Conceptual Design: Entity Relationship Models

Craig Van Slyke, University of Central Florida
cvanslyke@bus.ucf.edu
John Day, Ohio University

Objectives

- Define terms related to entity relationship modeling, including entity, entity instance, attribute, relationship, cardinality, and foreign key.
- Describe the entity modeling process.
- Discuss how to draw an entity relationship diagram.
- Describe how to recognize entities, attributes, relationships, and cardinalities.
- Describe how to model supertype/subtype structures and unary relationships.

Overview

The ability of a database designer to understand and model the information that an organization uses is a critical design skill. Data modelers use a variety of tools and techniques to understand an organization's data. In order to understand how to properly model data, you must become familiar with a modeling approach known as **entity relationship modeling**, which is the subject to of this chapter.

Entity-relationship (E-R) modeling is one approach to **semantic modeling**. When database designers attempt to understand and represent meaning, they are engaged in semantic modeling, which can help in making database design more systematic (Date, 1990). Although a number of approaches to semantic modeling exist, this chapter focuses on entity relationship modeling. More specifically, we use E-R modeling to carry out conceptual data modeling

ER modeling, introduced by Chen (1976) consists of a number of activities that help database designers understand the objects the organization wants to store information about, the important characteristics of these objects and the associations among various objects.

The Entity Relationship Diagram

The end result of E-R modeling is the **E-R diagram (ERD)**, a graphical representation of the logical structure of a database. An ERD serves several purposes. First, the database analyst/designer gains a better understanding of the information to be contained in the database through the process of constructing the ERD. Second, the ERD serves as a documentation tool. Finally, the ERD is used to communicate the logical structure of the database to users. In particular, the ERD effectively communicates the logic of the database to users.

Entities

The E-R modeling process identifies three basic elements: entities, attributes and relationships.

An *entity* is a thing that can be distinctly identified (Chen, 1976). In database design, entities are the "things" about which the database stores information. Entities can include, but are not limited to

- tangible items, such as equipment,
- concepts, such as accounts,
- people, such as employees
- events, such as sales, or
- places, such as business locations.

The term **entity type** refers to a number of related items, while an **entity instance** refers to a single occurrence of an entity type. For example, employee number 123-45-6789 refers to a single occurrence, or entity instance, of the entity EMPLOYEE. The term entity refers to entity type. In addition, note that entity occurrence is sometimes used rather than entity instance. In ER diagrams, rectangles represent entities, as shown in Figure 1.

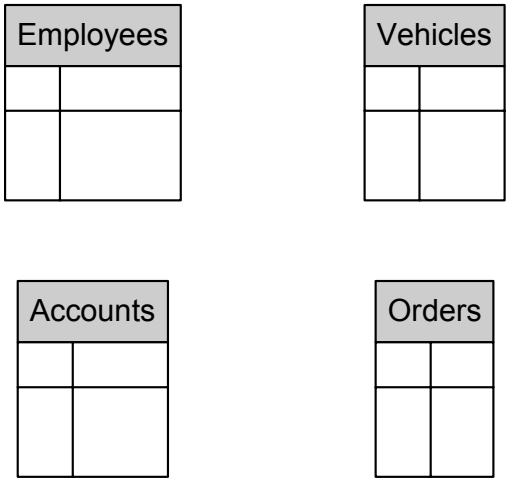

Figure 1: Entity Examples

Attributes

An **attribute** is a single data value that describes a characteristic of an entity. Other terms such as data item and field, describe the same essential concept.

Each entity has a corresponding set of attributes that represent the information about the entity that the organization is interested in. For example, a university may wish to know the name,

address, phone number, and major of each student. Put in database terms, STUDENT is the entity of interest, and NAME, ADDRESS, PHONE, and MAJOR are the attributes of interest.

Attributes are represented by ellipses in the ER diagram. The name of the attribute is shown inside the ellipse and a line is drawn from the attribute to its entity.

Primary Keys

Every entity *must* have a **primary key**. The primary key is an attribute or combination of attributes that uniquely identifies an instance of the entity. In other words, no two instances of an entity may have the same value for the primary key. Factors database designers must consider when choosing a primary key are discussed later in this chapter.

Sometimes it is useful to use more than one attribute to form a primary key. When a primary key for an entity is made up of more than one attribute the key is called a **composite key**. The terms composite key and compound key are also used to describe primary keys that contain multiple attributes.

When dealing with a composite primary key it is important to understand that it is the combination of values for all attributes that must be unique. It is not necessary for each attribute in the key to be unique. For example, the entity ENROLLMENT has a composite primary key comprised of the attributes STUDENT_ID and COURSE_ID. Each instance of ENROLLMENT must contain a unique combination of values for StudentID and CourseID. However, there can be duplications of StudentID or CourseID. So, it is possible for many instances of ENROLLMENT to have the value MIS100 for CourseID, but each of those instances must contain different values for StudentID.

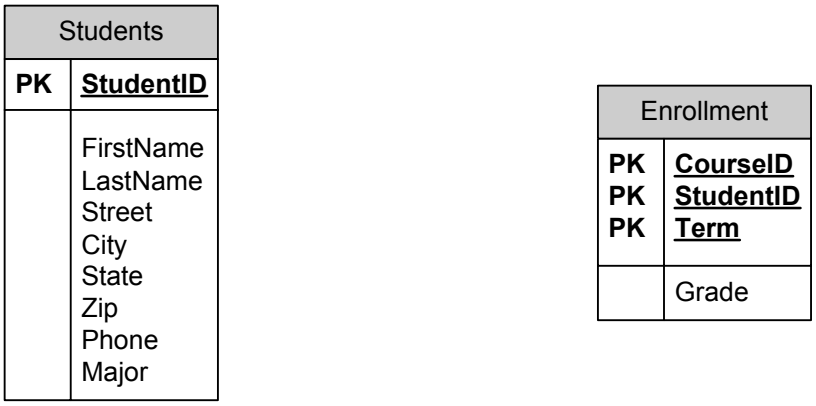

Figure 3: Entities with Attributes

Underlined attributes in Figure 3 indicate primary keys in the E-R diagram. Each attribute involved in making up a composite primary key is underlined. In Figure 3, the entity STUDENTS has nine attributes, including the primary key, which is StudentID. The entity ENROLLMENTS has four attributes, and a composite primary key made up of the attributes CourseID, StudentID and Term.

Relationships

A **relationship** is an association among two or more entities. For example, a STUDENTS entity might be related to a COURSES entity, or an EMPLOYEES entity might be related to an OFFICES entity. An ellipse connected by lines to the related entities indicates a relationship in an ERD, as shown in Figure 4. The line connecting the ORDERS and CUSTOMERS entities indicates that these two entities are related to one another.

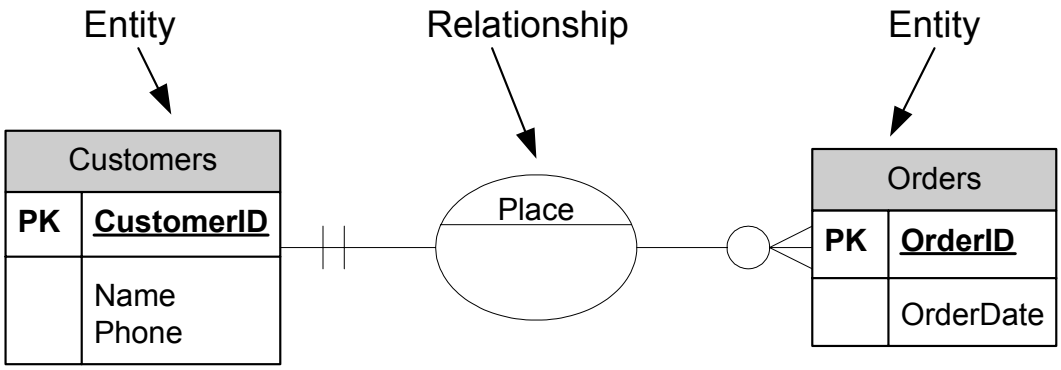

Figure 4: Entities and Relationship

Different relationship degrees exist. The degree of a relationship refers to the number of entities involved in the relationship. Although others exist, it is often sufficient to understand the meaning of three relationship degrees, unary, binary and ternary. A **unary relationship** (also called a recursive relationship) is a relationship involving a single entity. A relationship between two entities is called a **binary relationship**. When three entities are involved in a relationship, a **ternary relationship** exists. Relationships that involve more than three entities are referred to as n-ary relationships, where n is the number of entities involved in the relationship. Examples are provided for unary, binary and ternary relationship later in this chapter.

In some cases, attributes may be attached to a relationship, rather than an entity. For example, a GRADE attribute is a function of the combination of STUDENTS and COURSES, but is not strictly a function of either entity by itself. Attaching GRADE to STUDENT would not indicate that a STUDENT has a GRADE for a particular COURSE, while attaching GRADE to COURSES doesn't show that the GRADE is for a particular STUDENT. Attaching the GRADE attribute to the relationship between COURSES and STUDENTS shows that a value of GRADE is dependent on an intersection of COURSES and STUDENTS. A similar argument can be made for TERM. Figure 5 illustrates how to show this on an E-R Diagram.

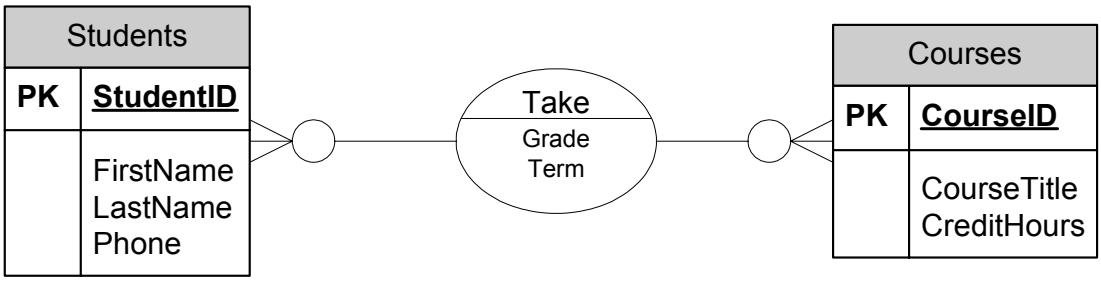

Figure 5: Relationship with Attributes

Cardinality

Relationships can also differ in terms of their cardinality. **Maximum cardinality** refers to the maximum number of instances of one entity that can be associated with a single instance of a related entity. **Minimum cardinality** refers to the minimum number of instances of one entity that *must* be associated with a single instance of a related entity. The following examples of binary relationships illustrate the concept of maximum cardinality (Fig. 6). Minimum cardinality is discussed in more detail later in the next section. If one CUSTOMER can be related to only one ACCOUNT and one ACCOUNT can be related to only a single CUSTOMER, the cardinality of the CUSTOMER-ACCOUNT relationship is **one-to-one** (1:1). [Note that each cardinality type is followed by a shorthand notation in parentheses.]

If an ADVISOR can be related to one or more STUDENTS, but a STUDENT can be related to only a single ADVISOR, the cardinality is **one-to-many** (1:N).

Finally, the cardinality of the relationship is **many-to-many** (M:N) if a single STUDENT can be related to zero or more COURSES and a single COURSE can be related to zero or more STUDENTS. Further examples are provided later in this chapter.

In E-R diagrams, cardinality is represented by symbols attached to the relationship line. A single vertical line intersecting the relationship line indicates a "one" cardinality. A crowfoot symbol indicates a "many" cardinality. Figure CARDINALITY shows ER diagrams with cardinality symbols. Figure 6(a) shows the E-R diagram for a 1:1 relationship, Figure 6(b) shows a 1:N relationship, and Figure 6(c) shows a M:N relationship. The symbols closest to the entities are the maximum cardinality symbols. We'll cover the symbols for minimum cardinalities later.

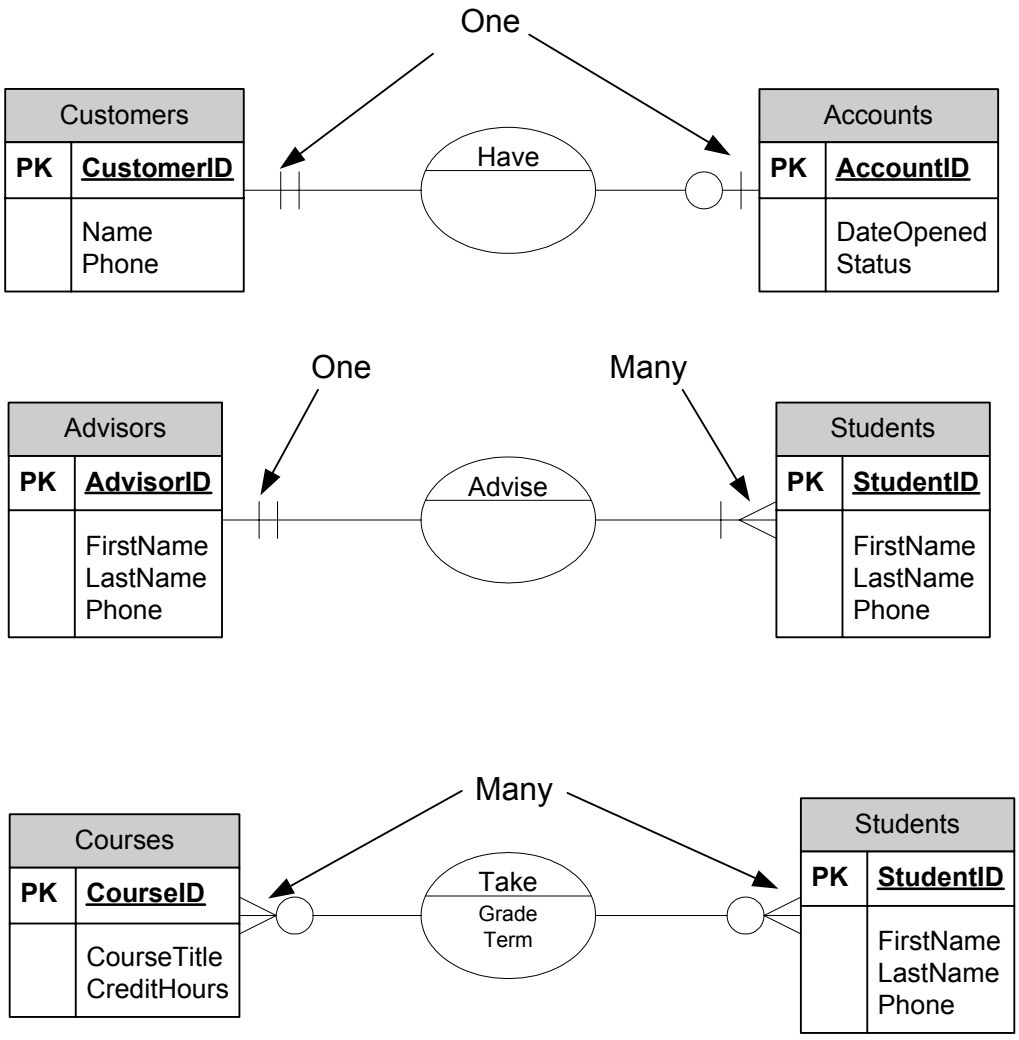

Figure 6: (a) 1:1 relationship (b) 1:N relationship (c) M:N relationship Cardinality Symbols

When determining the cardinality of relationships, it is important to remember that cardinality specifies how many instances an entity can be related to a *single* instance of a related entity. The trick to determining the cardinality of a relationship is to determine the cardinality of one side at a time.

For example, suppose you want to determine the cardinality of the STUDENTS to ADVISORS relationship. The first step is to determine how many STUDENTS *one* ADVISOR can be related to. One ADVISOR can be related to many STUDENTS. To represent this on an E-R diagram, show the "many" symbol (the crowfoot) next to the STUDENT entity. Next, you need to determine how many ADVISORS one STUDENT can be related to. While this may vary from school to school, in this case the answer is one. To show this on the E-R diagram, put the "one" symbol (vertical line) next to the ADVISOR entity. Figure 7 illustrates the process.

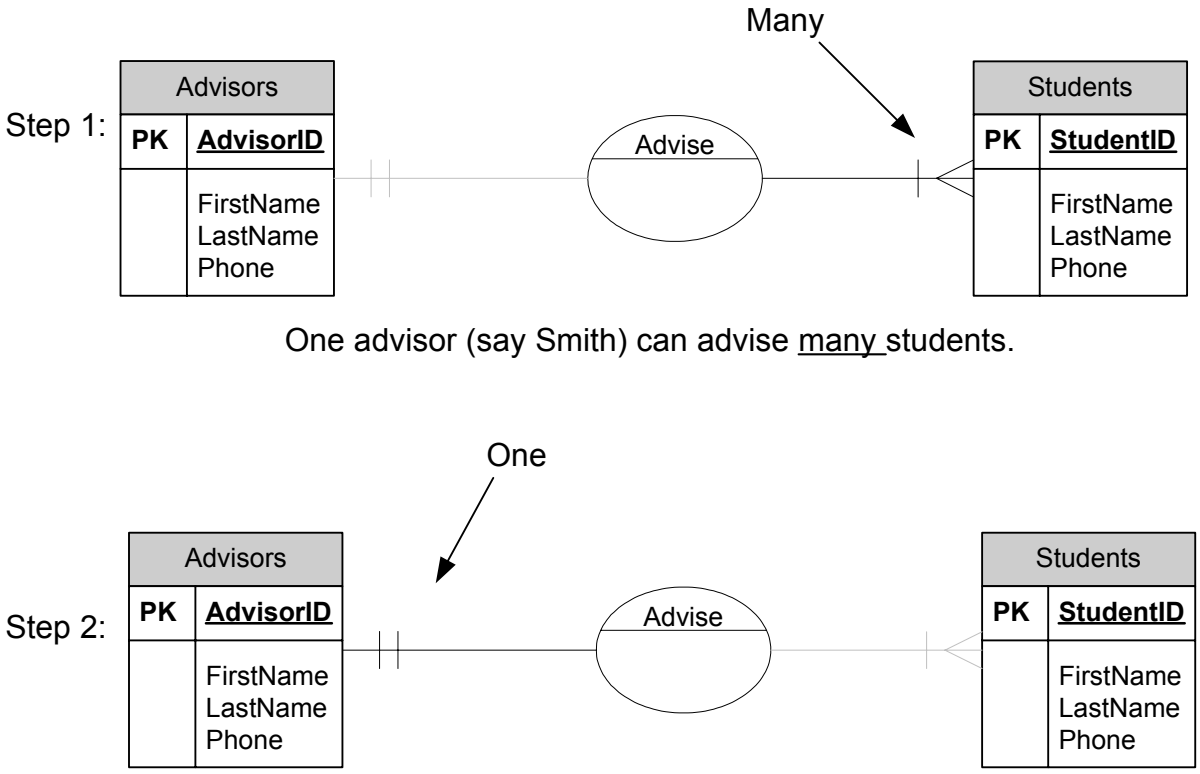

Figure 7: Determining Cardinalities

If you have trouble determining the cardinality of a relationship, the following method may help. Assign a name to the entity instance you want to hold to one. For example, if you were having trouble determining the cardinality of the ADVISOR to STUDENT relationship you could ask yourself, "How many ADVISORS can Jan Smith have?", or "How many STUDENTS can Dr. Smith advise?"

Creating an E-R Diagram

A number of steps are required to create an E-R diagram that accurately represents the organizational data. These steps are summarized in Figure 8 and discussed in this section.

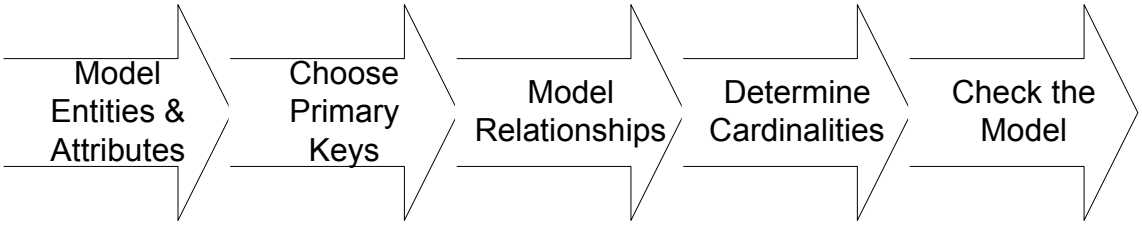

Figure 8: Steps in Building an E-R Diagram

E-R Modeling Example: An ORDER ENTRY FORM

To illustrate the steps in building an E-R diagram, we'll analyze an Order Entry Form. A sample of the Order Entry Form is shown in Figure 9.

ORDER-NO:	44-44-4444	**CUSTOMER-ID:**	1002	
DATE:	10/31/98	**CUST-NAME:**	ABC Inc.	
PROD-ID	**DESCRIPTION**	**PRICE**	**QTY**	**EXT**
A123	STEREO SYSTEM	375.00	2	750.00
C235	8" SPEAKER	150.00	8	1,200.00
X002	SPEAKER WIRE	10.00	5	50.00
		TOTAL		2,000.00

Figure 9: Order Entry Form

Step 1: Model the Entities

The first step in creating an E-R diagram is to model the entities. Recall that an entity is simply something about which the organization wishes to store data. A number of information sources may be helpful when identifying entities, including forms, data entry screens, reports and user interviews.

It is important to realize that an entity is basically defined by its attributes, so when identifying entities look for groups of related attributes. It is particularly helpful to look for possible primary keys for an entity. Generally, when a form has an identifier for a possible entity it is likely to be the entity. For example, if a form contains a space for a customer number, then the database (and the E-R diagram) probably needs to contain a CUSTOMER entity.

In user interviews, be particularly aware of nouns that the user mentions. Generally, entities are named with nouns such as CUSTOMER, STUDENT, and EQUIPMENT. Nouns that crop up often in the course of a user interview are good candidates for entities.

As you identify candidate entities, also try to determine attributes for the entities. An effective technique is to write down any attributes you identify next to the entity to which they belong. This may provide a means for you to distinguish between entities and attributes, which we discuss later in this section.

Another useful E-R modeling technique employed by database analysts is to highlight each item on a form, report, transcript or other information source to indicate that the item has been modeled. For example, when analyzing a form mark each item on the form as you write it on your list of possible entities and attributes. This can serve as a useful check. You can go back through each information source and make sure each important item is marked.

At this stage be liberal in identifying entities. If there is any possibility that some item on a form or some sentence in an interview identifies an entity, include it in the list of possible entity. It is more difficult to identify entities that are missed than it is to remove candidate entities that turn out not to be entities required in the database. Most of the potential entities that are later rejected turn out to be attributes of an entity rather than entities in their own right. These can be culled out later.

In the case of the Order Entry example, there are several candidate entities. The initial analysis of the Order Entry Form indicates that there are three entities that are clearly represented on the form: ORDER, CUSTOMER, and PRODUCT.

After identifying the entities, it may be necessary to make some assumptions. For example, when deciding which attributes to place with each entity, we must make an assumption about PRICE. If the price of a product does not change from one order to another, then PRICE is a function of PRODUCT. If, however, different orders for the same product have different prices, then PRICE is a function of the relationship between PRODUCT and ORDER. In actual practice, the analyst must consult with the user representative to determine which view is correct. We will make the assumption that PRICE varies for all orders—PRICE is an attribute of PRODUCT. Note that the attribute QTY (quantity) is purposely absent from the list in order to illustrate the process of checking the ERD.

Table 5.1 shows the initial list of entities and attributes identified in the analysis of the Order Entry form.

Entity	**Attributes**
ORDER	ORDER-NO, DATE, TOTAL
PRODUCT	PROD-ID, DESCRIPTION, PRICE
CUSTOMER	CUSTOMER-ID, CUST-NAME

Table 5.1: Entity List for the Order Entry Form

Once the list of candidate entities and their attributes is complete, the database analyst must determine which entities need to be in the database and which should be excluded. These decisions are based on whether a candidate entity is an entity or an attribute. Many of the candidate entities clearly belong in the database the most likely candidate entities that should be rejected are those for which no attributes have been identified. If you are unable to find any attributes for an entity, that entity is probably in reality an attribute for another entity. For example, CUSTOMER_NAME might be identified as a possible entity. However, no attributes can be identified for CUSTOMER_NAME. This leads to the conclusion that CUSTOMER_NAME is not an entity, but is better considered an attribute of the CUSTOMER entity.

Not all cases are so clear, however. Take the example of ADDRESS. Perhaps a number of attributes were identified for this candidate entity, such as NUMBER, STREET, CITY, STATE, and POSTAL_CODE. Does the presence of these potential attributes of ADDRESS indicate that ADDRESS is an entity? Although in some situations this will be the case, more often all of these should be attributes of some other entity, such as CUSTOMER. Resolving these types of situations becomes easier with more experience in data modeling and greater knowledge of the organization.

Step 2: Choose Primary Keys

After identifying and modeling each entity and its attributes, primary keys must be chosen for each entity. For many entities, the primary key is obvious or already in place in the organization's existing information systems. For example, a university may already be using a student identification number. In such cases, the best course of action is to retain the existing key.

When a primary key does not exist and is not obvious, candidate keys must be identified. The major requirement for an attribute to be a primary key is that the attribute uniquely identifies instances of the entity. In other words, for each instance of the entity the value of the attribute must be unique. In addition, the proper functioning of the database requires that primary keys can never be null. The primary key for each entity must always have a unique, valid value for each instance of the entity. Combining these two requirements results in each instance of an entity always having a one and only one primary key (even if it's a composite primary key), and each primary key is unique.

Other characteristics of primary keys are also desirable. In general, a good primary key is what is sometimes called "data-less." This means that no actual information is contained in the primary key. An account number, for example, typically contains no useful information and serves no purpose other than to identify an account. Primary keys that are data-less also posses another desirable characteristic—they never change. Primary keys whose values change over time lead to a number of problems with maintaining the database.

An example illustrates the problem with using a primary key that does not meet the characteristics in Table 5.2.

Desirable Primary Key Characteristics
1. Uniquely identifies an entity instance.
2. Non-null (always has a value)
3. Data-less
4. Never changes

Table 5.2: Characteristics of a Good Primary Key

Suppose that a database designer decided to make the primary key of the entity CUSTOMER the customer's phone number. Assume for the moment that each customer has a phone number and that no two customers can have the same phone number. In other words, assume that the attribute PHONE is unique and non-null. While this is enough for PHONE to serve as the primary key of CUSTOMER, problems may occur because PHONE is not data-less, and is also subject to change. What happens if a customer changes his or her phone number? Should the database be updated to reflect a new value for a primary key? This causes a number of database maintenance problems. These maintenance problems stem from using the primary key to link records in different tables. If the primary key value is changed for a record in the CUSTOMER table, related records in other tables must also be changed. Then, it is better to simply leave the old phone number in, but this introduces an inaccuracy in the database.

When no suitable primary key can be found among the existing attributes for an entity, it is acceptable, and in fact usually a good idea, to simply create a new attribute. These newly created primary keys are counters that increment with each new instance of an entity in the same manner in which a check number increments from one check to the next.

For the Order Entry example, we need to identify primary keys for three entities: ORDER, PRODUCT, and CUSTOMER. Fortunately, all of these entities have attributes that make acceptable primary keys. For ORDER, ORDER-NO is a good choice for the primary key, for PRODUCT, PROD-ID makes an acceptable key, and CUSTOMER-ID is a good key for CUSTOMER. Notice that all of these satisfy our requirements for a primary key. They are unique for each occurrence of the entity. They will never be null. They are data-less, and they are not subject to change.

Once all of the entities and their attributes and primary keys are identified, the actual drawing of the E-R diagram can begin. Simply draw a rectangle for each entity, labeling the rectangle with the entity name. Then add an ellipse for each attribute, making sure each is labeled and connected to the proper entity. Finally, indicate primary keys by underlining each primary key attribute. Figure 10 shows the results for the Order Entry example.

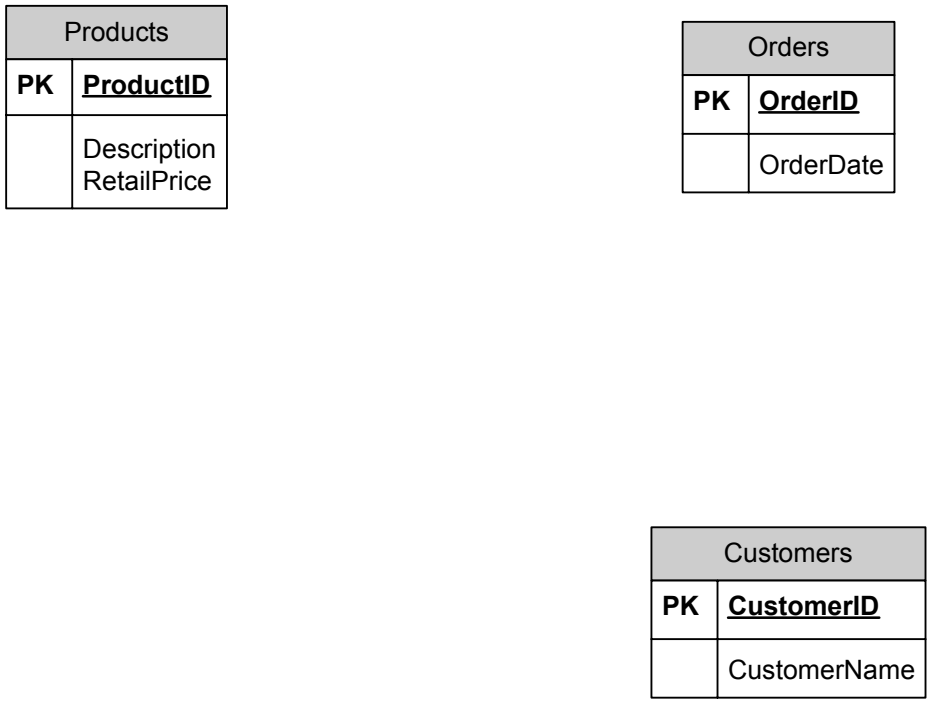

Figure 10: Order Entry Form Entities and Attributes

When all of the entities, attributes and primary keys are modeled, it is time to identify and model the relationships between entities.

Step 3: Model Relationships

Relationships among entities are a critical part of the Entity Relationship Diagram. When these relationships are implemented in the database, they provide the links among the various tables that give the database its flexibility. To maximize the flexibility of a database, relationships must be properly identified and modeled.

Many relationships are relatively easy to recognize, such as those between ORDERS and CUSTOMERS, or between STUDENTS and COURSES. Others, however, are less clear. Although becoming truly proficient at recognizing relationships requires experience and practice, there are some general guidelines that can help the database analyst recognize relationships.

When examining forms, reports and entry screens, be on the lookout for entities whose attributes appear on the same form, report or screen. Analysis of the Order Entry Form (Figure 9) reveals three entities: CUSTOMER, ORDER, and PRODUCT. We can conclude that these entities may be related to one another. But how are they related? In cases where the form being analyzed only represents two entities, the relationship is fairly obvious—the relationship is between the two entities. However, with the Order Entry Form, there are three entities represented. Further consideration is required to determine how these entities are related. Knowing something about the organization is particularly helpful in such situations.

In the case of the Order Entry Form example, ORDER and CUSTOMER are related, as are ORDER and PRODUCT. However, we know that there may not necessarily be a relationship between PRODUCT and CUSTOMER. Because both are related to ORDER we can report which products are ordered by a particular customer.

In some cases, the analyst works from user interviews, rather than forms. In these cases, the analyst can examine transcripts of user interviews to determine the relationships among entities. Often users mention related entities in the same sentence. Consider the excerpt from a user interview shown in Figure 11. Notice how the user talks about products and orders in the same sentence. Once CUSTOMERS and ORDERS are identified as entities, mentioning both in the same comment is a good indication that the entities are related. This interview also tells us that we need to store some additional information about products. It also lets us know that there are two prices, a selling price and a retail price. We'll deal with these later.

ANALYST:	What information do you need to know about orders?
USER:	Well, for each **order**, we need to know the order identifier and date as well as the **customer** placing the order. We also need to know what products are included in each order along with the quantity and selling price for each product on the order.
ANALYST:	What information do you need to know about products, beyond the information needed for an order?
USER:	Of course, we need a product ID and description. Oh yea, we also need to store a standard selling price, you know, a retail price.

Figure 11: Excerpt from a User Interview

As you identify relationships, note them on the E-R diagram as discussed earlier. Assign some meaningful name to the relationship and add the name to the relationship diamond. When having difficulty in coming up with a relationship name, many analysts simply combine the names of the entities on either side of the relationship. For example, a relationship between STUDENT and MAJOR could be called STUDENT-MAJOR. The diagram for the Order Entry Form is shown in Figure 12. The lines connecting the entities are relationships are shaded because we'll complete them by adding cardinalities in the next step.

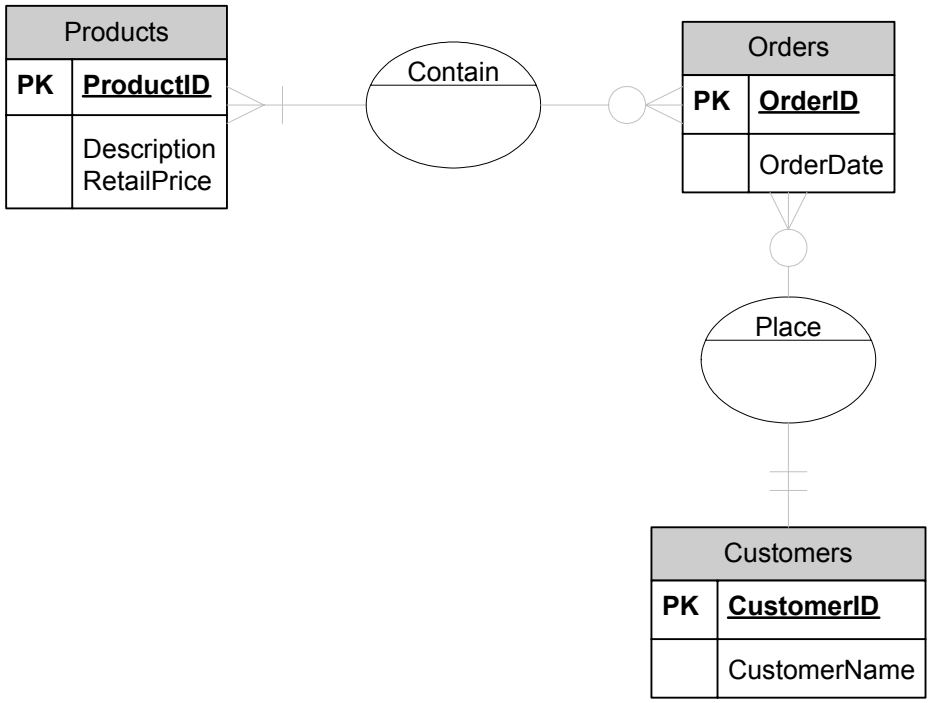

Figure 12: Relationships for Order Entry Database

After all of the relationships are modeled in the E-R diagram, the cardinalities of the relationship must be determined.

Step 4: Determine Cardinalities

Recall that there are both maximum and minimum cardinalities. The maximum cardinality of a relationship is the number of instances of one entity in a relationship that can be related to a single instance of another entity in the relationship. In contrast, the minimum cardinality is the number of instances of one entity that *must* be related to a single instance of the related entity. Many novice database analysts find determining the cardinalities of relationships more confusing than identifying entities, attributes and relationships. Thankfully there are ways to make this task easier.

It is common to determine maximum cardinalities before minimum cardinalities. There are two parts to the maximum cardinality of a binary relationship, one for each entity. Recall the example of the ADVISOR-STUDENT relationship discussed earlier in this chapter. One ADVISOR can be related to many STUDENTS. This indicates that the STUDENT side of the relationship has a many cardinality. This is only half of the relationship's cardinality, however. To complete the cardinality, we must recognize that one STUDENT can be related to only a single ADVISOR. This tells us that the ADVISOR side of the relationship has a cardinality of 1. Thus, the cardinality of the ADVISOR-STUDENT relationship is one-to-many.

One technique that often helps analysts determine the proper cardinality of a relationship is to give the instance of the single instance side of the relationship a name. For example, consider trying to determine the cardinality of the STUDENT-COURSE relationship. First, hold the

STUDENT entity to a single instance to determine the cardinality of the COURSE side of the relationship. This is easier to do if, rather than saying "How many COURSES can a STUDENT be related to?" say "How many COURSES can Mary Wilson be related to?" The answer, of course, is many, so the COURSE side of the relationship has a many cardinality. To determine the cardinality of the STUDENT side of the relationship, ask "How many students can be related to MIS380?" Once again, the answer is many, indicating that the STUDENT side of the relationship also has a many cardinality. When asking these cardinality questions, remember that you are always determining what cardinality symbol to draw next to the entity you are *not* holding to one instance.

Let's return to the Order Entry Form example. There are two relationships, one between PRODUCT and ORDER, and the other between ORDER and CUSTOMER. One instance of ORDER can be related to many instances of PRODUCT. In other words, one ORDER can contain many PRODUCTS. Thus, the cardinality from ORDER to PRODUCT is many. Turning to the other side of the relationship, we can see that one instance of PRODUCT can be related to many instances of ORDER—a single PRODUCT can be on many ORDERS. So, the cardinality from PRODUCT to ORDER is many. Combining the two sides gives us a many-to-many cardinality between ORDER and PRODUCT.

Now we must analyze the relationship between CUSTOMER and ORDER. A single ORDER can be related to only one CUSTOMER. In other words, a single ORDER can't be placed by more than one CUSTOMER. This means that the cardinality from ORDER to CUSTOMER is one. On the other side, a single CUSTOMER can place many ORDERS, so the cardinality from CUSTOMER to ORDER is many, and we can say that the cardinality of the CUSTOMER-ORDER relationship is one-to-many.

Now the minimum cardinalities must be determined. Both maximum and minimum cardinalities are determined by **business rules**. However, cardinalities are sometimes less obvious than others when the analyst does not have good knowledge of the organization. For example, it is relatively easy to determine that an ORDER must be related to at least one CUSTOMER, indicating that the minimum cardinality from ORDER to CUSTOMER is one. The same can be said for ORDER and PRODUCT. But, must a CUSTOMER be related to at least one ORDER? This is less clear. Maybe the organization allows customers to set up accounts prior to placing their first order. In this case, the minimum cardinality from CUSTOMER to ORDER is zero. We will make this assumption for the Order Entry Form example. The situation is similar for the minimum cardinality from PRODUCT to ORDER. It is reasonable to assume that we have products that have not been ordered yet. If this is allowed, then the minimum cardinality from PRODUCT to ORDER is zero.

Assumptions should only be temporary in actual practice. The analyst may have to make assumptions in order to proceed with the analysis, but it is critical that the validity of the assumptions be checked with the users before completing the analysis [and definitely before implementation!].

Figure 13 shows the Order Entry ER Diagram with maximum and minimum cardinalities indicated. It is now time for the last step in the E-R Diagramming process, checking the model.

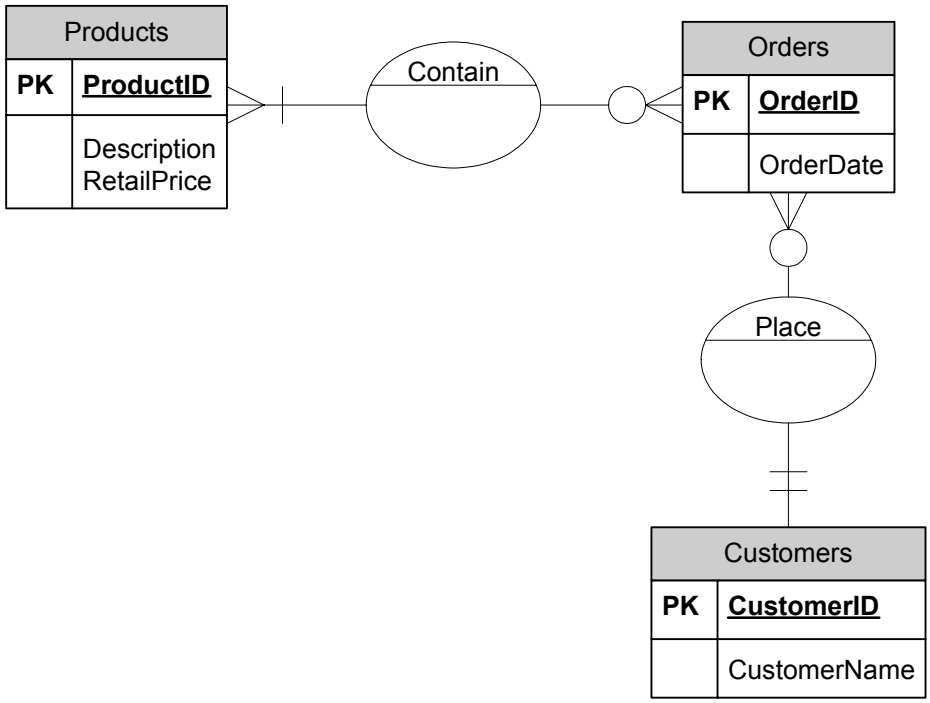

Figure 13: Incomplete Order Entry ERD with Cardinalities

Step 5 - Check the Model

The final step in creating an E-R diagram is often overlooked, but is just as important as any of the previous steps. Analysts who fail to carefully check their ERD often produce diagrams of poor quality, which of course should be avoided.

In order to check the ERD, you must return to your original information sources, the forms, reports, and interviews with users. The basic idea is to go back to the original documents and make sure that the structure represented in the ERD can satisfy the requirements. For example, the representations in the ERD must be able to reproduce any forms or reports required. We will use the Order Entry Form, which for convenience is reproduced below, as an example.

ORDER ENTRY FORM

ORDER-NO:	44-44-4444	**CUSTOMER-ID:**	1002	
DATE:	10/31/98	**CUST-NAME:**	ABC Inc.	
PROD-ID	**DESCRIPTION**	**PRICE**	**QTY**	**EXT**
A123	STEREO SYSTEM	375.00	2	750.00
C235	8" SPEAKER	150.00	8	1,200.00
X002	SPEAKER WIRE	10.00	5	50.00
		TOTAL		2,000.00

Figure 9: Order Entry Form (repeated)

The first step in using the Order Entry Form to check the ERD is to make sure that all of the information contained in the form is also represented on the ERD. The easiest way to do this is to take a copy of the form and check off each item as you verify that the item is on the ERD.

Examination of the Order Entry Form shows that there are three items that are not represented on the ERD: EXT, TOTAL, and QTY. Both EXT, which is short for Extended, and TOTAL can be computed, so it is not necessary to store these in the database, or represent them on the ERD. In most cases, it is not necessary to store attributes that can be computed. There are times, however, when it may make sense to store an item that can be computed. For example, TOTAL can be computed, but doing so required retrieving data from multiple records. It may be that storing this value improves performance enough to justify storing TOTAL.

Unlike EXT and TOTAL, QTY (Quantity) can not be computed, and therefore must be stored in the database. Once the analyst decides that QTY should be represented on the ERD, the question becomes how to represent it. Initial ideas might include representing QTY as an attribute of ORDER, or of PRODUCT. However, QTY is not really an attribute of either of these, but is properly represented as an attribute of the *relationship* between ORDER and PRODUCT. It is not unusual to have attributes attached to relationships with a many-to-many cardinality.

There is an additional, more subtle, decision that must be made. This regards the price shown in the order form. We must decide if price is a function of the product alone (as identified by ProductID), or if the price depends on both the product and the order. To make this decision, we must ask the following question. Is it possible for the same product to have different prices on different orders? If the answer is no, then price depends on the product alone and the price attribute should be included in the PRODUCTS entity. If the answer is yes, then product is a function of both product and order, and should be placed on the relationship between PRODUCTS and ORDERS. Typically, the latter is the case, so we'll place the price attribute on the relationship. Note that we are calling this attribute "SellingPrice" to indicate that is the price for which a product was sold a particular order. If we wanted to store a standard price, we would probably call it "RetailPrice" or something similar. Using more specific attribute names leads to less confusion when reading the ERD or resulting database.

Adding Quantity to the ERD results in the complete ERD, which is shown in Figure 14.

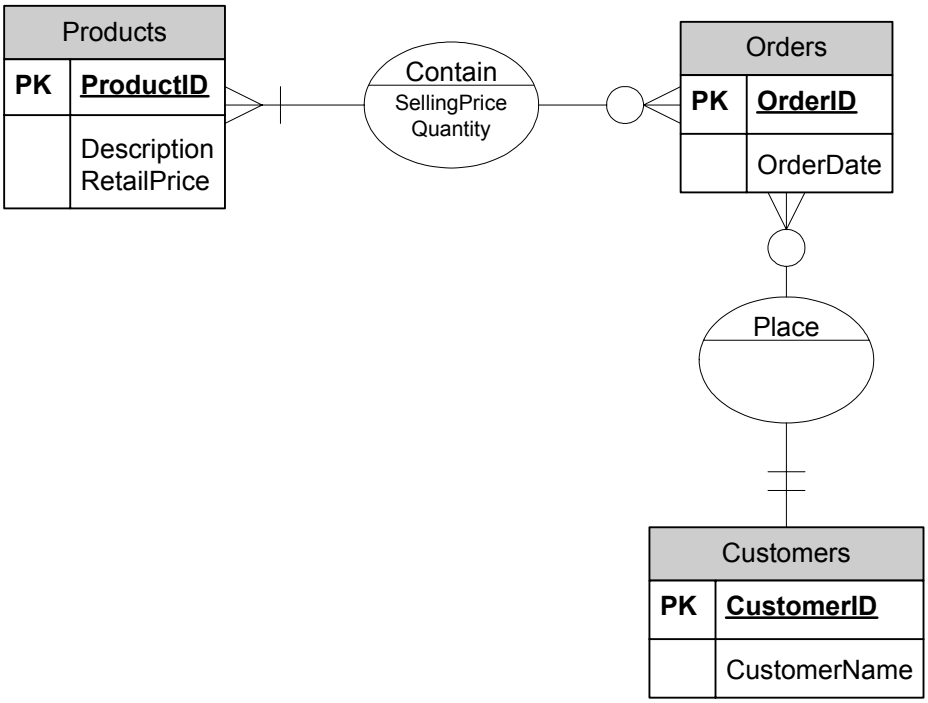

Figure 15: Complete Order Entry ERD

Special Topics in E-R Diagramming

This section discusses some special situations that you may encounter in creating E-R diagrams. These include resolving many-to-many relationships, supertype/subtype structures, and unary relationships.

Resolving Many-to-Many Relationships

Some database analysts find it useful to resolve many-to-many relationships (M:N) in the ERD. This makes the process of converting the ERD to a set of normalized relations easier.

In order to resolve M:N relationships, you must create an **associative entity** (which is sometimes called a gerund), which is essentially an entity. It is called an associative entity because it is an entity that associates two instances of other entities with one another. An associative entity is shown in the ERD in the same way any other entity is shown. We use the many-to-many relationship between PRODUCTS and ORDERS as an example to illustrate how to create an associative entity.

Associative entities are relatively easy to create. Start by creating a new entity (the associative entity) and using it to replace the many-to-many relationship. Next, we need to decide what attributes to give the associative entity. Any attributes that were part of the converted

relationship should also be in the associative entity. In our example, the attributes "Quantity" and "SellingPrice" are attributes of the relationship, so we'll include these in the associative entity.

Next, we need to decide on a primary key for the associative entity. A standard practice is to use the primary keys from the two original entities (PRODUCTS and ORDERS). In our example, these primary keys are ProductID and OrderID. We may be able to use a combination of these two entities as the primary key for our associative entity. Recall that a primary key that is made up of more than one attribute is called a composite primary key (or composite key for short). Before deciding to use ProductID + OrderID for our primary key, we must make sure that this combination is unique. In other words, no combination of the same ProductID and OrderID may be repeated in instances of the associative entity. Another way to think of this is to ask yourself if a product can ever appear twice in the same order. The answer is "no." So, we can use ProductID + OrderID as our primary key. If the answer happened to be "yes" we would need to either add one or more additional attributes to make the composite key unique, or we could simply create a new attribute to use as a primary key.

The next step in creating the associative entity is to determine the cardinalities. Maximum cardinalities are always the same when converting the M:N relationship. The maximum cardinalities coming into both sides of the associative entity are both many, while the maximum cardinalities going into the original entities is always a one.

Minimum cardinalities are a bit trickier, however. The minimum cardinalities for the original entities are always one. Each instance of the associative entity must be related to exactly one instance of each of the original entities. The minimum cardinalities going into the associative entity depend on the minimum cardinalities going into the original entities. Let's look at an example.

In the ERD shown in Figure 15, the minimum cardinality going into ORDER is a zero, while the minimum cardinality going into PRODUCT is a one. This means that an instance of ORDER must be related to at least one PRODUCT, but an instance of PRODUCT does not have to be related to an ORDER. It is helpful to think of the associative entity as being an intersection of PRODUCT and ORDER. Since it is possible for an instance of PRODUCT to not be related to any instances of ORDER, it is also possible for a PRODUCT to not be related to any intersections of PRODUCT and ORDER. In other words, the minimum cardinality on the PRODUCT side of the associative entity is zero. Following the same logic, since each instance of ORDER must be related to at least one instance of PRODUCT, the minimum cardinality of the ORDER side of the associative entity is one. Figure 16 shows an annotated example of converting a many-to-many relationship to an associative entity.

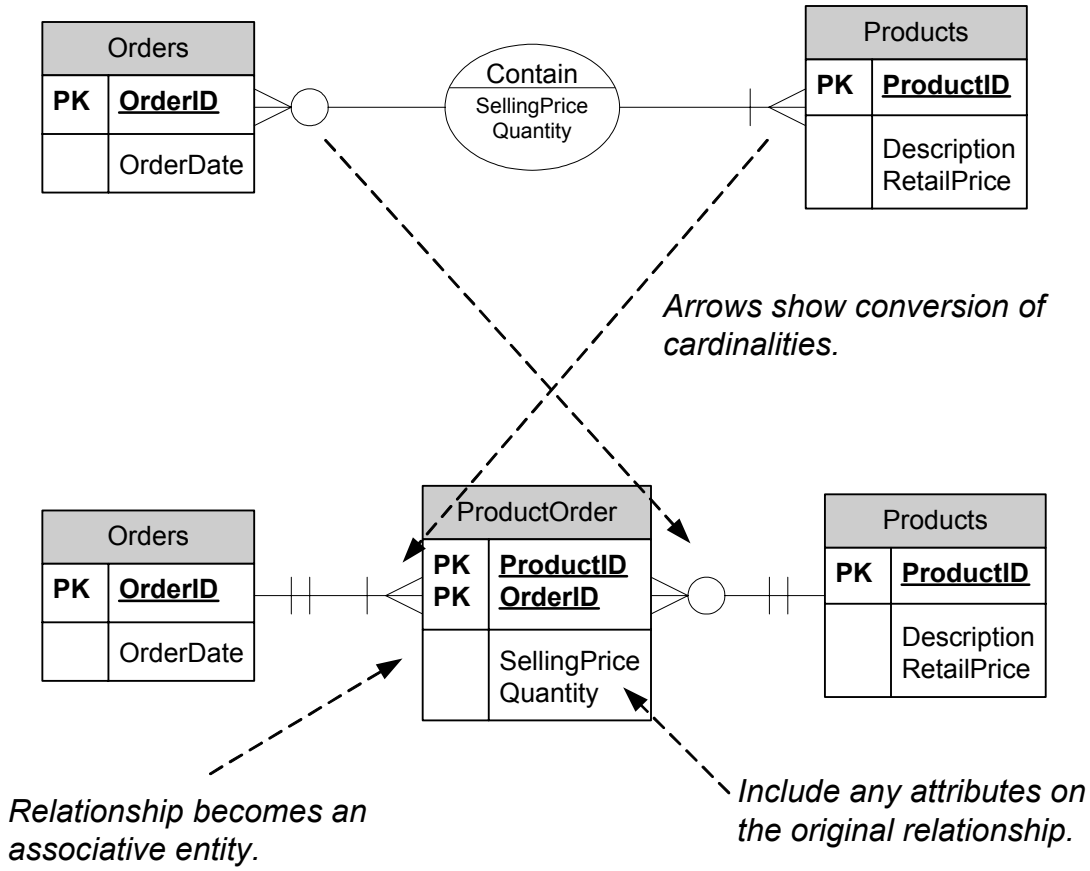

Figure 16: Converting a Many-to-Many Relationship

While converting many-to-many relationships in an ERD is optional, in practice most database designers do this conversion automatically. This is done because many-to-many relationships must be converted before the model can be implemented as a database. Converting them in the ERD saves time and effort in the next phase (logical design) of the database design and implementation process.

Supertypes and Subtypes

Certain instances of an entity class can include attributes that are not needed in other instances of the same entity class. In these cases, it is useful to use a supertype/subtype structure. This structure is also called a **generalization/specialization hierarchy**. An example of the portion of a database that deals with customer information will illustrate a generalization/specialization hierarchy. Note that the number of attributes in our example is artificially limited for illustration purposes.

For each CUSTOMER, we need to store an ID, and the customer's name. However, for retail customers we also need to track the sales tax rate. For wholesale customers there is no need to store the sales tax rate, but two additional attributes, discount and credit limit, need to be stored. Notice how each type of reference needs to have some common and some special information. This necessitates the use of a supertype/subtype structure.

One possibility for handling this situation is to simply include all attributes in a single entity. While this would work, it requires excessive storage space. If the database contains 1000 customers, 500 retail and 500 wholesale, each of the 1000 of the records in the database would have one or two empty fields. Even though these fields are empty, they still take up space. The supertype/subtype structure avoids this.

Figure 17 illustrates how the supertype/subtype structure is shown in an ERD. Note that it is not necessary to show a primary key for the subtypes. They automatically include the same primary key as the supertype. In addition, this is a case of a **disjoint** subtype, which is indicated by the "d" in the circle underneath CUSTOMERS. This means that each supertype can be only one subtype--it is not possible for one instance of CUSTOMERS to be both a RETAIL and a WHOLESALE customer. Sometimes subtypes follow the **overlap rule**, which means that an instance of the supertype may be an instance of more than one subtype. When the overlap rule is in effect, an "o" is placed in the circle.

An additional characteristic of this example is that the subtypes are complete—there are no other possible subtypes of the CUSTOMER supertype. This is known as the **completeness constraint**, and is either a complete specialization (each supertype instance **must** be an instance of at least one subtype), or a partial specialization (a supertype instance may or may not be an instance of a subtype). The double line between CUSTOMERS indicates a complete specialization. A single line would indicate a partial specialization.

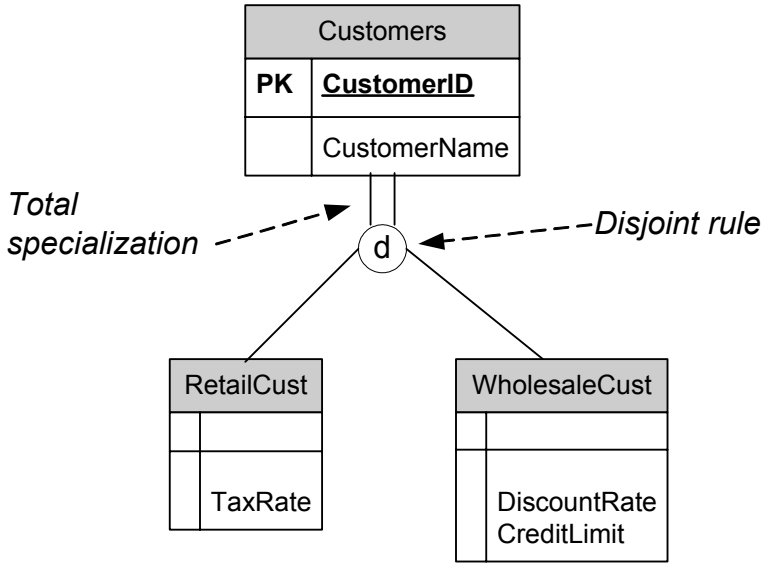

Figure 17: Supertype/Subtype Hierarchy

Now let's look at an example of a partial specialization with the overlap rule. Before doing so, however, it is important that you understand that the disjoint/overlap rule is a completely separate issue from the total/partial specialization. You can have any combination of these,

- disjoint with partial specialization,
- disjoint with total specialization
- overlap with partial specialization, or
- overlap with total specialization.

Figure 18 shows a supertype/subtype hierarchy that is a partial specialization with the overlap rule in effect. In this example, the supertype is STUDENTS, which has the subtypes ATHLETES, and MUSICIANS. A student can be both an athlete and musician (overlap rule) or may be neither (partial specialization).

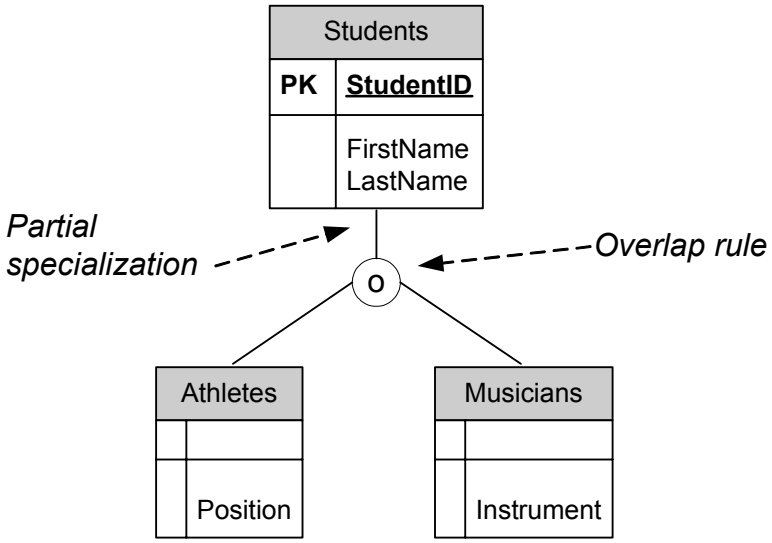

Figure 18: Supertype/Subtype Example with Overlap and Partial Specialization

One way to recognize the need for a supertype/subtype hierarchy is when some instances of an entity have special attributes that other instances do not. This was the case in the two examples we have seen so far. Another reason to have a supertype/subtype hierarchy is when some instances of an entity participate in relationships that others do not. An example of such a situation is shown in Figure 19. There are two types of properties in the database, single family homes and condominiums. For single-family homes, we need to store the lot size, but we don't need to store this for condos. However, we need to know information about the property manager who manages each condo. We need a subtype for condos because of the special relationship to property manager. We only need to know about this relationship for condos, not for single family homes. Note that sometimes a subtype has both special attributes and special relationships.

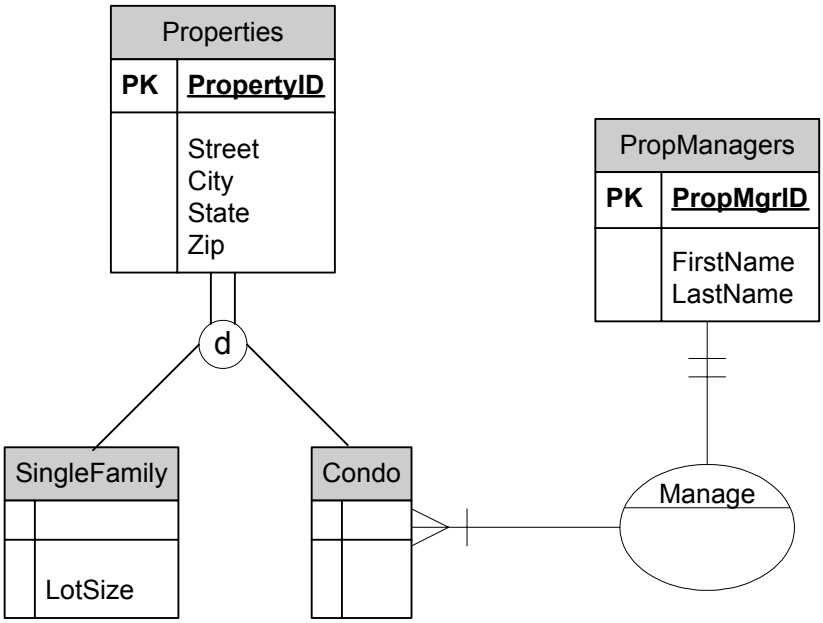

Figure 19: Supertype/Subtype Example Based on a Special Relationship

A key concept related to supertypes and subtypes is inheritance. Each subtype inherits all of the attributes of the supertype. In other words, each subtype automatically includes all of the attributes of the supertype. This alleviates the need to repeat these shared attributes in each subtype. Note also that there are no cardinality symbols shown for the supertype/supertypes. This is because the cardinality is understood to be a mandatory one (minimum and maximum are both one) on the supertype side and an optional one (minimum zero and maximum one) on the subtype side.

Unary Relationships

Sometimes, one instance of an entity is related to another instance of the same entity. For example, one employee may manage another employee, or one course may have other courses as prerequisites. Relationships such as these are known as **unary** relationships. They are sometimes called **recursive** relationships.

Representing unary relationships in an ERD is done by drawing an ellipse to represent the relationship, and then connecting it to the entity twice. The examples from the previous paragraph are shown in Figure 20.

Note that maximum and minimum cardinality symbols are shown for the unary relationships. From Figure 20, we can tell that a given employee may manage many other employees, or may not manage any. We can also tell that each employee must be managed by exactly one employee. From the diagram of the PREREQ relationship we can tell that a given course can have many prerequisites or may have none, and that a course can be a prerequisite for many courses or may

not be a prerequisite for any other courses. The type attribute is used to store the nature of the prerequisite (whether it's a prerequisite or a co-requisite). You do not have to have any attributes in a many-to-many unary relationship.

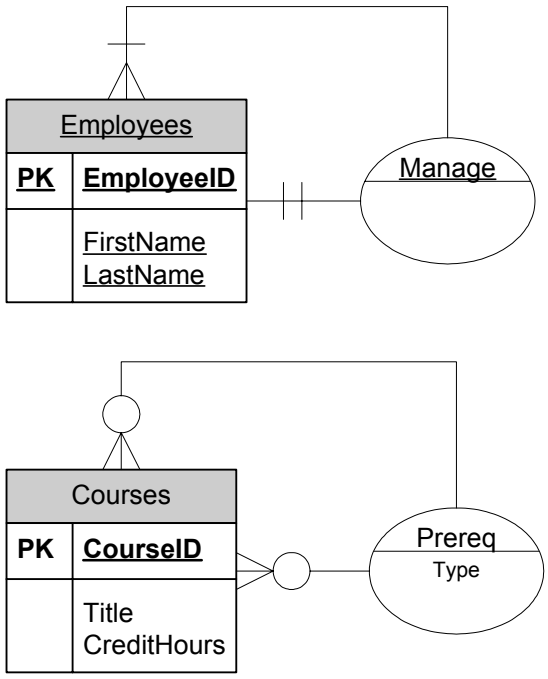

Figure 20 – Unary (recursive) Relationships

Although there are a number of situations not covered by the previous discussion, most situations can be modeled through the use of the examples provided. As you gain more practice in conceptual data modeling, you may realize that even complex models with many entities can be constructed using the skills covered in this chapter.

Summary

Creating an entity relationship modeling is an important step in the process of creating a database. The output of this modeling effort is an entity relationship diagram. The process of creating an ERD helps the database analyst understand the information needs to end users. In addition, the ERD provides a means of conveying this understanding to end users.

This chapter discussed and provided an example of the process of creating an ER diagram, including how to identify entities, attributes, primary keys, and relationship. The symbols used in drawing an ERD were also described.