition, it facilitates the communication between users and developers—a crucial factor in the system’s ultimate success.

For BigHit Video, documents might include customer applications, rental receipts, employee time cards, store schedules, and video purchase orders. From these information sources, we can discover the major data objects and many of their attributes. Interviews with employees will be needed to add detail and description to the objects and their attributes. In addition, interviews will reveal many of the activities of the enterprise. We also learn what terms are used within the organization to denote the objects, attributes, and activities.

Because BigHit Video is a fictional company, there are no employees to interview, no documents to peruse, and no previous operation to use as a model. We hope that the familiar nature of the video rental enterprise will allow each reader to supply the details of the appropriate information and behavior. Throughout the book, discussions will focus on the information and behavior that ought to be present in an enterprise of this type. Several sample documents for our fictional enterprise are included in the book to help in the process of information modeling.
The BigHit Video enterprise was chosen as the primary example for this book because it is a rich application domain that is familiar to most readers. The presentation of the applications will follow an incremental development strategy, rather than the complete design that is preferred by software professionals. The basic structure will be discovered and specified in this chapter. Later chapters will add to the software applications and extend the data and behavior models.

Organizing Information

Before we can specify the information content of a database, we must have the tools and vocabulary that will allow us to organize the specification. A schema is the precise description of one or more aspects of a database system. Many different kinds of schemas are used in database systems, each with its own purpose. The high-level or conceptual schemas are used to specify the organizational structure and information content of a system. A conceptual schema provides a means of communication between users and designers of information systems. Ideally, it should use the vocabulary of the enterprise in a precise way. Users can read it and understand what information will be stored in the system. Developers can read it and understand the requirements for system development. Once users and developers have written and agreed on the conceptual schema, each group can be confident that they are working toward the common goal of producing a useful system.

In later chapters, we will encounter logical schemas that define the information content in a manner that can be used to create a database, external schemas that organize the information in order to improve access for users and applications, and physical schemas that specify the representation of information in physical terms.

A database system provides for the representation of the characteristics of objects that occur in the enterprise. We call these objects entities and their characteristics attributes. For simplicity and clarity, we divide the entities into distinct classes (entity classes). The entities in a class share common attributes. Each entity has a value for each attribute of its class. In BigHit Video, we know that there is a Customer entity class with attributes lastName, firstName, and address, among others.

The objects of an enterprise have associations or relationships with other objects. A relationship between a customer and a videotape occurs when the customer rents the videotape. That relationship ends when the customer returns the videotape. A relationship represents a fact about the entities of the system. A relationship type represents the possibility that an entity of one class may have a relationship with an entity of another class. By defining a Rents relationship type as part of the schema, for example, we allow the system to record the rental relationships as they occur.

Each object in a relationship plays a specific role in it. The customer is the renter of the videotape, and the videotape is the object rented by the customer. Roles are particularly important in situations where two entity classes are linked by more than one relationship type and where a relationship type links an entity class to itself. For example, the WorksAt and Manages relationship types link the Employee and Store classes. An employee may have the role of manager or the role of worker. Likewise, the marriage relationship associates one person with another person.
We can use these concepts to organize our specification of the information content of a system. We can analyze the enterprise and identify its specific entities and entity classes. We can determine the attributes of each entity class and the types of relationships that may exist between entities. All of these information details must be specified in a conceptual schema.

In this chapter, we investigate two strategies for specifying information structure. The traditional approach was first described by Peter Chen in his 1976 paper “Entity–Relationship Model: Toward a Unified View of Data” [Chen76] that appeared in the *ACM Transactions on Database Systems*. A more recent strategy is to integrate data and behavior in an object-oriented model, as described in Section 3.2.

**Entity–Relationship Modeling**

2.3

An *Entity–Relationship (ER) model* is a high-level, conceptual model that describes data as entities, attributes, and relationships. It constrains the representation of data but does not specify it. In this model, particular attention is paid to the relationships because they represent the interactions between entities and require special treatment in database and application development. An ER model specifies the data requirements of an application and is usually accompanied by a behavioral model that specifies the functional requirements. The development of these two models proceeds in a coordinated fashion.

ER modeling concentrates on specifying the properties of the data rather than the storage requirements. It includes a detailed description of the names and types of all data that are part of the proposed database system.

**2.3.1 Entity Classes and Attributes**

The discovery process that was described in Section 2.1 identifies specific objects that must be stored and managed in the enterprise. The first step in ER modeling is to name and describe the classes of those objects. Table 2.1 gives our first list of the names and brief descriptions of important entity classes for BigHit Video.

A NOTE ABOUT NAMES

The convention used for names in this book is consistent with the naming convention adopted in the Java language (see Section 6.8.3 of the Java Language Specification). The name of a class or attribute consists of a descriptive noun or noun phrase. The first letter of the name is capitalized if it is the name of a class; it is not capitalized if it is the name of an attribute (a field in Java terminology). The first letter of each subsequent word is always capitalized. No underscores are used in names. Whenever the name of a class or attribute is used in the text, it will appear in a fixed-width (monospace) font.

Each entity class has specific attributes or properties that describe its characteristics. These attributes are used to further describe the entity classes. The attributes themselves exist somewhat independently of the entities. In particular, it is not unusual for an attribute to appear in more than one entity class. A **lastName**
attribute, for example, is likely to represent a characteristic of both the Customer class and the Employee class. Table 2.2 lists some (but not all) attributes that represent characteristics of the entity classes of Table 2.1. Each attribute is characterized by a name, a type, a domain of values, and a description.

**TABLE 2.2**

**Names and descriptions of some attributes for BigHit Video**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Domain of Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 title</td>
<td>String</td>
<td>Unbounded</td>
<td>The title of an item</td>
</tr>
<tr>
<td>2 lastName</td>
<td>String</td>
<td>30 characters</td>
<td>The last name of a person</td>
</tr>
<tr>
<td>3 firstName</td>
<td>String</td>
<td>30 characters</td>
<td>The first name of a person</td>
</tr>
<tr>
<td>4 rating</td>
<td>String</td>
<td>5 characters</td>
<td>The rating of a movie</td>
</tr>
<tr>
<td>5 ssn</td>
<td>String</td>
<td>10 digits</td>
<td>A Social Security number</td>
</tr>
<tr>
<td>6 accountId</td>
<td>Number</td>
<td>4-byte integer</td>
<td>The identifier of a customer account</td>
</tr>
<tr>
<td>7 numberRentals</td>
<td>Number</td>
<td>4-byte integer</td>
<td>Number of rentals for a customer</td>
</tr>
<tr>
<td>8 otherUsers</td>
<td>Set</td>
<td>Set of strings of 30 characters</td>
<td>Names of other people authorized to use this account</td>
</tr>
<tr>
<td>9 dateAcquired</td>
<td>Date</td>
<td>Month, day, year</td>
<td>Date that a videotape was acquired</td>
</tr>
<tr>
<td>10 address</td>
<td>Composite</td>
<td>2 strings of 30 characters, one string of 2 characters, and one string of 9 digits</td>
<td>An address that consists of a street, city, state, and ZIP code</td>
</tr>
</tbody>
</table>
A NOTE ABOUT STORING INFORMATION IN TABLES

Tables 2.1 and 2.2 are just the first of many uses in this book of two-dimensional structures to store information. Each table has a fixed number of columns, and each column has a name. Each row has a value for each column, and the meaning of these values comes from the meaning of the column. The values in a row go together to provide information about a single idea or entity. This structure is exactly the same one that is used by relational databases to store information.

Each of the attributes in lines 1–7 of Table 2.2 is single-valued. The first five attributes have character strings as their values. Each has its own form, however, as described in the Domain of Values column. This column specifies domain constraints, which are restrictions on the allowable values of the attributes. The title attribute (line 1) has no restriction on its length or form, but both lastName and firstName (lines 2 and 3) are limited to no more than 30 characters and the rating (line 4) to no more than 5 characters. The ssn attribute (line 5) is constrained in length to 10 characters and also constrained to contain only the digits 1–9. Attributes accountId and numberRentals (lines 6 and 7) are integer-valued.

Each of the attributes in lines 8–10 of Table 2.2 is multivalued. The otherUsers attribute (line 8) has a set of strings as its value. The value of this attribute is a set of the names of the other people who are allowed to use this customer account. This attribute has multiple values in the sense that each name is a value of the attribute. The values of dateAcquired and address (lines 9 and 10) are composites (or records). A dateAcquired value is composed of three fields: month, day, and year. The address attribute is composed of a street and city (30-character strings), a 2-character state, and a 9-digit zipcode. These attributes have multiple values in the sense that each field of the record is a value of the attribute.

A more complete description of an entity class includes its name, its description, and its list of attributes. Table 2.3 gives the attributes of some entity classes for BigHit Video. The attributes were defined in Table 2.2. The constraints on the attributes limit their values for a specific entity or among all of the entities of the class. The attributes are variously described as key, not null, and derived. The constraint not null requires that the attribute have a value other than the null value. Derived means that the attribute is not stored, but rather is derived—or calculated—from some other information.

Most entity classes have one or more attributes that form a key of the class. The key of an entity uniquely identifies it among all entities of the class. Each entity of the class must have a unique value for its key. In other words, the values of the key attributes are constrained in that no two different entities may have the same values for the key attributes. For example, the ssn attribute of the Employee class is a single attribute key. No two employees may have the same value for the ssn attribute. This limitation reflects the uniqueness of Social Security numbers. Because two employee entities represent different people, and different people have different Social Security numbers, the ssn attribute is an appropriate key for the Employee class.
In the case of the Customer class, the accountId attribute is an artificial attribute that was created for the purpose of being a key. No other attribute of a customer is guaranteed to be unique. The declaration of the key constraint for the attribute specifies that the information system must guarantee the uniqueness of the attribute values among the entities of the class. An attempt to add a new customer with the same accountId value as another customer must be disallowed.

In some cases, it is most appropriate to have a key that consists of multiple attributes. In this situation, it is the combination of values of the key attributes that is unique. For class Customer, for instance, we might find that the combination of lastName, firstName, and address forms a key. The accountId attribute was added to the class Customer as its key for several reasons. It is easier to enforce uniqueness with a single attribute key. As we will see in Chapter 4, it is much easier to represent relationships with single attribute keys. Finally, declaring that the three attributes form a key restricts the entities that we can store in the database. If two people have the same first and last names and the same address, this restriction will require that they be considered the same customer. In this situation, the data model may limit the applicability of the resulting information system. There will be no way to represent the real situation in the database.

An entity class that has no key and cannot exist by itself is called a weak entity class. Weak entities are discussed in more detail in Section 2.4.

<table>
<thead>
<tr>
<th>Class</th>
<th>Attribute</th>
<th>Constraints or Further Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Customer</td>
<td>accountId</td>
<td>Key</td>
</tr>
<tr>
<td></td>
<td>lastName</td>
<td>Not null</td>
</tr>
<tr>
<td></td>
<td>firstName</td>
<td></td>
</tr>
<tr>
<td></td>
<td>address</td>
<td></td>
</tr>
<tr>
<td></td>
<td>balance</td>
<td></td>
</tr>
<tr>
<td></td>
<td>otherUsers</td>
<td></td>
</tr>
<tr>
<td></td>
<td>numberRentals</td>
<td>Derived</td>
</tr>
<tr>
<td>Videotape</td>
<td>videotapeId</td>
<td>Key</td>
</tr>
<tr>
<td></td>
<td>title</td>
<td>Not null</td>
</tr>
<tr>
<td></td>
<td>genre</td>
<td></td>
</tr>
<tr>
<td></td>
<td>dateAcquired</td>
<td></td>
</tr>
<tr>
<td></td>
<td>rating</td>
<td></td>
</tr>
<tr>
<td>PayStatement</td>
<td>datePaid</td>
<td></td>
</tr>
<tr>
<td></td>
<td>hoursWorked</td>
<td></td>
</tr>
<tr>
<td></td>
<td>amountPaid</td>
<td></td>
</tr>
</tbody>
</table>
A NOTE ABOUT ATTRIBUTES AND ENTITY CLASSES

The data modeling process may identify attributes that could be entity classes themselves. It is the designer’s responsibility to distinguish between attributes and entity classes. For example, the address attribute of a customer is a composite attribute with several attributes of its own. We might decide that an address is an attribute of a customer entity or an independent entity. We could argue that an address represents a lodging or place of business and hence is a real object that is independent of the people who live or work there. Alternatively, we could argue that the address is simply an attribute of a person used to communicate with the person. The lodging or place of business is of no interest to our information system. The design in this book treats addresses as attributes.

Unfortunately, there is no simple rule to use in making these distinctions. A designer must decide whether an attribute should be an entity on the basis of whether it is sufficiently important to the users or is independent of other entities.

2.3.2 Entity Instances and Attribute Values

An entity is an instance of an entity class. Each entity is distinguished by the values of its attributes. Tables 2.4 and 2.5 show four entities of class \texttt{Customer} and five entities of class \texttt{Videotape}, respectively.

Each entity has a value for each of its attributes, but that value may be the special value \texttt{null}, which is represented here as an empty field. For example, the values of the \texttt{otherUsers} attribute of the middle two rows of Table 2.4 are null. It is not always easy to know what null means. In this case, because the attribute is a set of values, null refers to the empty set value. That is, there are no other users for this customer. In other cases, a null may mean

- **Not applicable**: the attribute is not applicable to this entity and hence should not have a value;
- **Missing**: a value of the attribute exists but is not recorded; or
- **Unknown**: the value of the attribute may be either missing or not applicable.

For example, the spouse’s name attribute of an unmarried person must be null because the attribute is not applicable to the person. A null value of a height attribute for a person means that the value is missing, because every person has a height. A null value for a phone number, however, may mean that the person has no phone (not applicable) or that the person has a phone but the value is not recorded (missing). The null value may mean that the person refuses to give the phone number, or that the number that was originally given was wrong or changed and was replaced by null. In the latter case, it may be appropriate to try to determine the customer’s phone number. Is it appropriate to ask the customer for a phone number the next time he or she is in the store? The null value does not give enough information for us to determine how to interpret the attribute.

A three-valued logic can be used to make the distinction between missing and unknown. Consider an attribute \texttt{hasPhone} of a person. A true value means the person has a phone and a false value means the person has no phone. A null value
### TABLE 2.4

*Entities of class Customer*

<table>
<thead>
<tr>
<th>account Id</th>
<th>last Name</th>
<th>first Name</th>
<th>street</th>
<th>city</th>
<th>state</th>
<th>zipcode</th>
<th>otherUsers</th>
<th>Rentals</th>
<th>number balance</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>Block</td>
<td>Jane</td>
<td>1010 Main St.</td>
<td>Apopka</td>
<td>FL</td>
<td>30458</td>
<td>Joe Block, Greg Jones</td>
<td>3</td>
<td>0.00</td>
</tr>
<tr>
<td>102</td>
<td>Hamilton</td>
<td>Cherry</td>
<td>3230 Dade St.</td>
<td>Dade City</td>
<td>FL</td>
<td>30555</td>
<td></td>
<td>1</td>
<td>3.47</td>
</tr>
<tr>
<td>103</td>
<td>Harrison</td>
<td>Kate</td>
<td>103 Dodd Hall</td>
<td>Apopka</td>
<td>FL</td>
<td>30457</td>
<td></td>
<td>0</td>
<td>30.57</td>
</tr>
<tr>
<td>104</td>
<td>Breaux</td>
<td>Carroll</td>
<td>76 Main St.</td>
<td>Apopka</td>
<td>FL</td>
<td>30458</td>
<td>Judy Breaux, Cyrus Lambeaux, Jean Deaux</td>
<td>2</td>
<td>34.58</td>
</tr>
</tbody>
</table>

### TABLE 2.5

*Entities of class Videotape*

<table>
<thead>
<tr>
<th>videoId</th>
<th>dateAcquired</th>
<th>title</th>
<th>genre</th>
<th>length</th>
<th>rating</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>1/25/98</td>
<td>The Thirty-Nine Steps</td>
<td>mystery</td>
<td>101</td>
<td>PG</td>
</tr>
<tr>
<td>145</td>
<td>5/12/95</td>
<td>Lady and the Tramp</td>
<td>animated comedy</td>
<td>93</td>
<td>G</td>
</tr>
<tr>
<td>90987</td>
<td>3/25/99</td>
<td>Elizabeth</td>
<td>costume drama</td>
<td>123</td>
<td>PG-13</td>
</tr>
<tr>
<td>99787</td>
<td>10/10/97</td>
<td>Animal House</td>
<td>comedy</td>
<td>87</td>
<td>PG-13</td>
</tr>
<tr>
<td>123</td>
<td>3/25/86</td>
<td>Annie Hall</td>
<td>romantic comedy</td>
<td>110</td>
<td>PG-13</td>
</tr>
</tbody>
</table>
means that it is unknown whether the person has a phone. It has even been suggested that a four-valued logic is appropriate: true, false, missing, and not applicable.

There is a great deal of controversy about the appropriate meaning and usage of nulls in information systems. C. J. Date devotes an entire chapter of his *Introduction to Database Systems* [Date99] to “missing information.”

### 2.3.3 Relationship Types and Relationship Instances

A relationship between two objects represents some association between them. In ER modeling, relationships are characterized by type (or class). For instance, two people can be associated as parent and child (a parent–child relationship) or as husband and wife (a marriage relationship). The *relationship* occurs between two specific objects and is an instance of a *relationship type* between the classes of the objects.

Each object plays a specific *role* in the relationship. If two people are associated by marriage, one person is the wife and one is the husband. That is, the relationship is “marriage,” and the roles of the individuals are “wife” and “husband.” The name of a role expresses the function of a particular entity in a relationship.

A relationship type represents the possibility that one entity may have an association with another entity. A relationship instance represents the fact that a specific entity does have this particular association with another specific entity. For example, the marriage relationship type is defined between entity class `Person` and entity class `Person`. It does not, however, guarantee that any specific person is married to any other specific person or even that anyone is married to anyone else.

The name of a relationship is a verb phrase that can be used in a sentence. We say, “Jane Block rents *Annie Hall,*” to represent an instance of the *Rents* relationship type. Similarly, “A customer may rent a videotape” expresses the *Rents* relationship type. We also use the roles, as in “Jane Block is the wife of Joe Block” or “Joe Block is the husband of Jane Block.”

Constructing such expressions may help in the discovery process because this kind of sentence reveals the presence of the relationship type. When someone uses such a sentence to describe facts about the enterprise, that person is expressing the existence of the relationship type.

Relationships often have their own attributes. A marriage relationship has a wedding date attribute, for instance. The *Rents* relationship type has attributes `dateDue`, `dateRented`, `amountPaid`, and `amountDue`.

### 2.3.4 Relationships Are Not Attributes

A major goal of conceptual modeling is to identify the relationships between entities. Relationships are singled out for special treatment. As we will see in Chapter 4, we can use many strategies to represent relationships in database systems. The conceptual model identifies the relationships, but does not determine an exact representation. To maintain flexibility, we must ensure that no relationship is specified as an attribute. Unfortunately, the discovery process will often identify fields that appear to be attributes, but should actually be relationships.
Consider the example of a rental receipt from BigHit Video that appears as Fig. 2.1. It includes information about the rental, the customer, and the videotape. The account ID field that appears in the receipt is the identifier of a specific customer. As we have already determined that a customer is a separate entity, the field actually serves as an indication of a relationship. The name and address of the customer are attributes of the Customer entity class, not of the Rental class. Similarly, the video ID and title are attributes of a related Videotape entity. Analysis of this receipt yields the information that a Rental has attributes dateRented, dateDue, and cost, and relationships to a Customer and to a Videotape. The other fields that appear on the receipt are not attributes of Rental.

A simple analysis of an entity class may identify attributes that represent relationships. For example, the Social Security number (ssn) and name of the employee appear on a pay statement. This fact leads us to add those attributes to entity class PayStatement. We also know these attributes apply to the Employee entity class and ssn is its key attribute.

The modeling process is incomplete because attributes of one entity class (Employee) are attached to another class (PayStatement). We must recognize that the ssn attribute of PayStatement is an indication that class PayStatement is related to class Employee, delete these attributes from PayStatement, and add a relationship between the classes.

### 2.3.5 Constraints on Relationships

Relationship constraints limit the application of relationship types. A cardinality constraint on a relationship type puts restrictions on the number of relationships that may exist at any one time. When we say that a relationship type is “to-one,” we mean that one entity may be related to at most one other entity through this type. For instance, the marriage relationship type is one-to-one because each person may be married to at most one other person. Making a relationship type to-one places a cardinality constraint on instances of the relationship.

As we have already seen, each relationship has two roles. Each cardinality constraint is applied to a single role of a relationship. The one-to-one constraint
on the marriage relationship is actually two constraints. The role of wife and the role of husband are each constrained to have cardinality no more than one. For a particular person, the set of people for whom she is the wife has cardinality no more than one. Together, these constraints form a cardinality ratio constraint on the marriage relationship type. In the mathematical sense, the set of relationships that exist at any one time forms a one-to-one relationship.

Four basic types of cardinality ratios exist:

- **One-to-one.** An entity in either role may participate in at most one relationship, and hence have at most one related object.

- **One-to-many.** An entity in one role may have any number of relationships, but an entity in the other role may have at most one.

- **Many-to-one.** The same as one-to-many, but reversed.

- **Many-to-many.** An entity in either role may participate in any number of relationships.

Figure 2.2 illustrates entities related by the Rents relationship type. The oval on the left lists theaccountId attributes of some Customer entities. The oval on the right lists the videoId attributes of some Videotape entities. The rectangle in the center contains a square for each relationship between a customer and a videotape. As the figure illustrates, each relationship connects exactly two entities—one from each class. The first relationship associates customer 101 (Jane Block) with videotape 90987 (Elizabeth). This relationship represents the fact that Jane Block has rented Elizabeth and not yet returned it.

Rents is one-to-many, because each customer may rent many videotapes, but each videotape can be rented at most once. Each customer may have many
relationships, but each videotape has at most one relationship. The customers and videotapes that have no relationships illustrate the lack of minimum cardinality. Customer 102 (Cherry Hamilton) has no current rentals. Videotape 145 (*Lady and the Tramp*) has no current rentals, and so is available for rental.

Figure 2.3 illustrates a collection of *PreviouslyRented* relationships that record the history of rentals. The first relationship records the fact that customer 101 (Jane Block) rented and returned videotape 123 (*Annie Hall*).

The *PreviouslyRented* relationship type is many-to-many, because each videotape may be checked out many times by many different customers. This change in cardinality ratio does not affect the relationship object. Each such object still connects exactly two entities. The videotapes that have more than one relationship illustrate the change in cardinality. One fact that can be seen by reviewing Fig. 2.3 is that customer 102 (Cherry Hamilton) rented and returned videotape 101 (*The Thirty-Nine Steps*) twice, as illustrated by the fourth and fifth relationships.

A *maximum cardinality constraint* restricts the number of relationships of a particular type that an entity may have. If we want to ensure that no customer can rent more than 10 videotapes at a time, we must specify the cardinality of the renter role to be no more than 10.
Constraints may also specify a minimum number of relationships for a particular entity. A constraint that specifies whether at least one relationship must exist for each entity is often called a participation constraint. That is, if the cardinality is “at least one,” then each entity must participate in the relationship type by being related to at least one other entity.

### 2.3.6 Relationships of Higher Degree

To this point, the discussion has assumed that a relationship is binary. That is, each relationship type connects two entity classes. It is not unusual to have multiparty relationships in an enterprise. For instance, a store may purchase videotapes from a supplier. This purchase represents a three-way (ternary) relationship. The relationship type connects three entity classes. The number of entity classes that are linked by a relationship type is called the degree of the relationship type.

#### Entity–Relationship Diagrams

An important aspect of ER modeling is the representation of a model by a diagram. The diagrams make ER models easier to understand and to explain. Figure 2.4 shows an ER model that includes classes Customer and Videotape and relationship type Rents. Entity classes are represented by rectangles, attributes by ovals, and relationships by diamonds and lines. Note that this diagram merely shows entity classes and relationship types—it does not describe any particular instances of them. The shaded boxes are comments and are not part of the ER diagram.

**Figure 2.4**

*Entity classes Customer and Videotape and relationship type Rents*
The attributes whose names are underlined are the keys of their entity classes. The border on numberRentals is dashed to show that it is derived. The multi-valued attribute otherUsers has a double border to indicate that it consists of a set of values. The fields of the composite attribute address are shown as its attributes.

Much of the detail that is needed to specify the information content, given in Tables 2.1 through 2.4, has been omitted from the diagram in Fig. 2.4. The descriptions of the classes and the types, descriptions, and constraints on the attributes are not shown here. Instead, this information must be maintained in text form as part of a specification of the system, as it is in Tables 2.1 and 2.2. This information is collected in a table called a data dictionary.

Cardinality constraints are represented in Fig. 2.4 by the symbols 1 and M that appear on the lines and by having single and double lines, as we will see in Fig 2.5. The symbol 1 on the diagram means that a videotape can be rented by no more than one customer. The symbol M means that a customer may rent many (zero or more) videotapes; that is, it is a one-to-many relationship type. The marks tell us how many entities of the related class (class near the cardinality mark) can be associated with one entity of the subject class (class on the other side of the diamond).

We can best understand the position of the cardinality symbol by creating sentences to represent the relationship roles. Each sentence is created by listing one entity, then the relationship, then the cardinality, and finally the other entity. The following two sentences were created by reading the diagram from left to right and right to left, respectively:

- A customer may rent many videotapes.
- A videotape may be rented by one customer.

This relationship allows for customers who have no videotapes rented and for videotapes that are not rented by any customer. It allows customers to rent any number of videotapes, but does not allow a videotape to be rented by more than one customer. From these facts, we can conclude that the Rents relationship represents current rentals and does not represent a history of rentals, as a videotape may be rented many times during its shelf life.

You might have noticed that the original list of entity classes (Table 2.1) gave Rental as an entity class, but the discussion in Section 2.3 and the diagram of Fig. 2.4 considered it as a relationship type. It is perfectly appropriate to represent that relationship as class Rental, as in Fig. 2.5. Class Rental has the attributes of relationship type Rents. A many-to-one relationship type and a one-to-one relationship type have replaced the relationship type between Customer and Videotape.

The double lines linking Rental to its two relationship types specify a participation constraint. These double lines denote that a Rental entity must participate in both relationships. That is, each rental entity must be related to a customer and a videotape. We can represent the relationship types with the following sentences:

- A customer may have many rentals.
- A rental must have one customer.
- A videotape may have one rental.
- A rental must have one videotape.
The sentence uses “may” before the verb if the entity is not required to participate in the relationship and “must” before the verb if it is required to participate.

A Rental object cannot exist without being related to a Customer and a Videotape, and no combination of attributes of class Rental is unique. Hence the attributes do not form a key for the class, and Rental must be an example of a weak entity class, denoted by the double border.

The double border on the diamond that relates Rental to Videotape marks this relationship type as an identifying relationship type for class Rental, and it marks Videotape as the owner entity class. From this notation, we know that a rental is identified by its relationship to a videotape. The related videotape is considered the owner of the rental. Without that relationship, the rental cannot exist. The key for a Rental entity is the videoId attribute, which is the key of the owner videotape.

As another example of how we draw diagrams for relationship types, consider the marriage relationship type that was described in Section 2.3.3. It relates one person to another, as shown in the ER diagram of Fig. 2.6. This diagram shows the entity class Person with its key attribute ssn and the relationship types MarriedTo and IsChildOf with their lines. The names of the roles wife, husband, child, and parent are shown on the diagram next to the lines of the relationship types. As shown in Fig. 2.6, for each pair of people related by parentage, one is the child and one is the parent.

Additional information contained in Fig. 2.6 is expressed by the following sentences:

- A person is the child of one or two parents.
- A person is the parent of zero or more children.

Reading along the relationship lines creates these sentences. Read in the order of the entity class, the relationship, the cardinality of the role, and the other role. The
cardinality symbol $1..2$ on the parent line means that the child may have between one and two parents. Figure 2.6 also shows the two names for the parent–child relationship. From the child’s point of view, the relationship is called IsChildOf; this phrase appears inside the diamond. Outside the diamond is the name of the relationship from the parents’ perspective: IsParentOf. The arrow on the child’s relationship line points toward the diamond. This depiction indicates that the name of the relationship inside the diamond refers to this role’s relationship name.

A NOTE ABOUT OVERSPECIFYING CARDINALITIES

Perhaps you’ve seen a flaw in the cardinalities of the diagram given in Fig. 2.6. If every person must have one or two parents, either there are infinitely many persons or some people are their own descendants. This problem is a common error created by overspecifying the cardinality of a relationship type. In this case, we must recognize that this relationship type will represent parent–child relationships between people who are part of the system. Some people in the system will have parents who are not part of the system. Consequently, the child must be allowed to have no parent.

Always keep in mind that we are not attempting to represent the entire world, but only a small part of it. The cardinality of a role should represent the number of related entities within the system of interest.

Although ER diagrams include declarations of cardinality constraints, many other constraints cannot be represented. The ER diagram for the IsChildOf relationship type restricts the cardinality of the child so that a child has no more than two parents. It does not, however, place any restrictions on which individuals are the parents. It is possible for Joe to be a child of Jane and Jane to be a child of
Joe at the same time. The diagram also allows Joe to be a child of Joe! Such nonfactual relationships should not be allowed, but cannot be excluded in an ER diagram. Instead, constraints on individual membership in relationships must be written down as part of the ER modeling process. These limitations will be considered for enforcement later in the development process, either as part of specifying the database or when developing applications.

Figure 2.7 gives examples of the basic symbols that we use to draw ER diagrams. Almost as many styles of ER diagramming exist as organizations that draw them. The style presented here is used by many developers, but is not necessarily the best one. Each designer must draw diagrams that conform to the style used by his or her organization. These diagrams serve as the primary means of communication between users and developers. The particular style chosen is more important to users than to developers. It must convey a precise specification to developers, but must be easily understood by trained users. Once system designers adopt a specific ER style, it is their responsibility to teach their users to understand it.

An ER Model for BigHit Video

2.5 Figure 2.8 is an ER diagram for the BigHit Video information system. It represents all of the entity classes from Table 2.1 and their relationship types. The diagram is not complete because it omits the attributes of most of the classes. You will be asked to finish the attribute and entity class definitions as part of the exercises.

In this section, we will look at what information this conceptual schema can represent and what it cannot represent. Careful study of this diagram will expose many of the issues that are raised in the discovery and specification stages. It will also illustrate how much detail can be specified by an ER model and how that detail constrains the resulting information system.

2.5.1 Recording the History of Rentals

Entity classes Customer, Rental, and Videotape are related as given in Fig. 2.5. As noted earlier, Rental entities represent the current state of video rentals. When a videotape is returned, the corresponding Rental entity is removed from the database. Certainly, most businesses need to record historical information. For example, BigHit Video wants to be able to determine which videotapes are being rented and what types of videotapes a particular customer tends to rent. Class Rental does not provide the information required for these analyses.

Class PreviousRental has been added to record the history of customer and videotape rentals. The difference between Rental and PreviousRental lies in the cardinality constraints of the relationship with Videotape. Class Rental has a one-to-one relationship with Videotape, whereas PreviousRental has a many-to-one relationship. This relationship allows many rentals of each videotape to be recorded. We also want to allow a PreviousRental entity to have no associated customer. That is, we will place a single line between PreviousRental and its relationship with a Customer. With this cardinality, we are free to delete
Cardinality Marks

1  No more than one related entity

M  Many (zero or more) related entities

i..j  At least i but not more than j related entities

Must participate in the relationship

May participate in the relationship

FIGURE 2.7  Symbols used in ER diagrams
FIGURE 2.8
ER diagram for BigHit Video
inactive customers and the records of their previous rentals without compromising our ability to analyze rental activity.

Class **PreviousRental** is a weak entity with a single identifying relationship. The related videotape does not uniquely identify the **PreviousRental**, however. This property is another difference between **Rental** and **PreviousRental**. Because a videotape can participate in at most one rental at a time, the rental is uniquely determined by its relationship to the videotape.

To uniquely identify a previous rental, we must add another attribute to the key. This attribute, called a *discriminator* or *partial key*, should uniquely identify the entity among all those related to a specific identifying entity. In this case, the **dateRented** attribute and the key of the related videotape together form a unique identification. The **dateRented** attribute *discriminates* among all of the previous rentals for a particular videotape. The partial key is shown with a dashed underline in the ER diagram in Fig. 2.8.

This definition of the key of entity class **PreviousRental** may create some problems for BigHit Video stores. In particular, under this definition, a videotape cannot be rented twice on the same day. The easiest way to eliminate this problem is to change **dateRented** into **dateTimeRented**, an attribute that includes both date and time. If this modification is not appropriate, we can create an artificial key, **rentalId**, to serve as the key of the entity class.

This strategy for handling rentals and previous rentals exposes some limitations of ER models. We have defined the relationship type between **Rental** and **Customer** as to-one so that a videotape can have at most one current rental. When a videotape is checked in, however, the **Rental** entity is deleted and a similar **PreviousRental** must be created. In some sense, the **Rental** is transformed into a **PreviousRental**. Unfortunately, the ER diagram has no way to represent this required transformation.

An alternative design is to have a single rental entity class with a to-many relationship type with **Customer**. In this strategy, the current rentals comprise all of the rentals with a null return date. In this design, however, the ER diagram cannot depict the requirement that there be at most one current rental. As designers, we must decide which is most important: ensuring that there is at most one current rental or ensuring that the previous rentals are reliably handled. The model of Fig. 2.8 is a compromise.

---

**A NOTE ABOUT ETHICAL ISSUES AND PRIVACY**

An important issue in recording previous rentals relates to the privacy of the customers. Some people may find it a violation of their privacy for a company like BigHit Video to keep track of all of the videotapes they have rented. Employees of BigHit Video could search the database to find out private information about its customers. Public libraries, for instance, are very careful to keep this kind of information private. U.S. courts have supported public libraries’ right to maintain the privacy of their circulation records. BigHit Video is not under the same constitutional constraints, but may not want to record a history of individual rentals.

Removal of the relationship type between **Customer** and **PreviousRental** would make it impossible for BigHit Video to record information about which customers rented which videotapes.
2.5.2 Employee Roles and Cardinalities

Two relationships exist between Employee and Store. An employee can be associated with a store as manager or as worker. These roles for the employee are shown next to the lines that connect Employee and its relationship types. The specification allows an employee to be both manager and worker. It also allows an employee to be a worker for more than one store—a usual occurrence in a business that has multiple outlets in an area.

According to the diagram, an employee can be the manager of no more than one store and each store has exactly one manager. This setup could be a problem if the business needs one person to manage more than one store. If the business fires a store manager, there will likely be a period of time when either there is no manager or some other manager will be asked to fill the vacancy temporarily. The database will be unable to represent this situation. As a result, it will be necessary to add a fictional employee to the Employee class and associate that employee with the store as manager. An alternative approach is to modify the cardinalities of the manages relationship type so that a store can have no manager and an employee can manage more than one store.

The precise specification of the cardinalities exposes this question about real business practices. It is the users’ responsibility to determine whether the representation of their enterprise is correct. It is the developers’ responsibility to find and expose the questions that always arise as part of the discovery and specification of information systems.

2.5.3 Purchase Orders and the True Meaning of Videotape

Class PurchaseOrder is a class that could be defined as weak. The cardinalities of its relationships require that a purchase order have a single supplier and at least one videotape; hence a purchase order cannot exist unless it is related to other entities. This structure is one characteristic of a weak class. Its natural key is some combination of date, supplier ID, and items. In this case, it is much more straightforward to create an artificial key (id) and to define PurchaseOrder as a strong entity, as is done in Fig. 2.8.

A weak entity has no key of its own. An example from BigHit Video is class PayStatement. This class has no key, because many employees may be paid on the same day. The relationship type PaidTo, however, is a many-to-one relationship that links each pay statement to a single employee.

The relationship between a purchase order and a videotape indicates that the specific videotape is being ordered as part of the purchase order. A purchase order can be related to many videotapes—many tapes are included in a single order. In contrast, a videotape can be related to only one order, because it is purchased only once. To create a purchase order, an entity must be added to the Videotape class for each item in the order. Each of these videotape entities is associated with a specific store. When the order is received, the information in the database can be used to determine where the tapes go.
This strategy is not the usual approach to purchasing. Typically, a purchase order contains a list of items to be purchased. Each item has some identifying information (for example, a catalog number) and a quantity. In this case, the intention may be to purchase 25 copies of *Lady and the Tramp*. Figure 2.4 would require that 25 entities be created, each with title “Lady and the Tramp,” genre “animated comedy,” and its own unique value for `videoId`. In turn, each entity would be individually linked with the purchase order. The effect would be an order for 25 items, rather than an order for one item with quantity 25.

This approach is a clear mistake in the diagram. To be correct, the conceptual schema should have a close correspondence to the real objects that it represents. As it now stands, a `PurchaseOrder` entity is not an accurate representation of a real purchase order.

We might be tempted to place a quantity attribute on the `Orders` relationship type, as shown in Fig. 2.9. Now the purchase order is a set of items, each with a quantity. In this model, however, the meaning of `Videotape` has changed. In Fig. 2.8, a `Videotape` entity represents a specific tape that may be rented. In Fig 2.9, a `Videotape` entity represents a specific title but not a specific physical tape. A tape can now be purchased multiple times and may represent more than one physical tape.

The problem that is exposed by this analysis is actually more than a problem with the correspondence between an entity class and its real counterpart. It is an error in the understanding of the nature of a videotape. Two different entity classes have been confused as a single class. One class represents the physical videotape—an object that can be rented, is taken away by a customer, and must be returned before its next rental. The other class represents a more conceptual object—the movie or the catalog item. Each physical videotape is a copy of a specific movie. The movie must be represented by another entity class. It is this object that is purchased and this object that a customer wants to find. After all, a customer will ask, “Do you have a copy of *Lady and the Tramp*?”, not “Do you have videotape number 112376?”

Figure 2.10 gives a more appropriate ER diagram for videotapes, movies, purchases, and sales. In this diagram, a videotape is a copy of a movie. The title, genre, and other attributes that are common to all copies of a single movie are attached to the movie class. A purchase order consists of many detail lines, each representing the purchase of some quantity of a single movie. The application that
handles the receipt of movie shipments will have to create \texttt{Videotape} entities for each videotape so that the tapes can be entered into the rental inventory.

### 2.5.4 Employees, Time Cards, and Pay Statements

The weak entity classes \texttt{TimeCard} and \texttt{PayStatement} record when employees work and what they are paid. In both cases, the entities are not uniquely determined by their identifying relationships. For instance, a pay statement is identified by its related employee, but is not unique for that employee. It is the combination of employee's \texttt{ssn} and \texttt{datePaid} that is unique. The attribute \texttt{datePaid} is called a \textit{discriminator} (or \textit{partial key}) because it identifies the entity among all of those that depend on the same strong entity.

Each time card is associated with an employee and a store. It records the date and the starting and ending times of a single period of work for one employee at one store. It is not possible to represent a situation in which an employee works in two stores with a single time card. Instead, a work period for an employee who begins work in one store, then changes to a different store, must be represented by two time cards.

### Chapter Summary

An information system is an organized repository of facts about an enterprise that includes application software to manipulate and create those facts. The development
of an information system must begin with data modeling. The goal of data modeling is to produce a conceptual schema for the information system. The data modeling process exposes design options and alternatives so that designers and users can consider their effects. However, faulty or incomplete data modeling often creates limitations on the ability of an information system to represent facts. Later phases of system development include the translation of the conceptual schema into a logical schema that can be used to create a relational database.

Discovery of application requirements—both information and processing—involves an investigation of the enterprise and its current information and processing methods. Developers must translate this information into precise data models using one of a variety of methods.

Entity–relationship modeling divides the information world into entities, attribute values, and relationships. An **entity class** represents the common properties of a collection of similar entities. An **attribute** is a property that describes a characteristic of an entity class. An **attribute value** is the value of that attribute for a specific entity in the class. An **entity** is an instance of an entity class. Each entity has a value for each attribute of its class.

A **relationship type** represents a particular association between entity classes. A **relationship** is an instance of a relationship type; it consists of two (or more) entities that are associated by the relationship type. A relationship type represents the possibility that two entities may be related. Each entity that participates in a relationship has a specific role.

A **cardinality ratio** is associated with each relationship type. This ratio limits the number of times that an individual entity may participate in the relationship type. In a one-to-one relationship type, an entity in either role may participate in at most one relationship. In a one-to-many relationship type, an entity in one role may have many related entities. In a many-to-many relationship type, an entity in either role may be related to many entities.

A **strong entity class** is one that has a **key**—that is, a set of attributes that uniquely determine an entity. A **weak entity class** has no key. The identity of a weak entity is determined by its identifying relationships along with zero or more attributes, or **discriminators**, of the entity.

An **Entity–Relationship diagram** (ER diagram) is a graphical representation of an ER model. In such a diagram, entity classes are represented as rectangles, attributes as ovals, and relationship types as diamonds and lines. ER diagrams depict the classes and interconnections, but do not show the individual entities and their properties.

**Key Terms**

- **Attribute value.** The values of a specific attribute for one entity.
- **Attributes (properties).** The characteristics that describe an entity.
- **Cardinality constraint.** A restriction on the cardinality of a role of a relationship. Typical constraints are **to-one**, in which an entity may be related to no more than one entity of the related type, and **to-many**, in which an entity may be related...
to an unlimited number of entities of the related class. A cardinality constraint may specify a minimum or maximum number of related entities.

**Cardinality ratio constraint.** A combination of two cardinality constraints, one on each role of a relationship. The four basic types of cardinality ratios are *one-to-one*, *one-to-many*, *many-to-one*, and *many-to-many*.

**Composite attribute.** An attribute whose value is composed of a collection of individual fields.

**Conceptual schema.** A precise definition of the data requirements of a system that is understandable to both users and developers of a database. This model includes detailed descriptions of data types, relationships, and constraints and is often represented as an ER model, ER diagram, or object-oriented model.

**Constraint.** A limitation on the contents of a database. Data models include constraints on the values of attributes and the cardinality of relationships, among others.

**Data dictionary.** A table that contains the descriptions of classes and the types, descriptions, and constraints on attributes of an information system.

**Discriminator.** An attribute of a weak entity class that identifies an entity from among all of those with the same identifying entities. A discriminator is part of the key of the weak entity class.

**Domain.** The set of allowable values of an attribute.

**Domain constraint.** A requirement that the values of an attribute must come from a specific domain.

**Entity.** An object in the real world that is of interest to the application.

**Entity class.** The common characteristics that represent a collection of entities.

**Entity–Relationship (ER) model.** A strategy for constructing conceptual data models using diagrams that focus on entity classes, relationship types, and attributes.

**External schema.** A definition of a user’s or application’s view of the information content of a system.

**Identifying relationship type.** A to-one relationship type between a weak entity class and a strong entity class that helps to uniquely identify an object of the weak class.

**Key.** A set of attributes of an entity class whose values uniquely identify an entity.

**Key constraint.** A constraint on the entities of a class such that no two different entities can have the same values for a specific set of attributes. This set of attributes acts as a key for the class.

**Logical schema.** The definition of the information content of a system in a manner that can be used to create a database.

**Multivalued attribute.** An attribute with a set of values.

**Null value.** A special attribute value that is different from any value in the domain of the attribute. The meaning of a null attribute value of an entity is ambiguous. It may represent a missing value, one that is unknown, or an attribute that is not applicable to the entity.

**Owner entity class.** The entity class that is related by an identifying relationship to a weak entity class.
Partial key. See Discriminator.

Participation constraint. A cardinality constraint on a role in a relationship that requires an entity to be related to at least one entity of the related class.

Physical schema. The definition of the information content of a system in physical terms.

Relationship (instance). An association between two or more entities.

Relationship type. A representation of the possibility that entities of two or more entity classes may be associated.

Role. The function of an entity in a relationship.

Schema. A precise description of one or more aspects of a database system.

Single-valued attribute. An attribute with a single, indivisible value.

Weak entity class. An entity class with no key. It must have at least one identifying relationship type.

Exercises

1. List four important characteristics that must be present for an information system to be successful.

2. Suppose that you are designing an information system for a university. What documents would you use to determine the information requirements?

3. Discuss the importance of data modeling to the success of an information system development. Be sure to describe how vocabulary can be used to improve the communication between developers and users.

4. What is meant by discovery in data modeling? What resources are available to developers in discovering information requirements?

5. Define the terms conceptual model, logical model, physical model, and external model.

6. Create a collection of documents for BigHit Video. Include the following items:
   a. Customer application
   b. Rental receipt
   c. Purchase order
   d. Employment application
   e. Time card
   f. Pay statement
   g. Report on rental activity

7. Characterize the difference between the following pairs of terms:
   a. Entity and entity class
   b. Relationship and relationship type
   c. Attribute value and attribute
   d. Strong entity class and weak entity class
   e. Conceptual schema and logical schema
8. What is a cardinality constraint? What is a participation constraint? Give an example of relationship types that are one-to-one, one-to-many, and many-to-many. What are the participation constraints on these relationship types?

9. Why are weak entity classes important to conceptual modeling? Give an example (not from BigHit Video) of a weak entity class. What are its identifying relationship(s), its owner entity class(es) and its discriminators, if any?

10. Give an example of a relationship type of degree higher than 2. Show how a weak entity class can represent this relationship type.

11. Augment the ER diagrams given in this chapter with attributes as follows:
   a. Augment Fig. 2.4 by including attributes for information about customer preferences, credit cards, and forms of identification. You may use the customer application from Exercise 6(a) as the model.
   b. Augment Fig. 2.4 by including attributes for information about videotapes, including length, rating, studio, and so on.
   c. Augment Fig. 2.8 by including attributes for employee information, as described in the employee application form of Exercise 6(d).

12. Consider the ER diagram shown in Fig. 2.10.
   a. Write a sentence (in English) that expresses the role of a supplier in the diagram.
   b. Write sentences that express the roles of a purchase order in the diagram.
   c. Write sentences that express the roles of a movie in the diagram.
   d. Can a movie be purchased from more than one supplier?
   e. Can a purchase order include more than one detail line for a single movie?
   f. Write three more questions like (d) and (e) that ask questions about cardinalities and participation that can be answered from the diagram.

13. Create an ER model to maintain information about a university’s course offerings, students, faculty, student registrations, and student transcripts.

14. Create an ER model for the following enterprise:
   Each building in an organization has a different building name and address. The meeting rooms in each building have their own room numbers and seating capacities. Rooms may be reserved for meetings, and each meeting must start on the hour. The hour and length of use are recorded. Each reservation is made by a group in the company. Each group has a group number and a contact phone.

15. Create an ER model for the following enterprise:
   The Lafayette Park Tennis Club teaches tennis and offers both private and group lessons. The Club charges $45 per hour per student (or couple) for private lessons and $6 per hour for group lessons. Students must preregister for private and group lessons. Each individual lesson has an instructor. Each group lesson must have two instructors. The Club also has weekly tournaments that can be attended by any member of the club at a cost of $5 per person. The Club would like to have an information system to keep track of lessons, students, and the schedules of lessons and instructors. It would also like to record the number of attendees in the tournaments and the amount collected; it is not interested in recording who plays, however.
16. Consider the following ER diagram and attributes:

```
Buyer: id, name, address
Property: id, askingPrice, salePrice, address
StaffMember: id, lastName, firstName, phone
Branch: id, name, address, phone
```

a. Write sentences (in English) to describe the roles of staff members in the diagram.
b. Write sentences to describe the roles of property in the diagram.
c. Can the amount of sales for a branch be calculated?
d. Can a property have more than one buyer?
e. Can a property be sold more than once?

17. Consider the following ER diagram:
a. Add attributes to complete the conceptual model. Include keys.
b. Write sentences to describe the roles of sections in the diagram.
c. Does every student have to take a section to complete the corresponding course?
d. Can a teacher teach more than one section of the same course?
e. Does the Section class need a unique key? Why or why not?

18. Develop a full ER diagram for one of the aspects of the BigHit Video information system, as listed below. Include all appropriate attributes of the primary entity classes and relevant attributes of all related classes.
   a. Employees, their time cards and pay statements, and the stores in which they work
   b. Customers, rentals, history of rentals, videotapes, and movies
   c. Suppliers, purchase orders, movies, and videotapes

19. Develop a full ER diagram for an information system that records student records. Include the registration of students for courses, the list of class offerings, the students, and their grades.

Further Readings

Discovery and modeling of information systems are described in Herbst [Her97] and Teorey [Teo94]. The original paper on ER modeling is [Chen76]. Several books have extensive discussions of ER modeling, including Elmasri and Navathe [ElNa99]; Date [Date99]; Batini, Ceri, and Navathe [BCN92]; Benyon [Ben90]; and Howe [Howe89].