CHAPTER 4

Data Modeling with Entity-Relationship Diagrams

In this chapter, you will learn:

- What entity-relationship data modeling is
- How to read and draw entity-relationship (E-R) diagrams
- How to represent entity classes, attributes, and relationship types in E-R diagrams
- How to add cardinality and other constraints to E-R diagrams
- How to model video rentals and other BigHit Video information using E-R diagrams
- What inheritance is and how to use it to improve data models
- How to represent inheritance in entity-relationship diagrams

In Chapter 3, you learned what data models consist of and why they are valuable. In this chapter, you'll learn about a special data-modeling method called entity-relationship modeling.

Entity-relationship modeling is a high-level data modeling technique that helps designers create accurate and useful conceptual models. E-R models are best expressed using graphical E-R diagrams. This technique was originally developed by Professor Peter Chen to serve as a tool for communication between designers and users.

Chen recognized the problems that are caused when developers and users fail to understand each other. It is typical for developers and users to think that they each know exactly what the other is thinking. Unfortunately, human communication is not that good. In the presence of misunderstanding, developers build information systems that do not meet user needs. The result is either the total failure of the system that was developed or a major increase in costs as the system is rewritten.

E-R diagrams provide a visual, graphical model of the information content of a system. Developers create E-R diagrams that represent their understanding of user requirements. Users then carefully evaluate the E-R diagrams to make sure that their needs are being met.

Once the E-R diagram has been approved by the user community, the diagram provides the specification of what must be accomplished by the developers. In the presence of accurate models, developers can be confident that they are building useful systems.

Without a precise description of the agreement between users and developers, the system is doomed to failure. Our goal, then, is to produce a data model that is understandable to users and that accurately and precisely describes the structure of the information to be stored in the database.

4.1 ENTITY-RELATIONSHIP MODELING

An **entity-relationship (E-R) data model** is a high-level conceptual model that describes data as entities, attributes, and relationships—all terms you encountered in Chapter 3. The E-R model is represented by E-R *diagrams* that show how data will be represented and organized in the various components of the final database. However, the model diagrams do not specify the actual data, or even exactly how it is stored. The users and applications will create the data content and the database management system will create the database to store the content. In this chapter, we'll focus on E-R diagrams; Chapter 5 will take a closer look at how E-R diagrams are represented as tables in a relational database.

This modeling method pays particular attention to relationships—the interactions among entities. Relationships require special treatment in the development of databases, because they are the glue that holds information together and because their realization in relational databases is particularly important. Moreover, an E-R model is usually accompanied by a *behavioral* model, which describes the way that the applications of the information system must behave. Database developers create these two models together.

An E-R model attempts to capture those aspects of the real environment that are necessary for the proper functioning of a business or other system. Not everything about the real environment can be captured by the E-R model. For instance, it will not be possible to design a data model that requires that no employee work more than 40 hours per week. Rules like this one must be enforced through the behavioral model of the system.

The data modeling process is iterative: You start putting some ideas together, then the process reveals some problems in your thinking. So, you go back and rework your ideas. You keep doing this until it all makes sense. In other words, a full, accurate E-R model doesn't just spring fully formed from a designer's mind and appear on paper.

This chapter walks you through a condensed version of the modeling process—including encountering some thinking problems and some dead ends. The primary examples come from the BigHit video rental business. The data model for this business will emerge through a series of decisions and specifications. In more than one case, we will find errors in the model. Fixing those errors will require modifying, or even destroying, E-R diagrams so that the final diagrams represent an accurate view of the video rental business.

As you see these diagrams and draw some yourself, you will recognize the need for E-R drawing tools. Professional information system developers always use E-R diagramming tools. Organizations that are serious about achieving useful systems adopt a variety of software development tools to make their jobs easier and their results more reliable. You can feel confident that your ability to draw these diagrams by hand will easily translate to understanding and using diagramming tools.

4.2 ENTITY-RELATIONSHIP DIAGRAMS

As one important aspect of E-R modeling, database designers represent their data model by **E-R diagrams**. These diagrams enable designers and users to express their

understanding of what the planned database is intended to do and how it might work, and to communicate about the database through a common language.

The myriad of styles and conventions for E-R diagramming make it difficult to choose the best one. This book utilizes an acceptable style, but certainly not the only one. Each organization that uses E-R diagrams must adopt a specific style for representing the various components. While studying this book, you should use the style presented by the text. You can be sure that the principles of E-R diagramming are independent of the stylistic details.

4.2.1 Entity Classes and Attributes

Figure 4.1 shows an E-R diagram that describes the entity class `Customer`, as previously shown in Table 3.5. In this E-R diagram style, entity classes are represented by rectangles and attributes by ovals. Solid lines connect attributes to the entity class. Note that the diagram shows only entity class and attributes—it does not describe any particular *instances* of either. The shaded boxes are comments, and would not be parts of the E-R model diagram.

Figure 4.1 illustrates the basic shapes used for describing entity classes and their attributes. In the center of the diagram is the rectangle that represents the `Customer` class. Single-valued attributes `lastName`, `firstName`, `accountId`, and `balance` are shown as ovals with connecting lines. The `accountId` attribute is underlined to show that it is the key of the class.

Additional characteristics of attributes are distinguished by their display. The multi-valued attribute `otherUsers` is shown as an oval whose border is a double line. The derived attribute `numberRentals` is shown with a dashed border.

The composite attribute `address` is shown as the central point of connection for its component attributes `street`, `city`, `state`, and `zipcode`.

You may have noticed that much of the detailed information that you saw in Tables 3.1 and 3.4—such as descriptions of classes, relationships, and attribute constraints—is not shown in Figure 4.1. Instead, this information would be maintained in text form as part of the database specifications, in a table called the **data dictionary**.

FIGURE 4.1
E-R Diagram for Entity Class **`Customer`**

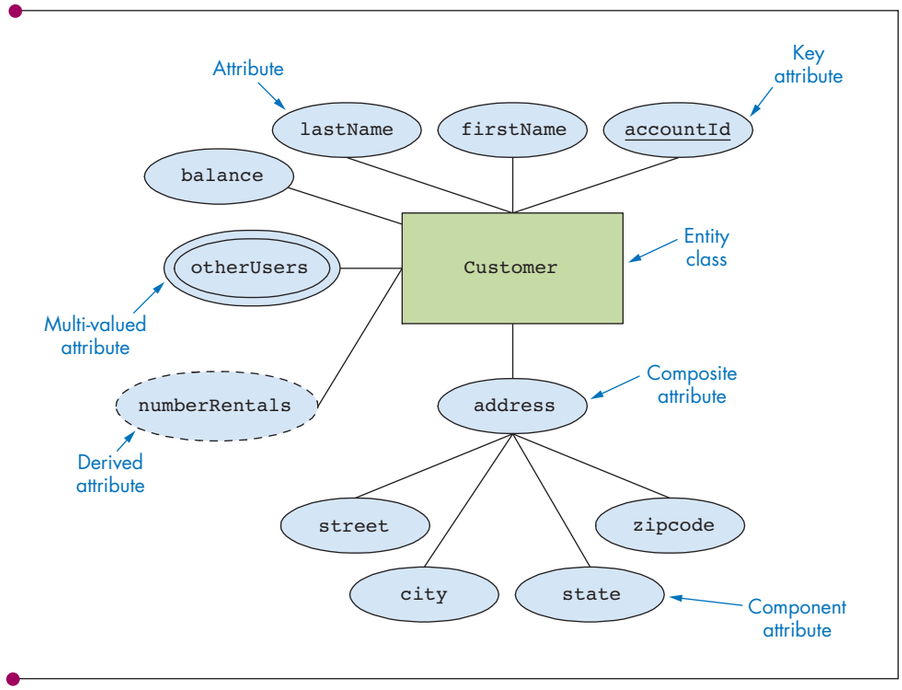


Figure 4.2 shows the E-R diagram for entity class `Video`. It uses the same style as `Customer`. Each of the attributes of class `Video` are simple and single-valued.

4.2.2 Relationship Types and Constraints

Figure 4.3 shows the relationship type `Owns` that connects classes `Store` and `Video`. A relationship type is represented by a diamond and two connecting lines. The name of the relationship is shown inside the diamond. Reading this diagram from left to right yields "a store owns a video." The name of the relationship type is a verb or verb phrase that describes the relationship. The alternate relationship name `IsOwnedBy` is the name of the relationship type when reading the other direction. That is, "A video is owned by a store." You will notice that the attributes of `Store` and `Video` have been omitted.

Additional symbols have been added to the relationship type in Figure 4.4 to express the cardinality and participation constraints of the relationship type. The relationship type is one-to-many. That is, a store may own many videos and a video may be owned by no more than one store.

Cardinality constraints are represented in Figure 4.4 by the symbols `1` and `M`. The symbol `1` in Figure 4.4 means that a video can be owned by no more than one store. The symbol `M` means that a store may own many (zero or more) videos. That is, these

FIGURE 4.2
E-R Diagram for Entity Class **`Video`**

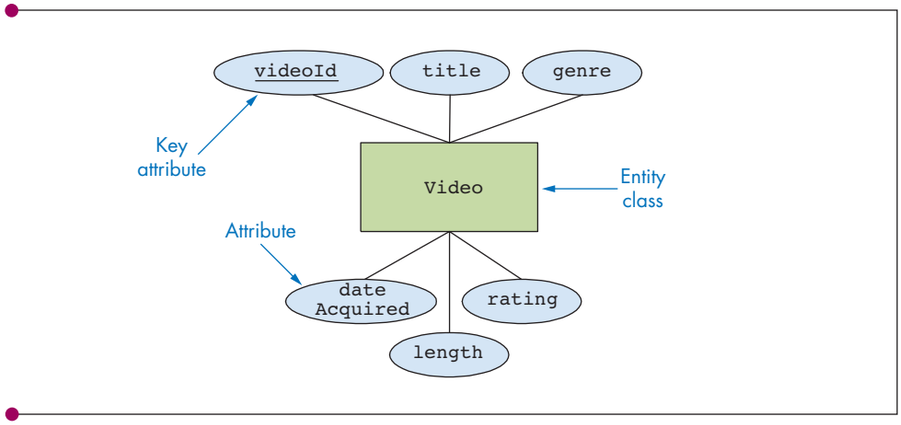
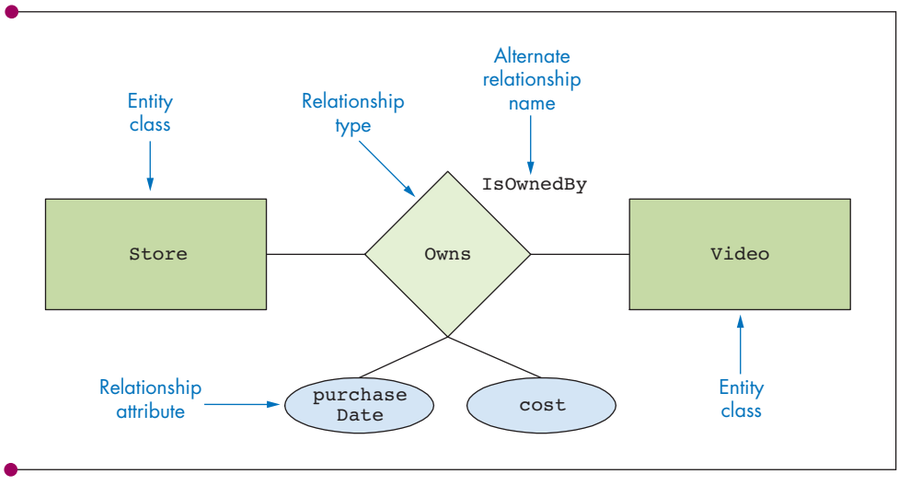


FIGURE 4.3
E-R Diagram for the **`Owns`** Relationship Type



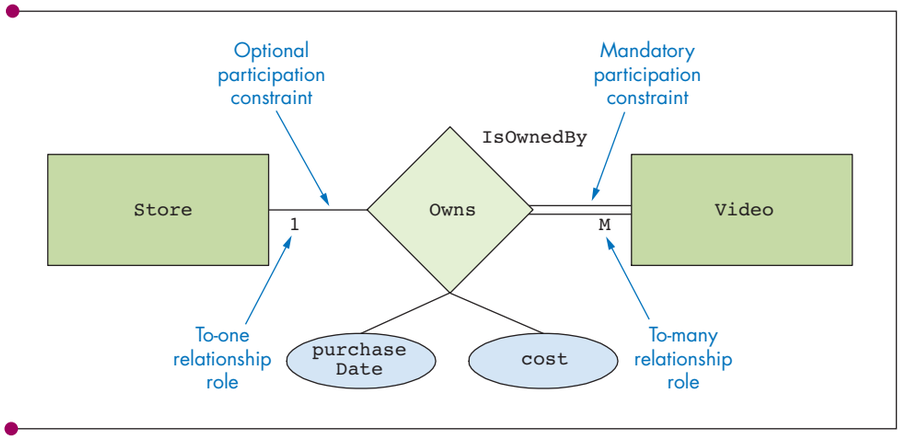


FIGURE 4.4
E-R Diagram for the **Owns** Relationship Type Showing Cardinality and Participation Constraints

two symbols together define the relationship type as one-to-many. The constraint symbols tell you how many entities of the *related* class (the class near the constraint symbol) can be associated with one entity of the *subject* class (the class on the other side of the diamond).

We can best understand the meaning of these cardinality marks by creating sentences to represent the relationship roles. To create such a sentence, name one entity, then the relationship, then the cardinality, and finally the other entity. For example, we could create the following two sentences by reading the diagram from left to right and right to left, respectively:

- A store may own many (`M`) videos.
- A video is owned by no more than one (`1`) store.

Again notice that the symbol that tells us how many relationships a store is allowed to have is on the opposite side of the diamond from the store class. The position of the cardinality mark makes it easy to create the sentence that makes the constraint understandable, as shown above.

You've probably also noticed that the constraint symbols can appear either above or below their double lines. They simply must be located near them.

This relationship type also has participation constraints. A store is not required to own any videos, and hence has *optional participation*. A video must be owned by a store and hence has *mandatory participation*.

Figure 4.4 adds these participation constraints to the E-R diagram for the `Owns` relationship type using single and double lines. The line between `Store` and `Owns` is single to specify that a store does not need to participate by being related to (owning) any videos. The line between `Video` and `Owns` (or `IsOwnedBy`) is double to specify that each video must participate by being related to (owned by) a store.

Again we can create sentences to understand the constraints. We add either "may" or "must" to the sentence depending on whether the participation is optional or mandatory. These are the sentences:

- A store may own many videos.
- A video *must be owned* by one store.

The first sentence, about the constraint on stores, has not changed, and the diagram didn't change either, because the line was already single. The second sentence, about the constraint on videos, has changed. The phrase "may be owned by no more than one" has become "must be owned by one." If the cardinality constraint on videos were to-many, the sentence would be "A video must be owned by at *least* one store."

4.3 MODELING VIDEO RENTALS

We are now ready to look at more complex objects and relationships. We will take on the renting of videos by customers. We begin with a description of the classes and relationship types and then produce an E-R diagram.

The rental of a video is an agreement between the BigHit company and a customer that the customer will be given possession of the video for a particular period of time for a particular cost. At BigHit Video, we need to keep track of our videos. We need to know whether the video is currently rented, and if so, who has it and when it is due back.

4.3.1 Modeling Rentals as a Relationship Type

There are two different ways to represent video rentals. The first is to identify the rental as a relationship between a customer and a video, as shown in Figure 4.5. Relationship type `Rents` links classes `Customer` and `Video` in a one-to-many relationship. A customer may rent many videos, but a video can be rented by no more than one customer. Both of the relationship roles have optional participation (connected by a single line) since a customer doesn't have to have any rentals and a video may be unrented, and thus available for rental.

A relationship between the customer and the video is created when the customer rents the video. The relationship is removed when the customer returns the video. This accurately models the movement of the video, the behavior of the customer, and the business rules of BigHit video. The relationship is created when the customer accepts possession of the video. The relationship ceases to be of primary interest to BigHit Video when the video has been returned to the store.

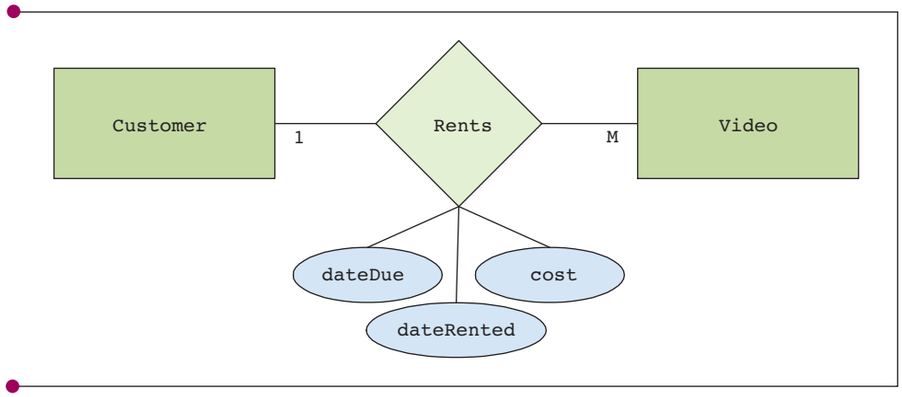
Once the video has been returned by the customer, we will keep track of that previous rental as shown in Figure 4.6. The relationship type `PreviouslyRented` is many-to-many because each customer may have previously rented many videos and each video may have been rented many times. The participations are optional for both customers and videos.

The BigHit information system will create a relationship when a customer rents a video. The relationship will be stored in the database. When the video is returned, that relationship will be removed from the database and a permanent `PreviouslyRented` relationship will be created and stored in the database.

4.3.2 Modeling Rentals as an Entity Class

An alternative view is that a rental is an entity and not simply a relationship. Note that we have been referring to a "rental" as an object of interest. The use of the noun

FIGURE 4.5
E-R Diagram of Relationship Type `Rents`



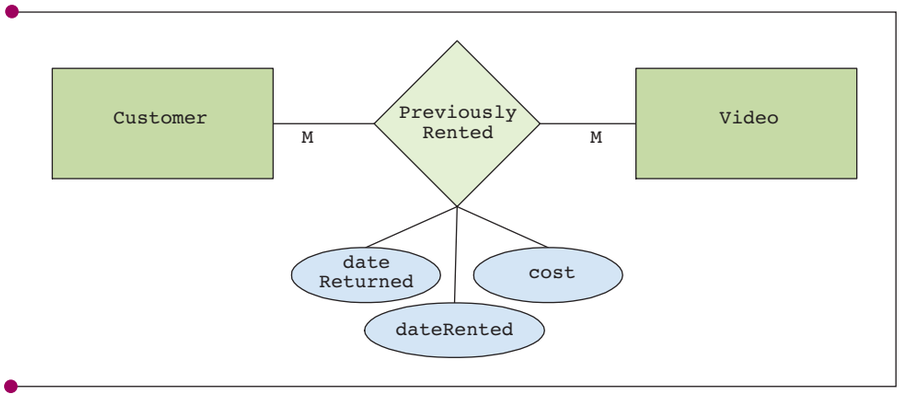


FIGURE 4.6
E-R Diagram of Relationship Type **Previously Rented**

"rental" suggests that we are referring to an entity. You will recall that entity classes are named by nouns and relationship types are identified by verbs. Hence, it is very natural to think of a rental as an entity that has an existence distinct from the customer and video that it connects.

Figure 4.7 shows the `Rental` entity class and its relationship types with classes `Customer` and `Video`. Each rental entity has three attributes: `dateDue`, `dateRented`, and `cost`. The relationship types are called `Has`. This is a name that can be used for a relationship type whose meaning is obvious. These are the relationship sentences for the diagram:

- A customer may have many rentals.
- A rental must have one customer.
- A rental must have one video.
- A video may have no more than one rental.

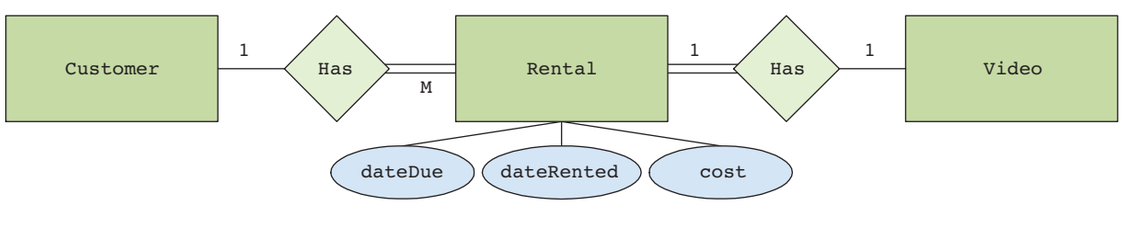
Thus each rental entity is related to one customer and one video just as it should be according to our understanding of what a rental is.

One difficulty with understanding the meaning of `Rental` entities is the problem of distinguishing between one rental and another solely based on attribute values. That is, what is the key of a rental? None of the attributes is unique and not even all of the attributes together form a unique key. For instance, a customer may rent two different videos at the same time. This may result in two rentals with the same date rented, the same cost, and the same due date. We can only distinguish between the two rentals according to which video each is related to.

We have an entity with no distinguishing attributes that is partly identified by its relationships to other entities. We call this a *weak entity* and class `Rental` a **weak entity class**. Figure 4.9 shows how weak entity classes are shown in E-R diagrams.

Two items in the diagram now have double-lined borders. The border on entity class `Rental` has been doubled to specify that `Rental` is a weak class. The border on

FIGURE 4.7
An E-R Diagram Showing Entity Class **Rental** and Its Relationship Types



Concept

Distinguishing Between Current and Previous Rentals

The cardinality constraint on videos in the `Rents` relationship type of Figure 4.5 shows that we must be modeling current rentals. Each video can have only one `Rents` relationship with a customer. Before the video can be rented by a different customer, the relationship must be deleted so that the video is available for rental.

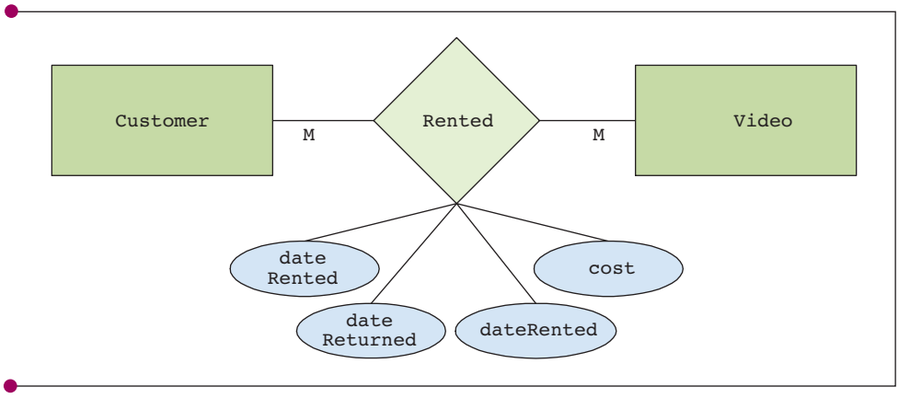
Obviously a video may be rented over and over. The BigHit Video business depends on collecting money for renting each video many times. However, a major distinction exists between videos that are currently rented to customers and those that have been rented and returned. The record of previous rentals is important for analyzing customer behavior and video usage. However, the current rentals are used to determine where videos are located, who has them, and how much the renters owe in overdue fees. Our data model makes a clear distinction between those two very different relationship types.

The cardinality constraint exposes the distinction between current and previous rentals, and thus demonstrates one of the important features of modeling constraints: Being specific about constraints exposes questions about the meaning of entities and relationships that must be answered in the data model.

Finally, it is worth noting that we can combine the current and previous rentals into a single relationship type, as in Figure 4.8. In this model, a current rental is one in which the `dateReturned` is null. The major problem with this model is that it fails to distinguish between relationships that have a very different meaning and importance to the business, as described above.

This example serves to emphasize that one of the most important goals of data modeling is to accurately express the meaning of information as it is understood by the users of the system. We as developers may prefer the model of Figure 4.8 because it has fewer relationship types. But if it is difficult for users to understand, or if it is inconsistent with the users' understanding of their information, it is not the right model. For BigHit video, Figure 4.8 is less accurate and harder to understand and explain.

FIGURE 4.8
E-R Diagram of a Combined Current and Previous Rental Relationship Type



the `Has` relationship type with class `Video` has been doubled to specify that it is the **identifying relationship type** of class `Rental`. This notation means that a rental is identified by its relationship to a video and cannot exist without it. An identifying relationship type is always to-one.

With these two double borders, we have completely specified the nature of the class. No two rentals can be related to the same video because of the cardinality constraint on class `Video`. Thus we can distinguish two rentals by their relationships with videos.

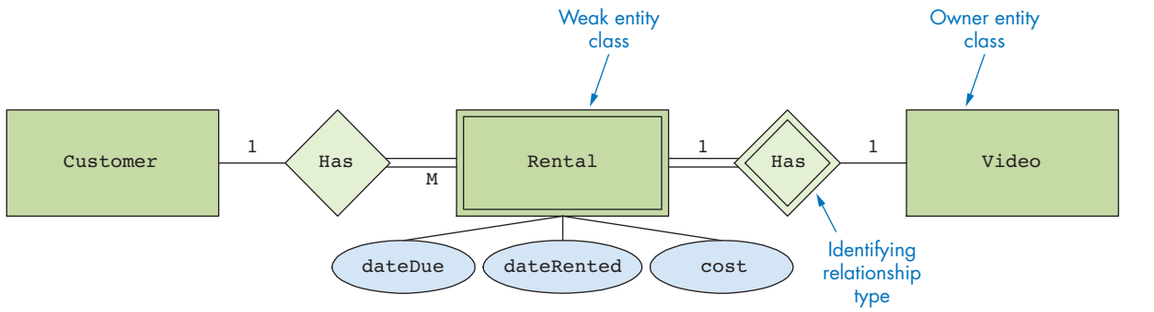


FIGURE 4.9
An E-R Diagram Showing Weak Entity Class **Rental**

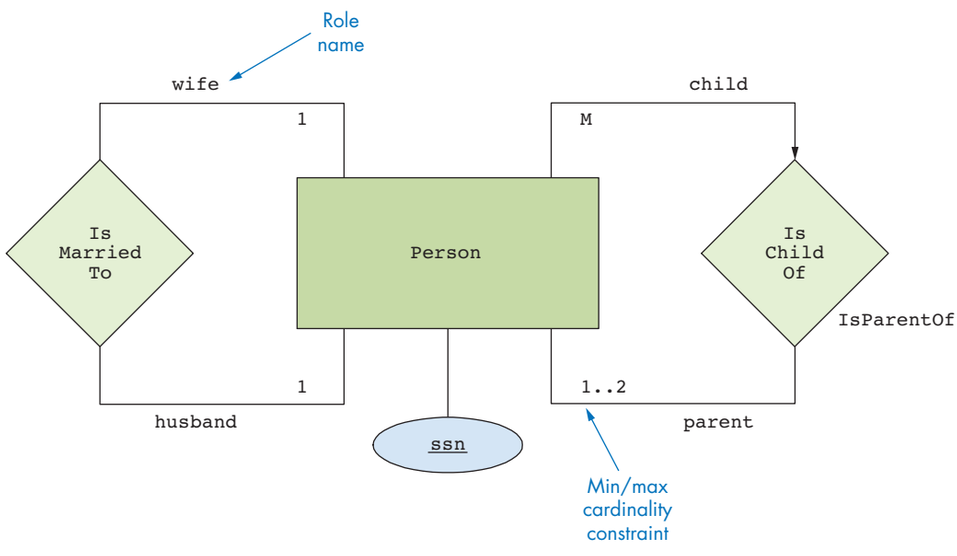
The double border on the diamond that relates `Rental` to `Video` also marks `Video` as the *owner entity class*. The related video is considered the *owner* of the rental. Without that relationship, the rental cannot exist. The *key* for a `Rental` entity is therefore the `videoId` attribute, which is also the key for its owner video.

4.4 ROLES IN RELATIONSHIP TYPES

As you saw in Chapter 3, each entity in a relationship plays a specific *role* in it. For example, in the `Rents` relationship type of Figure 4.5, the customer is the renter of the video, and the video is rented by the customer. The customer plays the "renter" role and the video plays the "rented by" role. We'll take a brief break from modeling BigHit Video information and consider some interpersonal relationships among people.

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, consider the traditional marriage relationship. It relates one person to another, as shown in Figure 4.10. This diagram shows the entity class `Person`,

FIGURE 4.10
Relationship Types **IsMarriedTo** and **IsChildOf**, with Role Names



with its key attribute `ssn`, and the relationship types `IsMarriedTo` and `IsChildOf`, with their lines and cardinalities. The names of the roles `wife`, `husband`, `child`, and `parent` are shown in the diagram next to the relationship-type lines. For each pair of people related by parentage, one is the child and one is the parent.

Additional information from the figure is expressed by the following sentences:

- A person may be the child of one or two parents.
- A person may be the parent of zero or more children.
- A person may be the wife of a husband.
- A person may be the husband of a wife.

You can create these sentences by reading along the relationship lines in the diagram. Read the entity class, the role, the cardinality of the role, and finally the other role.

Note the unfamiliar cardinality symbol `1..2` on the `parent` line. This symbol means that the child may have between one and two parents. Figure 4.10 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 pointing indicates that the name of the relationship inside the diamond refers to this role's relationship name.

Concept

Overspecified Cardinalities

Perhaps you've seen a flaw in the cardinalities shown in Figure 4.10. If every person must have one or two parents, either there are infinitely many persons—or some people are their own descendents. This is a common error created by overspecifying the cardinality of a relationship type. In this case, we must recognize that this relationship type represents parent–child relationships between people whose names will be included in this database. Some people in the database will have parents who are not part of the system. Hence the child must be allowed to have no parent.

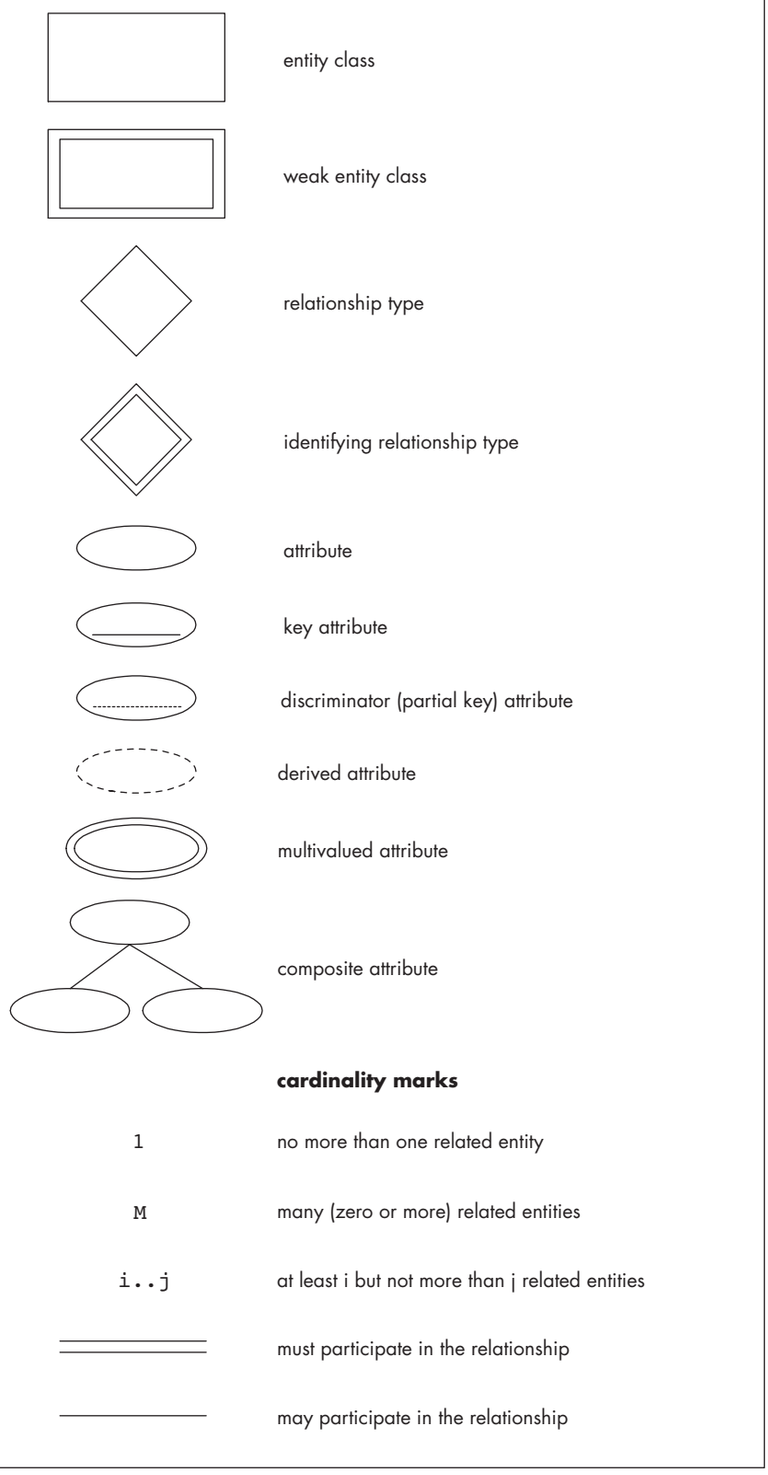
Remember: All data models and all databases attempt to represent just a small part of the world. The cardinality of a role need only represent the number of related entities that are relevant to the planned database.

Although E-R diagrams include declarations of cardinality constraints, many other constraints cannot be represented in this format. The E-R diagram for the `IsChildOf` relationship type restricts the cardinality of the child so that a child has no more than two parents. However, it does not place any restrictions on which individuals are the parents. According to this model, it is possible for Joe to be a child of Jane and for Jane to be a child of Joe at the same time. The diagram also allows Joe to be a child of Joe! Unfortunately, such nonfactual relationships cannot be excluded from an E-R diagram. Instead, constraints on individual membership in relationships must be written down as part of the E-R modeling process. Developers will consider these later in the database- and application-development processes.

Figure 4.11 shows examples of basic symbols used to draw E-R diagrams. Almost as many styles of E-R diagramming exist as organizations that draw them—and there's no best one. The style presented in this chapter is used by many developers. Each designer works within or for an organization and must draw diagrams using symbols that conform to the style conventions established by that organization. These symbols 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 database designers adopt a specific E-R style, it is their responsibility to make sure their users understand it.

FIGURE 4.11
Examples of E-R Diagram Symbols



4.5 AN E-R MODEL FOR BIGHIT VIDEO

Now that you've become familiar with the symbols used in E-R modeling, let's see what a more comprehensive E-R model for BigHit Video might look like. Figure 4.12, a conceptual schema, shows one possibility. It depicts all of the entity classes you saw in Table 3.1, along with their relationship types. The diagram is incomplete, because it omits the attributes of most of the classes. You will be invited to finish the attribute and entity-class definitions as part of the exercises at the end of this chapter.

In this section, you'll discover more about what information this conceptual schema can and cannot represent. By carefully studying this diagram, you can gain insights into many issues that arise during the discovery and specification processes. The figure also illustrates how much detail can be specified by an E-R model and how that detail constrains the resulting database.

4.5.1 Recording the History of Rentals

The entity classes `Customer`, `Rental`, and `Video` are shown as related in Figure 4.12. As noted earlier, `Rental` entities represent the current state of video rentals. When a video is returned, the corresponding `Rental` entity is removed from the database.

But what if BigHit wanted to record historical information about its business? For example, suppose it wanted to determine which videos are being rented most often and what types of videos a particular customer tends to rent. The entity class `Rental` does not provide the information required for these analyses.

The entity class `PreviousRental` of Figure 4.13 has been added to record the history of customer and video rentals. The difference between `Rental` and `PreviousRental` lies in the cardinality constraints of the relationship with `Video`. Class `Rental` has a one-to-one relationship with `Video`, whereas `PreviousRental` has a many-to-one relationship. This relationship allows the database to record many rentals of each video.

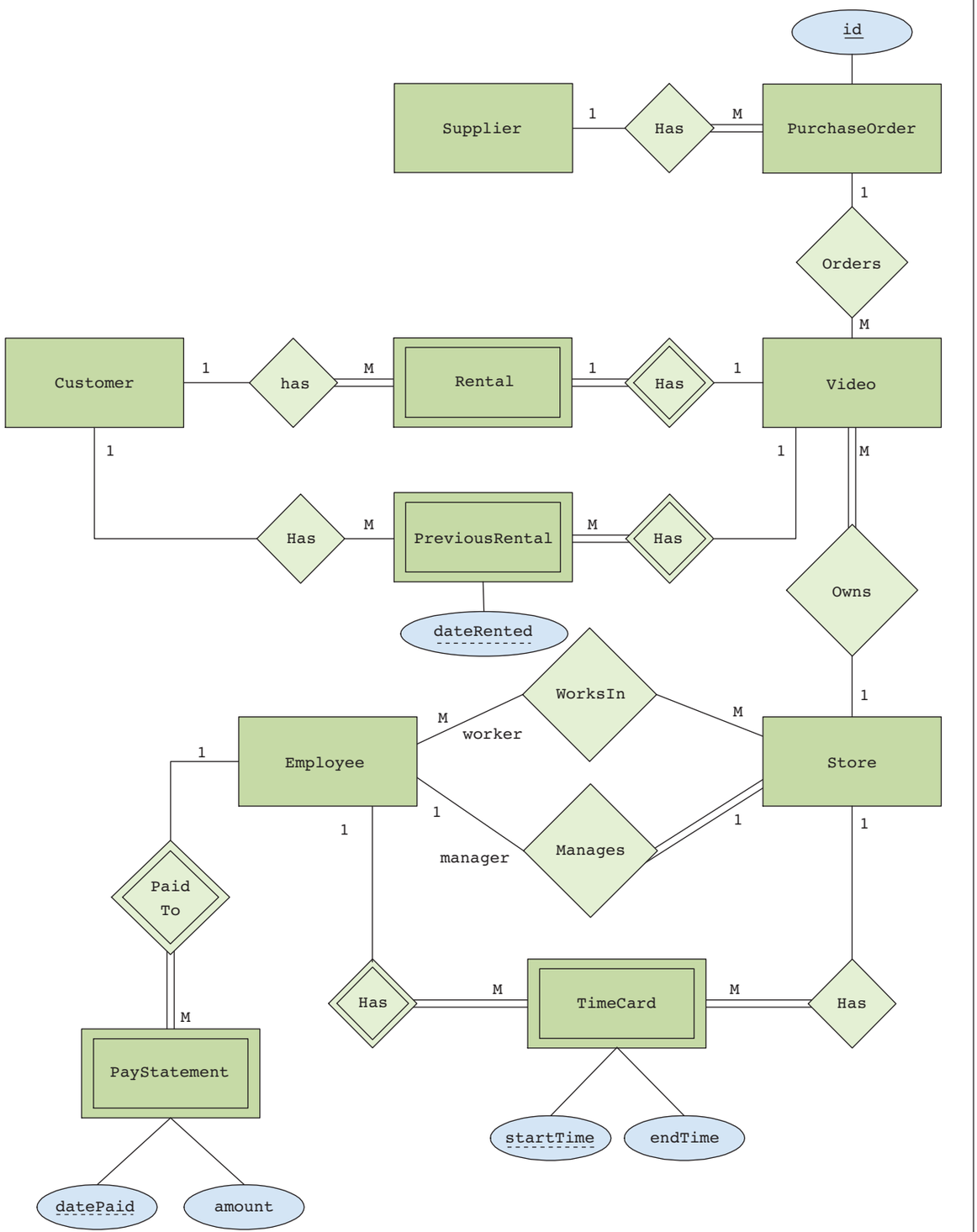
The entity class `PreviousRental` is a weak entity class with a single identifying relationship. Unfortunately, the related video does not uniquely identify the `PreviousRental`. This property is another difference between `Rental` and `PreviousRental`. Because a video can participate in at most one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

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 video together form a unique identification. The `dateRented` attribute *discriminates* among all of the previous rentals for a particular video. The partial key is shown with a dashed underline in Figure 4.11.

This definition of the key of entity class `PreviousRental` may create some problems for BigHit Video stores. In particular, under this definition, a video 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, BigHit's database designer could create an artificial key `rentalId` to serve as the key of the entity class.

But BigHit might also want to allow a `PreviousRental` entity to have no associated customer. As shown in Figure 4.14, the database designer would place a single line between `PreviousRental` and its relationship with a `Customer`. With this cardinality, managers or clerks at BigHit could delete inactive customers and the records of their previous rentals without compromising the company's ability to analyze rental activity.

FIGURE 4.12
An E-R Diagram for BigHit Video



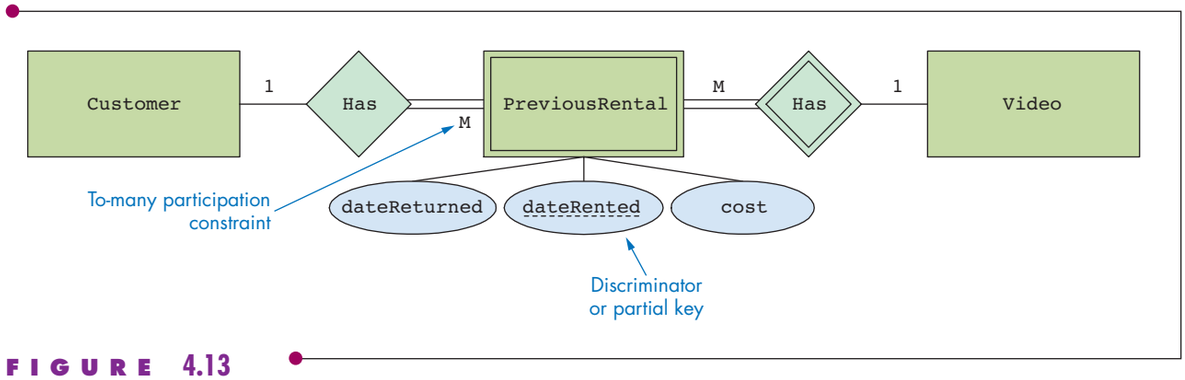


FIGURE 4.13
E-R Diagram for Entity Class **PreviousRental**

This strategy for handling rentals and previous rentals with separate entity classes exposes some limitations of E-R models. We have defined the relationship type between `Rental` and `Customer` as one-to-one, which means that a video can have at most one current rental. However, when a video is returned, the `Rental` entity must be deleted and a similar `PreviousRental` created. In some sense, the `Rental` is transformed into a `PreviousRental`. Unfortunately, the E-R diagram has no way to represent this required transformation.

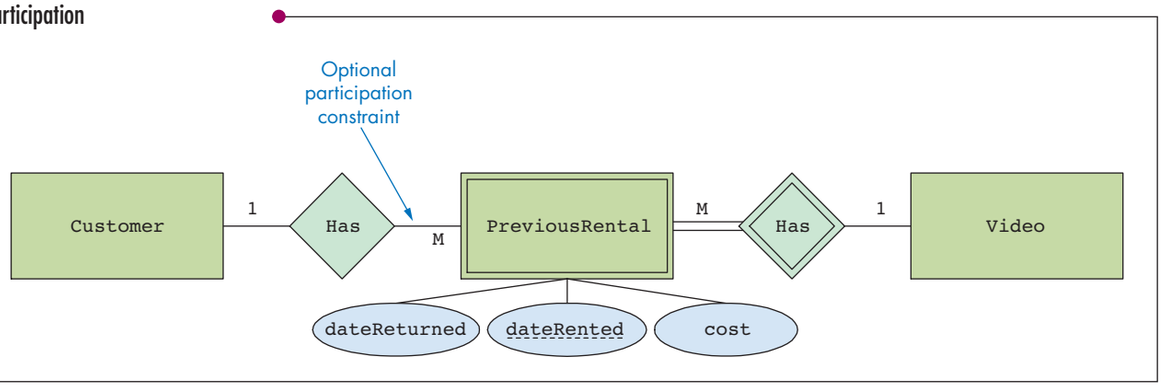
As described in the Concept box in Section 4.3.1, an alternative way of representing rental history would be to have a single `Rental` entity class with a one-to-many relationship type with `Customer`. With this strategy, the current rentals would consist of all rentals with a null return date. However, the E-R diagram would be unable to depict the requirement that there be at most one current rental. A database designer must decide which is most important: ensuring that there is at most one current rental or ensuring that the previous rentals are reliably created by the information system when videos are returned. The model shown in Figure 4.14 is a compromise between these two goals.

4.5.2 Employee Roles and Cardinalities

Now let's look at the relationship types between `Employee` and `Store`. As you might see in Figure 4.15, two such types exist. An employee can be associated with a store as manager or as worker. These employee roles are shown next to the lines that connect `Employee` and its relationship types. This 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 one 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 pose a problem if the

FIGURE 4.14
E-R Diagram for Entity Class **PreviousRental** with Optional **Customer** Participation



Concept

Customer Privacy in the BigHit Information System

The idea of recording previous rentals raises the issue of customers' privacy. Some people may consider it a violation of their privacy for a company such as BigHit Video to keep track of all the videos they've ever rented. Employees of BigHit Video could search the database to find out private information about customers. Public libraries avoid these accusations by keeping this kind of information private. U.S. courts have supported libraries' right to maintain the privacy of their circulation records. BigHit Video is not under the same constitutional constraints. However, it may not even want to record a history of individual customers' 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 videos.

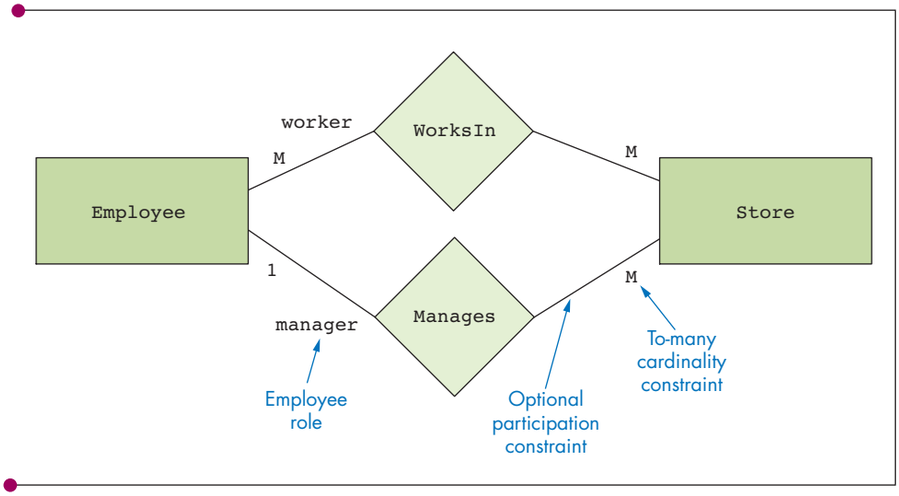
business needs one person to manage more than one store. For example, if the company fires a store manager, there will likely be a period of time when there is no manager or some other manager temporarily fills the vacancy. The database would be unable to represent this situation. The information system would have to add a fictional employee to the `Employee` class and associate that employee with the store as manager. An alternative approach, as shown in Figure 4.15, would be 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 the kinds of issues that can arise from real business practices. It is users' responsibility to determine whether the E-R model adequately represents their enterprise and meets its needs. It is developers' responsibility to find and expose the questions that always arise as part of the discovery and specification processes.

4.5.3 Purchase Orders and the True Meaning of Video

Let's now take a closer look at how to handle weak entity classes in an E-R diagram. The entity class `PurchaseOrder` could be defined as weak. (Purchase orders refer to BigHit's ordering of videos from its suppliers.) The cardinalities of the relationships

FIGURE 4.15
E-R Diagram for **`WorksIn`** and **`Manages`** with Modified Cardinality and Participation Constraints



of class `PurchaseOrder` require that a purchase order have a single supplier and at least one video; hence a purchase order cannot exist unless it is related to other entities. This situation typifies a weak class. The natural key for `PurchaseOrder` is some combination of `date`, `supplierID`, and `items`. But 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 Figure 4.12 and repeated here as Figure 4.16.

A weak entity has no key of its own and cannot exist without being related to another entity. Another example from BigHit Video is the entity class `PayStatement`. This class has no key, because many employees may be paid on the same day. However, the relationship type `PaidTo` is a many-to-one relationship that links each pay statement to a single employee.

The relationship between a purchase order and a video indicates that the specific video is being ordered as part of the purchase order. A purchase order can be related to many videos—many DVDs and videotapes may be included in a single order. In contrast, a video can be related to only one order, because it is purchased only once. To create a purchase order, the information system must add an entity to the `Video` class for each item in the order. Each of these video entities is associated with a specific store. When the order is received, a clerk can use the information in the database to determine which store the DVDs and videotapes go to.

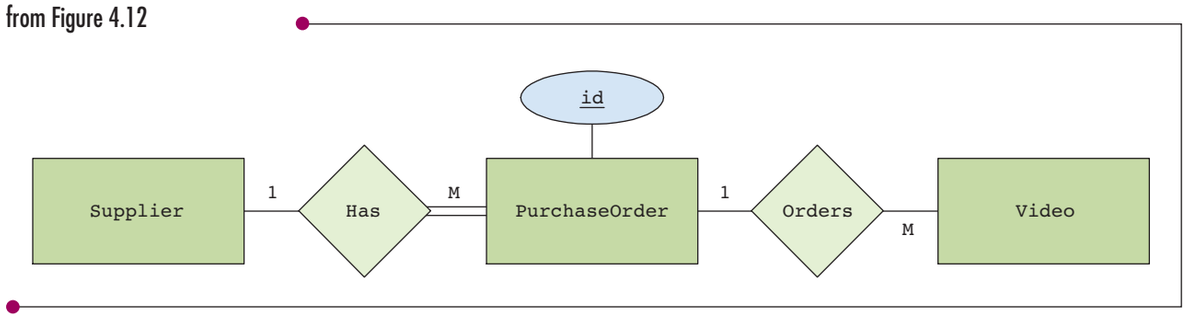
This strategy is not the usual approach to purchasing. Typically, a purchase order contains a list of items to be purchased. Each item has some identifying information (for example, a catalog number) and a quantity. In this case, the intention may be to purchase 25 copies of *Lady and the Tramp*. Figure 4.12 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 situation reveals a clear mistake in the diagram of Figure 4.16. To be correct, the conceptual data model 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 an actual purchase order.

We might be tempted to place a quantity attribute on the `Orders` relationship type, as shown in Figure 4.17. Now the purchase order is a set of items, each with a quantity. Yet in this model, the meaning of `Video` has changed. In Figure 4.12 , a `Video` entity represents a specific DVD or videotape that may be rented. In Figure 4.17, a `Video` entity represents a specific title, but not a specific physical DVD or videotape. A video can now be purchased multiple times and may represent more than one physical DVD or videotape.

This problem reveals another error as well—this one in the understanding of the nature of class `Video`. In the E-R diagrams of previous figures, two different entity classes have been confused as the single class `Video`. One class represents the physical video—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.

FIGURE 4.16
Model of Purchase Orders from Figure 4.12



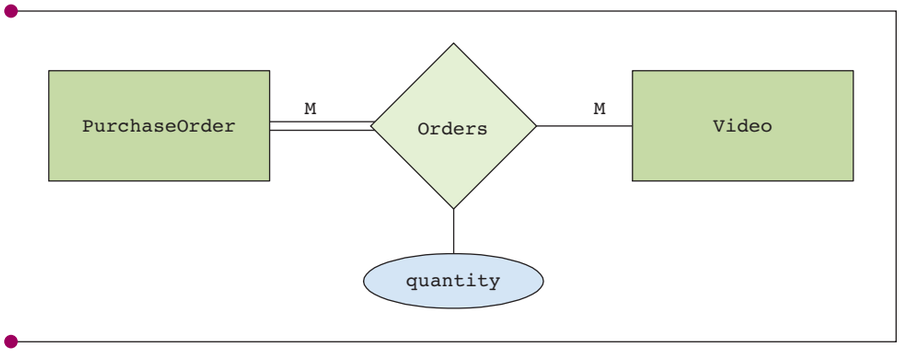


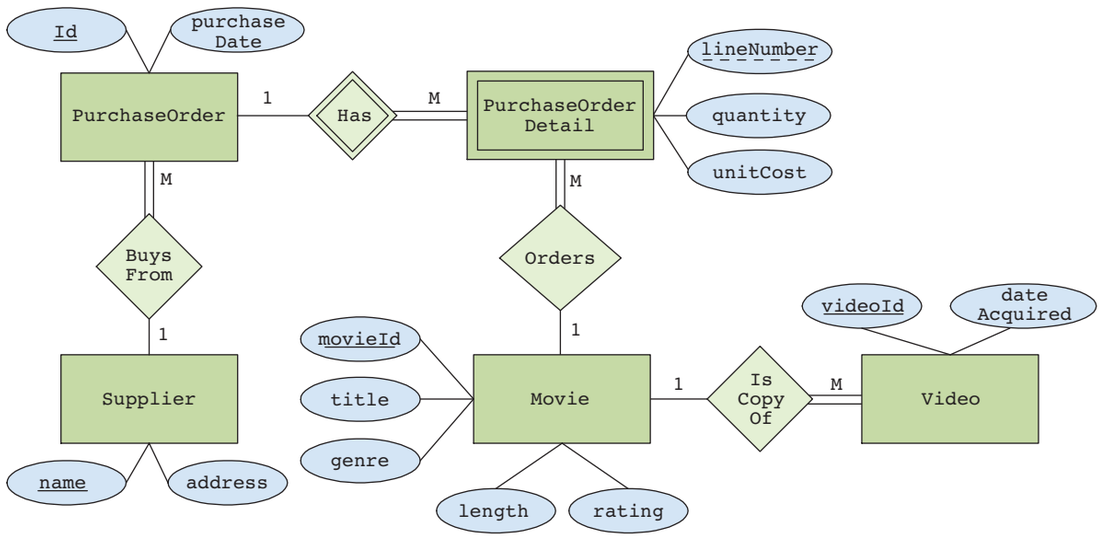
FIGURE 4.17
Alternative Representation of a Purchase Order

Each *physical* video is a *copy* of a specific movie. The movie itself must be represented by another entity class. It is this movie object that BigHit purchases from its supplier and that customers want to watch. After all, a customer doesn't ask "Do you have video number 112376?" He asks, "Do you have a copy of *Lady and the Tramp*?" Figure 4.18 gives a more appropriate E-R diagram for videos, movies, purchases, and sales. In this diagram, a video is designated as 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` entity 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 `Video` entities for each new item so that the clerk can enter the DVDs and videotapes into the rental inventory.

4.5.4 Employees, Time Cards, and Pay Statements

Are you ready to move on to another aspect of BigHit Video's business? Let's take a look at the entity classes `TimeCard` and `PayStatement` from Figure 14.12, repeated here as Figure 14.19. These weak entity classes record when employees work and what they are paid. In both cases, the entities are not uniquely determined by

FIGURE 4.18
E-R Diagram for Suppliers, Purchases, Movies, and Videos



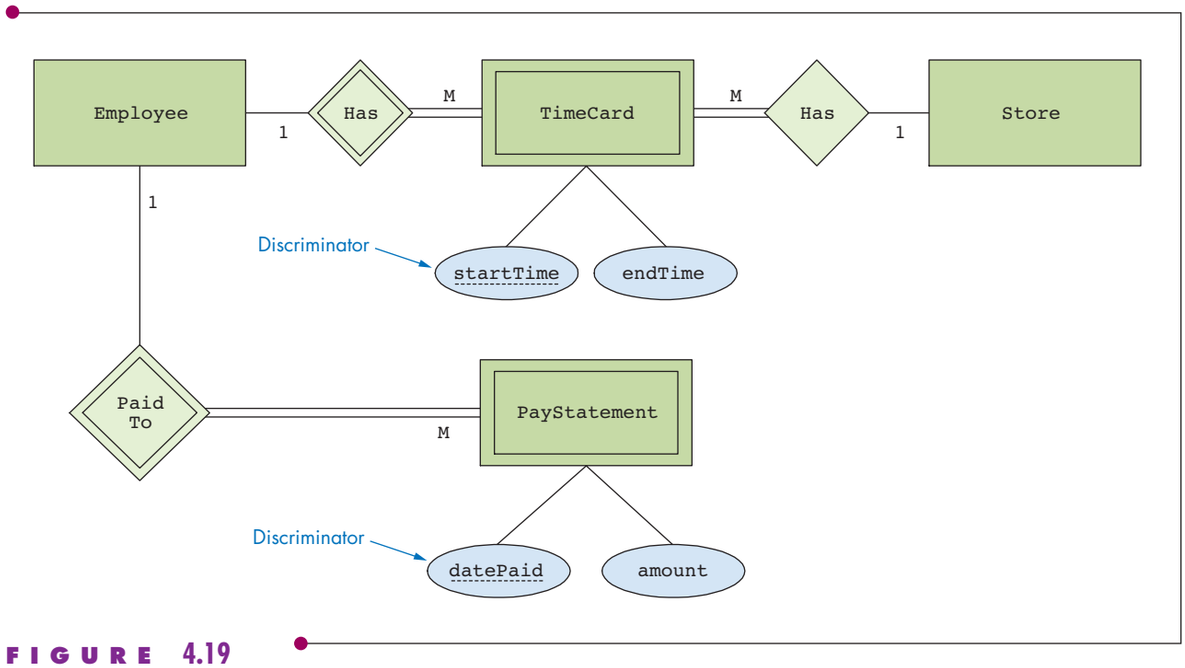


FIGURE 4.19

`TimeCard`, `PayStatement`, and Their Related Entity Classes from Figure 4.12

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 the employee's `ssn` and `datePaid` that is unique. Like `dateRented` of class `PreviousRental` in Figure 4.14, the attribute `datePaid` is called a discriminator (or 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. A database based on this E-R diagram cannot 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 and then changes to a different store must be represented by two time cards.

4.6 OBJECT-ORIENTED DATA MODELS

Thus far, we have considered the rentals of videos. But what about the difference between videotapes and DVDs? Should these differences be important to BigHit Video? These questions lead us to the topic of **object-oriented data models**. Such data models allow us to express the idea that a video can be either a DVD or a videotape, but not both. The concepts supporting object-oriented design will emerge from the consideration of these different types of videos. We will also see how E-R diagrams can be enhanced to directly express object-oriented designs.

Not surprisingly, the company must be able to tell whether a particular video is DVD or tape. Customers typically care which format they rent, so the company needs to store them in separate shelf sections. The database designer can easily make arrangements for the database to record the media type of a video by adding an attribute to the `Video` entity class. The attribute media has only two possible values: "dvd" and "videotape."

It is more difficult to represent attributes that are specific to one media. To illustrate, a videotape has a single soundtrack in a specific language and format. The format might correspond to either U.S. video standards or European standards. By

contrast, most DVDs have soundtracks in several languages, closed captioning, and possibly even multiple video formats. Figure 4.20 shows one way to add attributes to the `Video` entity class in order to maintain information specific to each media. The `media` attribute records whether a video object is a videotape or a DVD. Tape objects have values for the fields of the compound attribute `tape`. DVD objects have no values for the fields of `tape` but rather have values for the fields of `dvd`.

Unfortunately, Figure 4.20 does not specify any correlation between the value of `media` and the presence of values for `videotape` and `dvd`. We want to ensure that the attributes of videotape have non-null values only if `media` is `videotape`, and the attributes of `dvd` have non-null values only if `media` is "dvd." In order to guarantee accurate information in these attributes, we must make a clearer distinction between objects of these two kinds.

To clearly distinguish between videotapes and DVDs, we can create two new entity classes, one for each media type. We add the appropriate attributes to each class. You may be thinking that the simplest strategy is to replace class `Video` with two classes `Videotape` and `DVD`, as shown in Figure 4.21. But there is significant duplication in the attributes of these classes. And when we try to put these classes into the larger E-R diagram, as in Figure 4.22, an even worse problem arises. We have had to dupli-

FIGURE 4.20
E-R Diagram with **`dvd`** and **`videotape`** Attributes of Entity Class **`Video`**

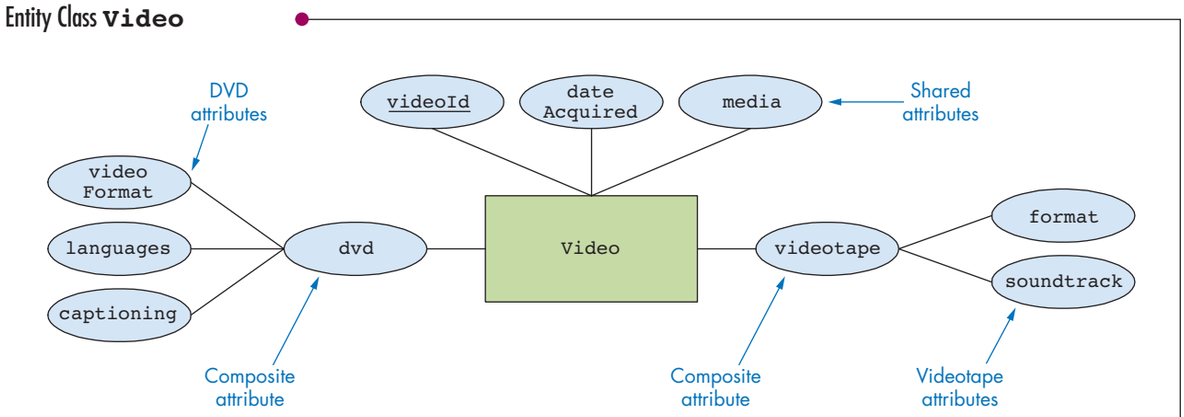
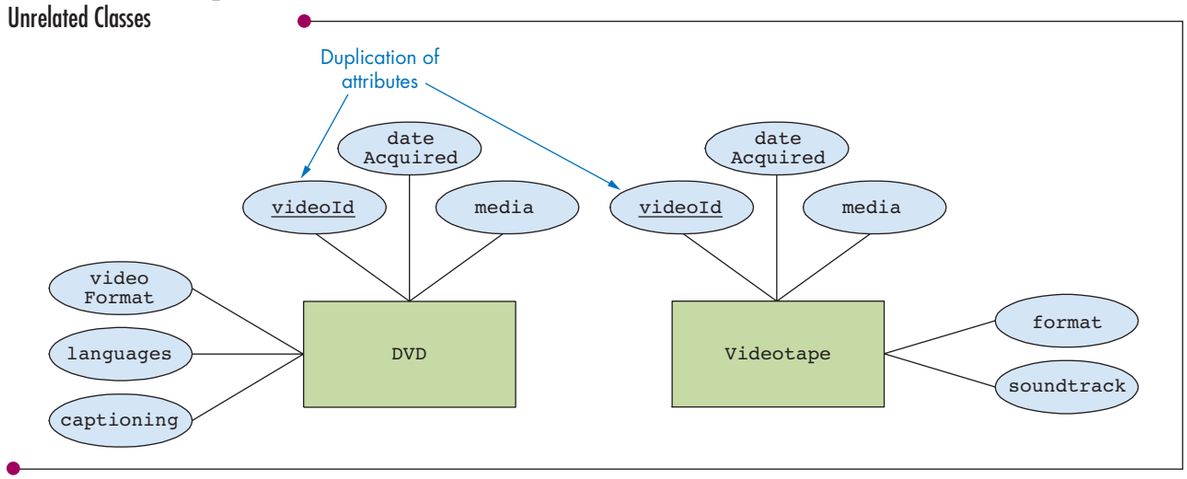


FIGURE 4.21
E-R Diagram Showing Classes **`DVD`** and **`Videotape`** as Unrelated Classes



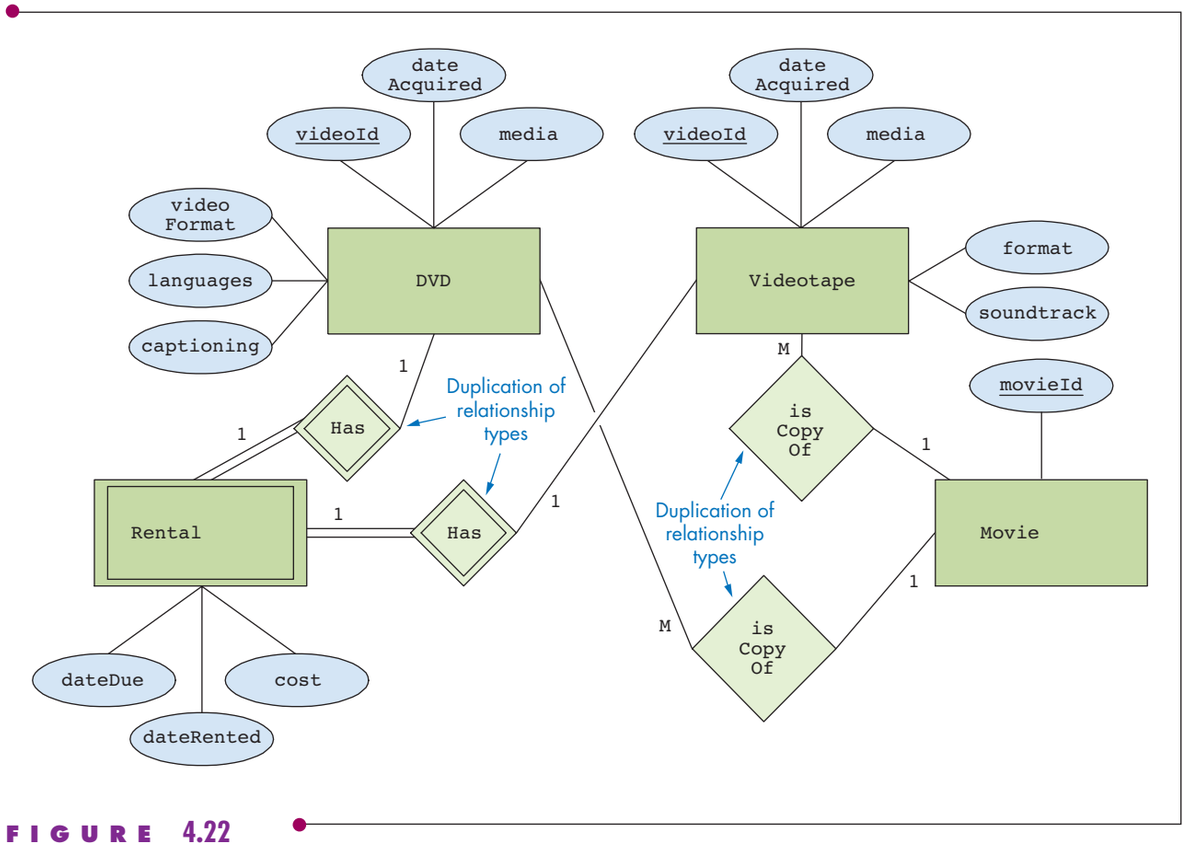


FIGURE 4.22
E-R Diagram with Relationship Types for Classes **DVD** and **Videotape**

cate every relationship type of entity class `Video`. Each movie has related videotapes and related DVDs. Adding these new classes has created a huge, complicated mess!

The object-oriented approach to representing this information recognizes that there is a commonality between videotape objects and DVD objects. Each one is a video object, has rentals and previous rentals, and has a related movie. Figure 4.23 shows the representation of the three related classes using the concept of **inheritance**, or **is-a**, **relationship types**. We say that a DVD entity *is a* video entity, and that entity class `DVD` *inherits* attributes from entity class `Video`. Similarly a videotape *is* a video and entity class `Videotape` *inherits* attributes from class `Video`. Inheritance relationship types in our style of E-R diagramming are depicted by the small circles (*inheritance circles*) and their associated lines and arcs. The cup symbols (*inheritance symbols*) point toward the superclass; in this case, `Video`.

Notice that the duplication of attributes and relationship types that we saw in Figure 4.22 has been transformed in Figure 4.23. A DVD has a `videoId`, a `dateAcquired`, and a `media` because of its relationship with a parent video. It also has relationship types with `Rental` and `Movie`. A DVD, as represented by this model, is a member of the entity classes `Video` and `DVD` and has all of the attributes and relationship types of both classes.

The major advantages of including object-oriented features in E-R diagramming are that the is-a relationship is complex and hard to describe without these features, and that the accurate maintenance of entities in classes such as `Video`, `DVD`, and `Videotape` require special treatment by database developers. These additional E-R diagramming symbols allow designers to accurately capture the nature of inheritance in a simple, graphical style. We'll see in Chapter 7 how the translation of inheritance from E-R diagrams to relational databases preserves the structure and the information content of these classes.

INTERVIEW

PETER CHEN

On the Birth of the Idea for the E-R model

Peter Chen published the first paper on the Entity-Relationship Model. That paper was selected as one of the top 40 great papers in the field of computing and remains one of the most cited papers in the computer software field. Today the E-R model is ranked as the top methodology used in database design by several Fortune 500 companies.

First Exposure to Computing I changed my major to CS my senior year at the National Taiwan University (NTU). I still feel very thankful to the Dr. Richard Chueh, visiting professor at NTU who introduced me to the computer field. After graduating from NTU, I got a fellowship from Harvard, where I studied CS under Dr. Ugo Gagliardi and Dr. Jeff Buzen.

The Earliest Career Steps I received my Ph.D. from Harvard and joined Honeywell to work on their next-generation computer project. One of my teammates was Mr. Charles Bachman, who is the inventor of the CODASYL/Network data model. Honeywell eventually abandoned our project and a few years later I joined MIT as an Assistant Professor of information systems in the Management School.

An Idea is Developed and Presented While at MIT, I received a lot of guidance from Dr. Stuart Madnick and met Dr. Mike Hammer, who was working on data models at that time. It was during my time at MIT that I published the first paper on the Entity-Relationship (E-R) model. A few years later, I moved to UCLA's management school. There, I learned a lot from several experts in the CS department including Dr. Wesley Chu, Dr. Alfonso Cardenas, and Dr. Leonard Kleinrock. Another colleague, Dr. Eph McLean, encouraged me to organize the first E-R conference. We expected a small gathering of 50 people or so. It turned out that over two hundred people showed up. Now, the E-R conference is an annual gathering of scientists and practitioners working in the conceptual modeling area and is held in a different country each year.

> **"It [the E-R model] solves a critical problem of almost any business: How to understand what data and information is needed to operate and manage the business."**

A Theoretical Concept's Every-Day Business Application Almost every business needs a methodology to organize the ever-increasing amount of data. The E-R model methodology has proven to work very well in practice in almost any kind of business all over the world. It solves a critical problem of almost any business: How to understand what data and information is needed to operate and manage the business. It helps address data issues such as: (1) Assuring that crucial data the business needs is collected, (2) disregarding unimportant information to avoid too much data, (3) organizing data in a retrievable manner, and (4) relating the data obtained. The E-R model helps a business identify and focus on what data it needs and, thus, solves issues (1) and (2). If a business uses the E-R model to guide the design of its databases, it will help to solve issues (3) and (4), particularly using the relationships of data to find the desired information.

The meaning of inheritance in E-R diagrams is both precise and complex. According to the E-R diagram of Figure 4.23, each entity of class `Video` may be a member of either class `DVD` or class `Videotape`. The inheritance diagram also specifies that each entity in class `DVD` and each entity in class `Videotape` must be an entity in class `Video`. Another way of saying this is that the set of all entities of class `Video` includes

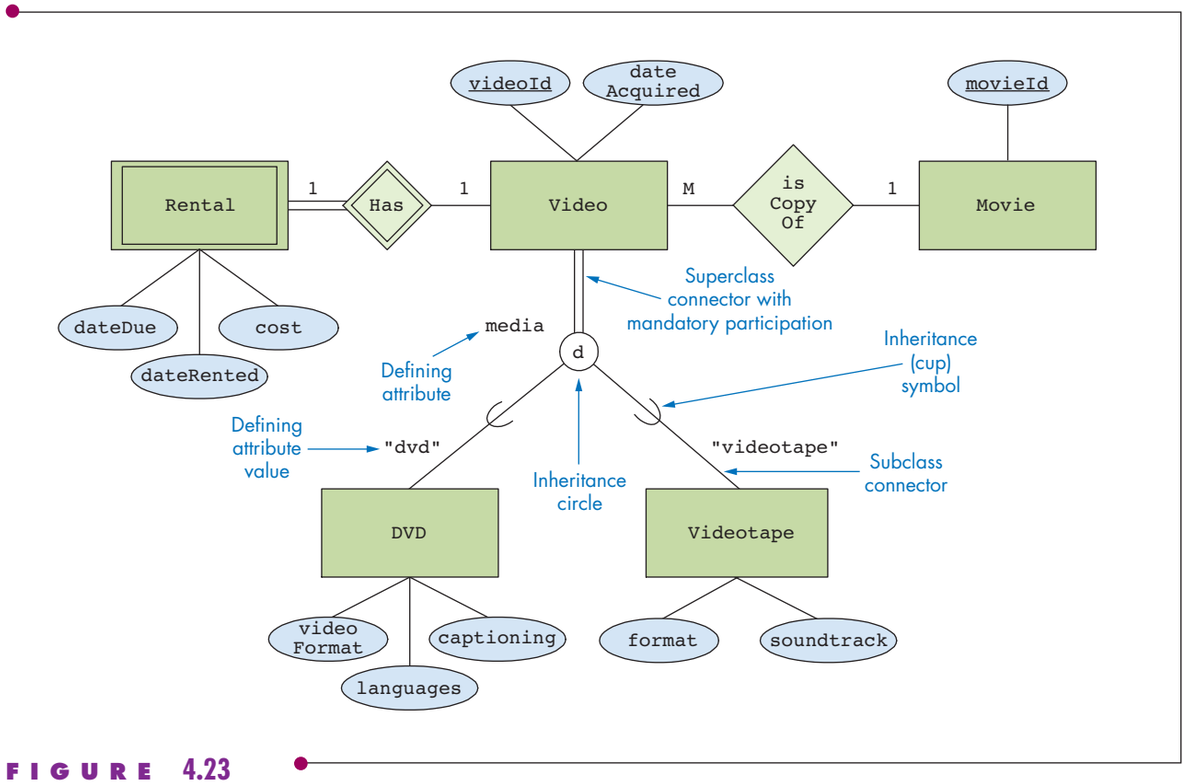


FIGURE 4.23
Object-Oriented E-R diagram for Entity Classes **Video**, **DVD**, and **Videotape**

all of the entities in classes DVD and Videotape. All of this meaning is defined simply by including the superclass connector, the inheritance circle, and the subclass connectors with their inheritance symbols.

Additional features of object-oriented modeling are included in Figure 4.23, as noted by the comments. These features allow us to represent more subtle characteristics of the inheritance relationship as graphical components of the E-R diagram.

Attribute media has been moved and is now placed near the line that connects Video to its inheritance circle, and its allowable values, "dvd" and "videotape," appear near the subclass connectors. This definition of the media attribute and its values combine to specify that the value of the media attribute of a video entity determines whether the video is in class DVD or class Videotape. Attribute media is called the *defining attribute* of the inheritance and the values "dvd" and "videotape" are the *defining attribute values.*

The superclass connector is a double line in Figure 4.23 to denote a mandatory participation constraint. That is, each video entity must participate in the inheritance by also being either a DVD or a videotape. This diagram does not allow the existence of a video entity that is neither DVD nor videotape. The small "d" in the inheritance circle specifies that the inheritance is *disjoint.* That is, each video entity is either a DVD or a videotape and cannot be both.

A different combination of participation and disjointedness is shown in Figure 4.24, which defines an example of inheritance for employees. As before, the inheritance diagram specifies that an employee may be a supervisor or a clerk and that each supervisor and each clerk must also be an employee. In this case, however, the superclass connector is a single line to express an optional participation constraint. This optional inheritance allows an employee to be neither a supervisor nor a clerk. The inheritance circle is empty to denote that this is *overlapping* (non-disjoint) inheritance. An employee is allowed to be both a supervisor and a clerk. The diagram also allows an employee to be a supervisor but not a clerk, or a clerk but not a supervisor.

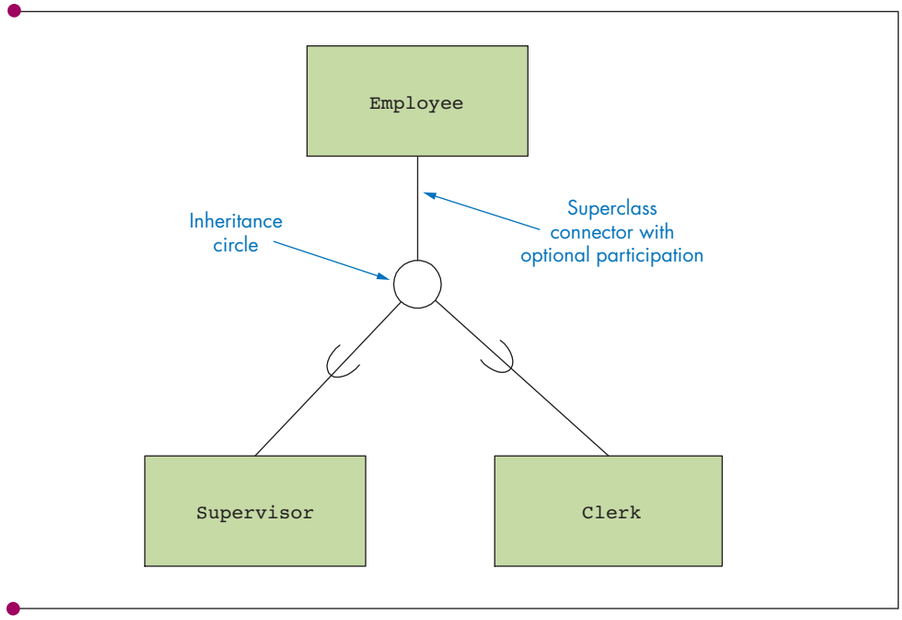


FIGURE 4.24
E-R Diagram Showing Optional and Overlapping Inheritance for Class **Employee**

Inheritance of attributes between classes is an important concept in E-R data modeling. You'll learn much more about inheritance in the following chapters.

4.7 CASE IN POINT: E-R MODEL FOR VIDEO SALES FOR BIGHIT VIDEO

The following statement is repeated and slightly modified from Chapter 3 and describes some aspects of the BigHit Video information system. This time, our job is to take the entity classes, attributes, and relationship types from Chapter 3 and create an E-R model for the system. The text highlighted in green has been added.

> BigHit Video Inc. wants to create an information system for online sales of movies in both DVD and videotape format. People will be allowed to register as customers of the online site and to update their stored information. Information must be maintained about customers' shipping addresses, e-mail addresses and credit cards. In a single sale, customers will be allowed to purchase any quantity of videos. The items in a single sale will be shipped to a single address and will have a single credit card charge.
>
> A customer will be provided with a virtual shopping cart to store items to be purchased. As each item is selected, it is added to the shopping cart. When the customer finishes shopping, he will be directed to a checkout area where he can purchase all of the items in the shopping cart. At this time, payment and shipping information is entered. Once the sale is complete, the shopping cart will be deleted and the customer will be sent a receipt by e-mail.

The analysis of this description in Chapter 3 produced a list of entity classes and attributes and a list of relationships that must be part of this information system. These lists are reproduced as Tables 4.1 and 4.2. The text highlighted in green tells us that there are two kinds of movies for sale. Table 4.1 has two new classes—`DVD` and `Videotape`—which are subclasses of `Movie`, as discussed in Section 4.4.

The representation of this information as an E-R diagram is shown in Figure 4.25. The diagram is a faithful representation of the entity classes, attributes, and relation-

Table 4.1 Entity Classes and Attributes for Online Movie Sales with Subclasses

Entity Class	Attribute	Constraints or Further Description
Customer	accountId	Key
	lastName	Not null
	firstName	
	shippingAddresses	Multivalued composite with components name, street, city, state, zipcode
	emailAddress	
	creditCards	Multivalued composite with components type, accountNumber, expiration
	password	Not null at least 6 characters
Movie	movieId	Key
	title	
	genre	
	media	Either "dvd" or "videotape" determines subclass
DVD	languages	Subclass of Movie
	videoFormat	
	captioning	
Videotape	format	Subclass of Movie
	soundtrack	
Sale	saleId	Key
	totalCost	
	dateSold	
	creditCard	Composite with components type, accountNumber, and expiration
ShoppingCart	cartId	Key
	dateCreated	

Table 4.2 Relationship Types for Online Movie Sales

Relationship Type	Entity Class	Entity Class	Cardinality Ratio	Attributes
Purchases	Customer	Sale	one-to-many	
Includes	Sale	Movie	many-to-many	quantity
Selects	Customer	ShoppingCart	one-to-many	
Includes	ShoppingCart	Movie	many-to-many	quantity

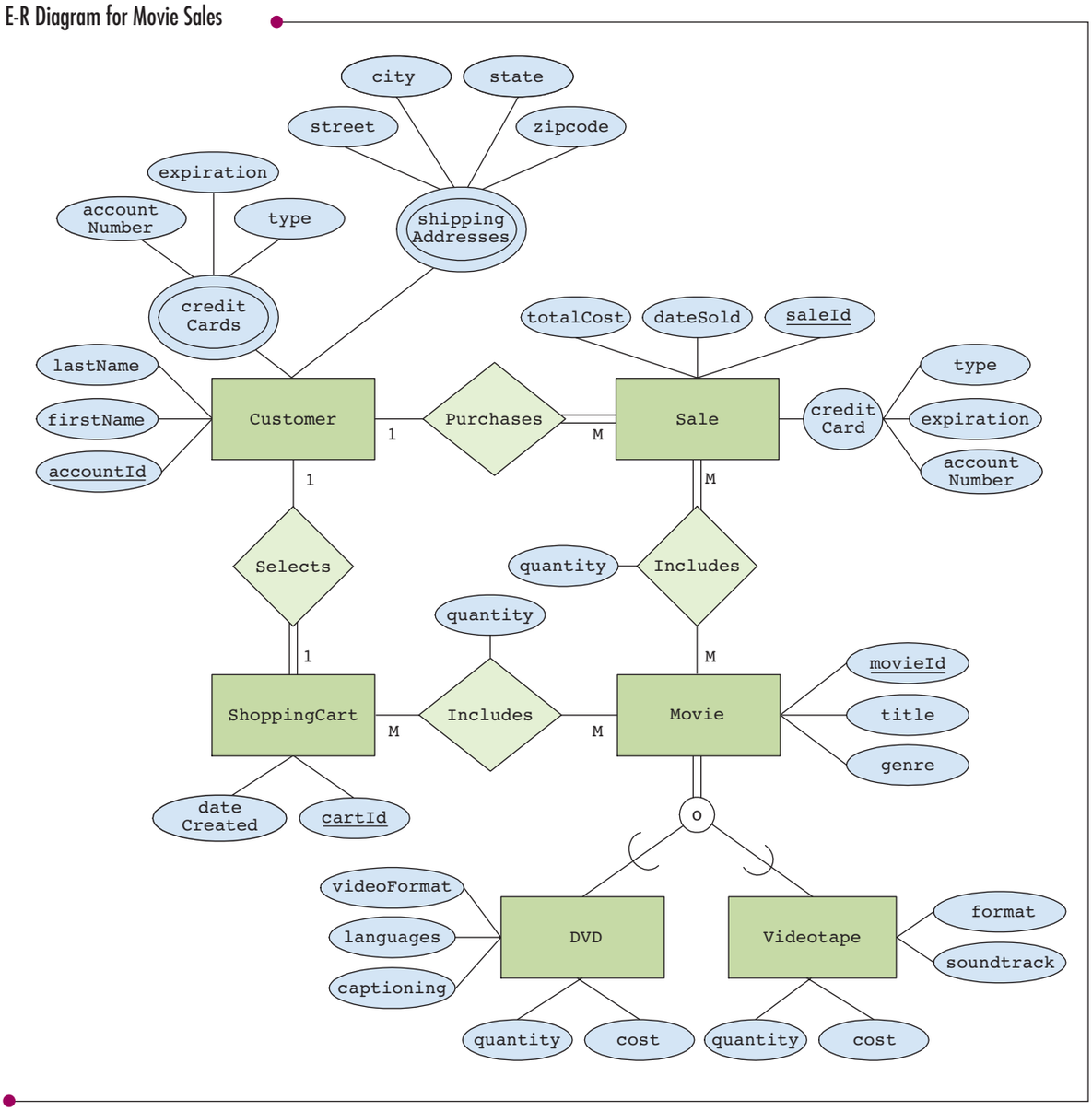
ship types. The only major improvement is in the participation constraints of the relationship types.

The E-R diagram specifies required participation by having double lines connect certain entity classes and relationship types. Each sale must be purchased by exactly one customer because of the double line (required participation) between `Sale` and `Purchases`. Similarly, each shopping cart must be selected by exactly one customer. A sale must include at least one movie because the line connecting `Sale` and `Includes` is double.

Single lines connecting an entity class and a relationship type denote optional participation. A customer need not have any sales nor any shopping carts. Also, because a shopping cart may be empty, `ShoppingCart` has optional participation with `Includes`.

The representation of the subclasses `DVD` and `Videotape` of entity class `Movie` is exactly that of Figure 4.25.

FIGURE 4.25
E-R Diagram for Movie Sales



CHAPTER SUMMARY

E-R diagrams are a concise and understandable representation of E-R data models. Diagrams are relatively easy for both developers and users to understand. They provide an excellent way for users and developers to talk about a planned database.

E-R diagrams represent entity classes by rectangles, attributes by ovals, and relationships by diamonds and connecting lines. Each of these symbols contains the natural name of the object being represented. Key attributes' names are underlined. Cardinality marks are placed alongside the relationship lines. Participation constraints are represented by double lines.

Users and developers can easily express and understand the relationship types represented in an E-R diagram by creating sentences through reading the diagram. They state or write the name of one entity class, the participation constraint of its role ("may" or "must"), the verb phrase that states the name of the relationship type, the cardinality constraint of its role ("one" or "many"), and finally the name of the other entity class.

An initial diagram representing the purchasing aspects of the BigHit Video system revealed errors in the data model so far. Designers must make a clear distinction in the data model between a movie and a video, which is an individual copy of some specific movie. `Title` and `rating` are attributes of the movie, but `dateAcquired` and `mediaType` are attributes of the video copy.

Closely related classes such as `Video`, `DVD`, and `Videotape` can be modeled with object-oriented extensions to E-R diagramming. An inheritance ("is a") relationship links the superclass (or parent class) `Video` with each of its subclasses `DVD` and `Videotape`. Each attribute and relationship type of entity superclass `Video` is thus inherited by entity classes `DVD` and `Videotape`. The entities in classes `DVD` and `Videotape` are also entities in class `Video`.

KEY TERMS

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

Discriminator or partial key. 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.

Entity-relationship (E-R) data model. A strategy for constructing conceptual data models using diagrams that focus on entity classes, relationship types, and attributes.

Entity-relationship (E-R) diagram. A graphical strategy for representing E-R models.

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

Inheritance (is-a) relationship type. A relationship type in which the subclass (child) is related to a superclass (parent). Each object of the subclass is also an object of the superclass and inherits all of the superclass attributes and relationships.

Object-oriented data model. A conceptual data model that divides objects into classes and supports the direct representation of inheritance relationship types. E-R diagramming supports the inclusion of object-oriented features.

Weak entity class. An entity class with no key of its own, whose objects cannot exist without being related to other objects. A weak entity class must have at least one identifying relationship type and corresponding owner entity class.

QUESTIONS

1. 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. Multivalued attribute and composite attribute

2. What is a data dictionary? Give three examples of information about entities and attributes that must be stored in a data dictionary and cannot be directly represented in an E-R diagram.

3. Discuss the relative merits of the E-R diagrams of Figures 4.5 and 4.6. Why would you choose one over the other?

4. Consider the E-R diagram for the BigHit Video enterprise as shown in Figure 4.12. Write relationship sentences, including participation and cardinality constraints to describe the roles of
   a. An employee in a store (2 roles).
   b. A pay statement to an employee.
   c. A store to an employee (2 roles).
   d. A time card to a store.

5. 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?

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

7. Consider the E-R diagram shown in Figure 4.19.

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.

8. Give a list of constraints that apply to "is-a" relationship types that don't apply to all relationship types. Consider participation constraints and cardinality constraints, among others.

PROBLEMS

9. Augment the E-R diagrams given in this chapter with attributes, as follows:

a. Augment Figure 4.1 by including attributes for information about customer preferences, credit cards, and forms of identification. You may use the customer application from Problem 3.16a as the model.

b. Augment Figure 4.2 by including attributes for information about videos, including length, rating, studio, and so on.

c. Augment Figure 4.3 by including attributes for information about stores, including location, name, and so on.

d. Augment Figure 4.12 by including attributes for employee information, as described in the employee application form of Problem 3.16d.

10. Consider the following description of an enterprise.

An auction Web site has items for sale that are provided by sellers. Each item has an opening price, a description, and an ending time. Customers submit bids. The highest, earliest bid submitted before the ending time is the winning bid and the item is sold to the bidder. Each seller must pay the auction company 5% of the winning bid. The auction company wants to be able to analyze the sales behavior of its customers and sellers and so must keep track of all bids and sales.

Draw an E-R diagram to represent this enterprise. Be sure to include all attributes and to identify the key attributes of each class.

PROJECTS

11. Continue the development of the BigHit Video system as shown in the E-R diagrams of this chapter. Extend the model of this chapter by adding additional entity classes, attributes, and relationship types as required. Produce a full E-R model for the system.

12. Continue the development of the Movie Lovers system started in the project of Chapter 2. Draw an E-R diagram to represent the entity classes, attributes, and relationship types of the Movie Lovers system. Include key designations and constraints on relationship types. Be sure to include the following activities:

a. The system allows people to become members by presenting them with a membership form. Include all appropriate attributes of members.

b. The system records information about people who are involved in making movies. Each person may take on the role of actor, director, or writer. Feel free to add additional roles. Consider whether these roles should be relationship types or entity classes. Don't forget that there may be many actors, writers, etc., for each movie.

c. Members should be able to record information about when and where they have seen movies, and make plans to see additional movies.

d. Members want to record their evaluations of movies both with a rating scheme and by writing reviews.

FURTHER READING

Discovery and modeling of information systems is described in Herbst [Her97] and Teorey [Teo94]. The original paper on E-R modeling is [Chen76]. Several books have extensive discussions of E-R modeling, including Elmasri and Navathe [ElNa99]; Date [Date99]; and Riccardi [01]. Modeling specific to electronic commerce is presented in Reynolds and Mofazali [ReMo02].