The Entity Relationship Model

At a basic level, databases store information about distinct objects, or *entities*, and the associations, or *relationships*, between these entities. For example, a university database might store information about students, courses, and enrollment. A student and a course are entities, while an enrollment is a relationship between a student and a course. Similarly, an inventory and sales database might store information about products, customers, and sales. A product and a customer are entities, while a sale is a relationship between a customer and a product.

A popular approach to conceptual design uses the *Entity Relationship* (ER) model, which helps transform the requirements into a formal description of the entities and relationships that appear in the database. We'll start by looking at how the Entity Relationship modeling process itself works, then apply it in Entity Relationship Modeling Examples" for three sample databases.

Representing Entities

To help visualize the design, the Entity Relationship Modeling approach involves drawing an Entity Relationship (ER) diagram. In the ER diagram, a rectangle containing the entity name. For our sales database exa e sets would be shown as in Figure 4-1.

Figure 4-1. An entity set is represented by a named rectangle

We typically use the database to store certain characteristics, or *attributes*, of the entities. In a sales database, we could store the name, email address, postal address, and telephone number for each customer. In a more elaborate customer relationship managment (CRM) application, we could also store the names of the customer's spouse and children, the languages the customer speaks, the customer's history of interaction with our company, and so on. Attributes describe the entity they belong to.

An attribute may be formed from smaller parts; for example, a postal address is composed of a street number, city, ZIP code, and country. We classify attributes as *composite* if they're composed of smaller parts in this way, and as *simple* otherwise.

Some attributes can have multiple values for a given entity. For example, a customer could provide several telephone numbers, so the telephone number attribute is *multivalued*.

Attributes help distinguish one entity from other entities of the same type. We could use the name attribute to distinguish between customers, but this could be an inadequate solution because several customers could have identical names. To be able to tell them apart, we need an attribute (or a minimal combination of attributes) guaranteed to be unique to each individual customer. The identifying attribute or attributes form a *key*.

In our example, we can assume that no two customers have the same email address, so the email address can be the key. However, we need to think carefully about the implications of our choices. For example, if we decide to identify customers by their email address, it would be hard to allow a customer to have multiple email addresses. Any applications we build to use this database might treat each email address as a separate person, and it might be hard to adapt everything to allow people to have multiple email addresses. Using the email address as the key also means that every customer must have an email address; otherwise, we wouldn't be able to distinguish between customers who don't have one.

have the same name, so we can use the combination of the telephone number and the name as a composite key.

Clearly, there may be several possible keys that could be used to identify an entity; we choose one of the alternative, or *candidate*, keys to be our main, or *primary*, key. You usually make this choice based on how confident you are that the attribute will be non-empty and unique for each individual entity, and on how small the key is (shorter keys are faster to maintain and use).

In the ER diagram, attributes are represented as labeled ovals and are connected to their owning entity, as shown in Figure 4-2. Attributes comprising the primary key are shown underlined. The parts of any composite attributes are drawn connected to the oval of the composite attribute, and multivalued attributes are shown as double-lined ovals.

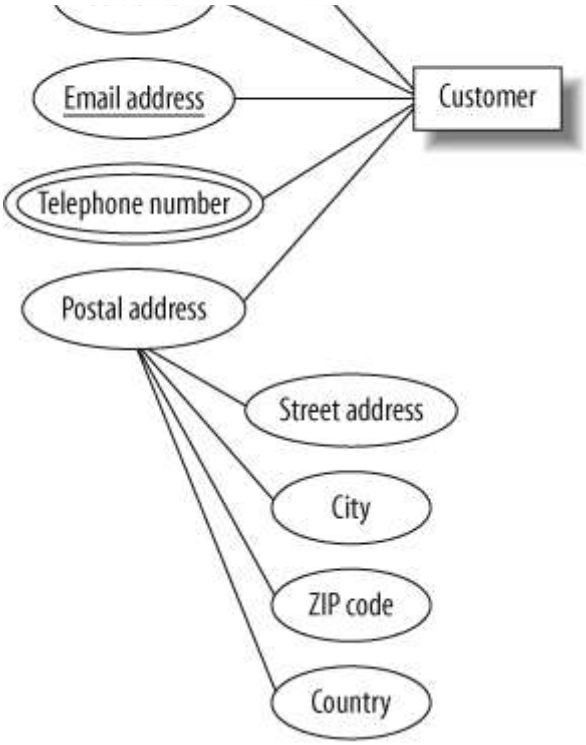

Figure 4-2. The ER diagram representation of the customer entity

Attribute values are chosen from a *domain* of legal values; for example, we could specify that a customer's given names and surname attributes can each be a string of up to 100 characters, while a telephone number can be a string of up to 40 characters. Similarly, a product price could be a positive rational number.

Attributes can be empty; for example, some customers may not provide their telephone numbers. The primary key of an entity (including the components of a multiattribute primary key) must never be unknown (technically, it must be `NOT NULL`); for example, if it's possible for a customer to not provide an email address, we cannot use the email address as the key.

You should think carefully when classifying an attribute as multivalued: are all the values equivalent, or do they in fact represent different things? For example, when listing multiple telephone numbers for a customer, would they be more usefully labeled separately as the customer's business phone number, home phone number, cell phone number, and so on?

Let's look at another example. The sales database requirements may specify that a product has a name and a price. We can see that the product is an entity because it's a distinct object.

For some applications, no combination of attributes can uniquely identify an entity (or it would be too unwieldy to use a large composite key), so we create an artificial attribute that's defined to be unique and can therefore be used as a key: student numbers, Social Security numbers, driver's license numbers, and library card numbers are examples of unique attributes created for various applications. In our inventory and sales application, it's possible that we could stock different products with the same name and price. For example, we could sell two models of "Four-port USB 2.0 Hub," both at $4.95 each. To distinguish between products, we can assign a unique product ID number to each item we stock; this would be the primary key. Each product entity would have name, price, and product ID attributes. This is shown in the ER diagram in Figure 4-3.

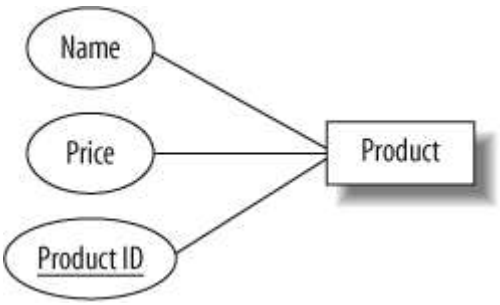

Figure 4-3. The ER diagram representation of the product entity

Representing Relationships

Entities can participate in relationships with other entities. For example, a customer can buy a product, a student can take a course, an artist can record an album, and so on.

Like entities, relationships can have attributes: we can define a sale to be a relationship between a customer entity (identified by the unique email address) and a given number of the product entity (identified by the unique product ID) that exists at a particular date and time (the timestamp).

Channel Sub-Woofer Speakers."

Different numbers of entities can appear on each side of a relationship. For example, each customer can buy any number of products, and each product can be bought by any number of customers. This is known as a *many-to-many* relationship. We can also have *one-to-many* relationships. For example, one person can have several credit cards, but each credit card belongs to just one person. Looking at it the other way, a *one-to-many* relationship becomes a *many-to-one* relationship; for example, many credit cards belong to a single person. Finally, the serial number on a car engine is an example of a *one-to-one* relationship; each engine has just one serial number, and each serial number belongs to just one engine. We often use the shorthand terms 1:1, 1:N, and M:N for one-to-one, one-to-many, and many-to-many relationships, respectively.

The number of entities on either side of a relationship (the *cardinality* of the relationship) define the *key constraints* of the relationship. It's important to think about the cardinality of relationships carefully. There are many relationships that may at first seem to be one-to-one, but turn out to be more complex. For example, people sometimes change their names; in some applications, such as police databases, this is of particular interest, and so it may be necessary to model a many-to-many relationship between a person entity and a name entity. Redesigning a database can be time-consuming if you assume a relationship is simpler than it really is.

In an ER diagram, we represent a *relationship set* with a named diamond. The cardinality of the relationship is often indicated alongside the relationship diamond; this is the style we use in this book. (Another common style is to have an arrowhead on the line connecting the entity on the "1" side to the relationship diamond.) Figure 4-4 shows the relationship between the customer and product entities, along with the number and timestamp attributes of the sale relationship.

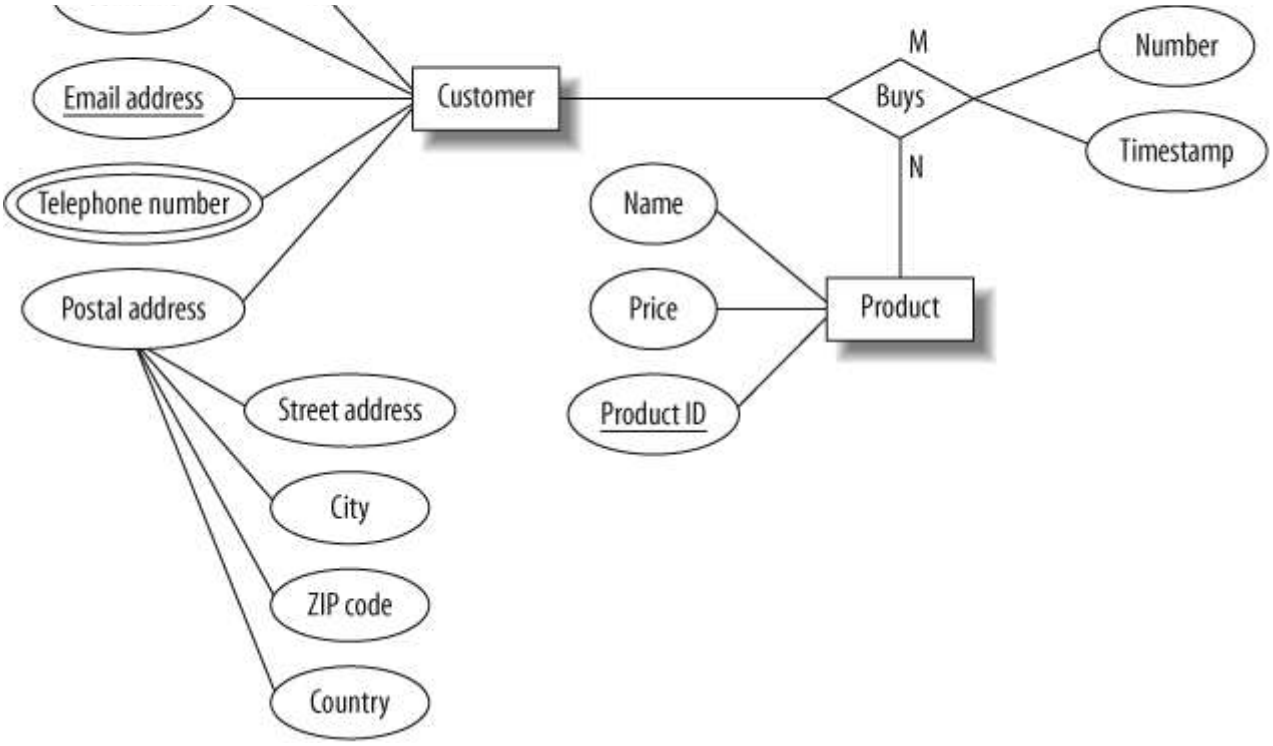

Figure 4-4. The ER diagram representation of the customer and product entities, and the sale relationship between them.

Partial and Total Participation

Relationships between entities can be optional or compulsory. In our example, we could decide that a person is considered to be a customer only if they have bought a product. On the other hand, we could say that a customer is a person whom we know about and whom we hope might buy something—that is, we can have people listed as customers in our database who never buy a product. In the first case, the `customer` entity has *total participation* in the bought relationship (all `customer`s have bought a product, and we can't have a `customer` who hasn't bought a product), while in the second case it has *partial participation* (a `customer` can buy a product). These are referred to as the *participation constraints* of the relationship. In an ER diagram, we indicate total participation with a double line between the entity box and the relationship diamond.

bute or an entity on its own. For example, an email address could be modeled as an entity in its own right. When in doubt, consider these rules of thumb:

Is the item of direct interest to the database?

Objects of direct interest should be entities, and information that describes them should be stored in attributes. Our inventory and sales database is really interested in customers, and not their email addresses, so the email address would be best modeled as an attribute of the `customer` entity.

Does the item have components of its own?

If so, we must find a way of representing these components; a separate entity might be the best solution. In the student grades example at the start of the chapter, we stored the course name, year, and semester for each course that a student takes. It would be more compact to treat the course as a separate entity and to create a class ID number to identify each time a course is offered to students (the "offering").

Can the object have multiple instances?

If so, we must find a way to store data on each instance. The cleanest way to do this is to represent the object as a separate entity. In our sales example, we must ask whether customers are allowed to have more than one email address; if they are, we should model the email address as a separate entity.

Is the object often nonexistent or unknown?

If so, it is effectively an attribute of only some of the entities, and it would be better to model it as a separate entity rather than as an attribute that is often empty. Consider a simple example: to store student grades for different courses, we could have an attribute for the student's grade in every possible course; this is shown in Figure 4-5. Because most students will have grades for only a few

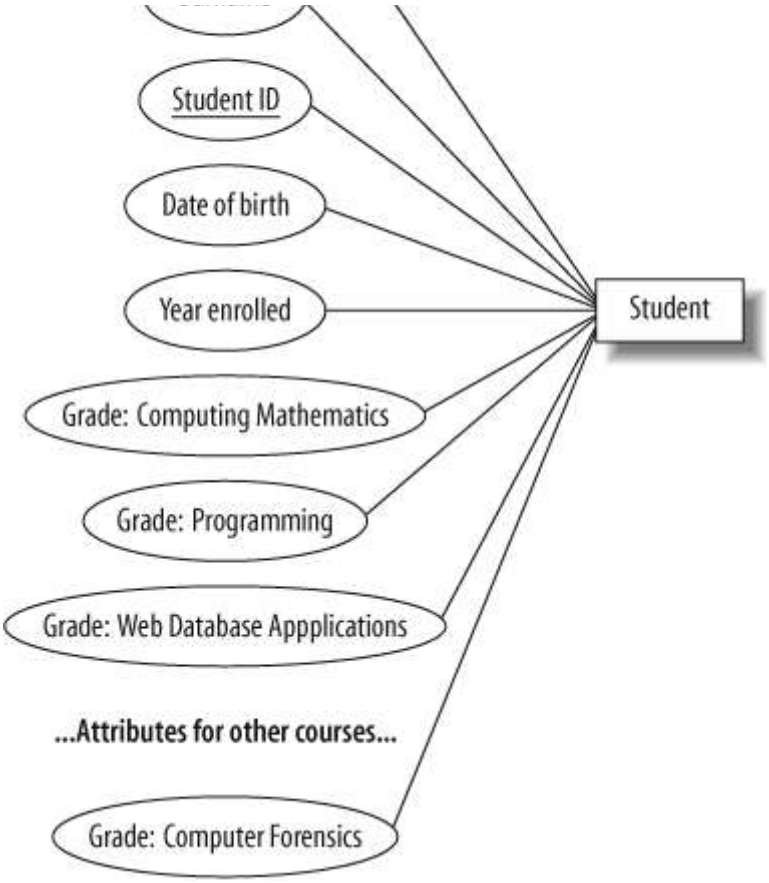

Figure 4-5. The ER diagram representation of student grades as attributes of the student entity

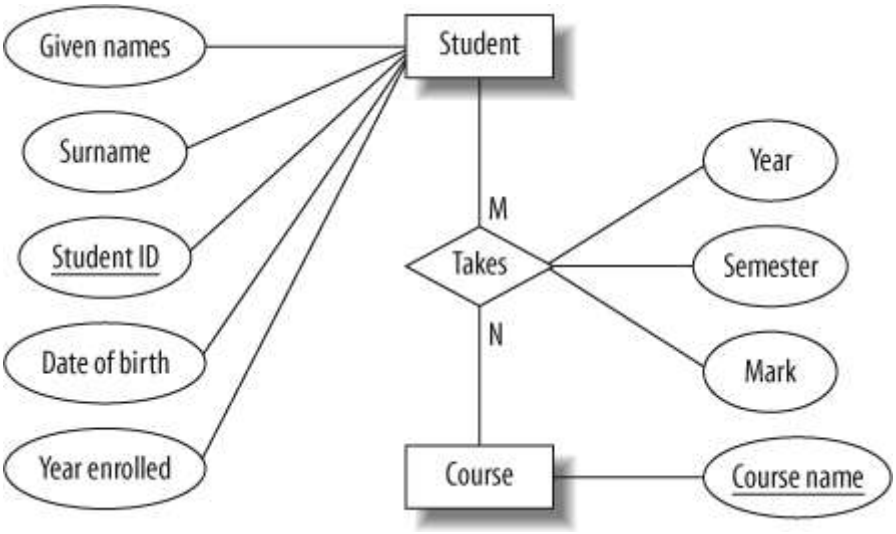

Figure 4-6. The ER diagram representation of student grades as a separate entity

in the requirements to entities, and to map the verbs to relations. For example, in the statement, "A degree program is made up of one or more courses," we can identify the entities "program" and "course," and the relationship "is made up of." Similarly, in the statement, "A student enrolls in one program," we can identify the entities "student" and "program," and the relationship "enrolls in." Of course, we can choose different terms for entities and relationships than those that appear in the relationships, but it's a good idea not to deviate too far from the naming conventions used in the requirements so that the design can be checked against the requirements. All else being equal, try to keep the design simple, and avoid introducing trivial entities where possible; i.e., there's no need to have a separate entity for the student's enrollment when we can model it as a relationship between the existing student and program entities.

Intermediate Entities

It is often possible to conceptually simplify many-to-many relationships by replacing the many-to-many relationship with a new *intermediate entity* (sometimes called an *associate entity*) and connecting the original entities through a many-to-one and a one-to-many relationship.

Consider the statement: "A passenger can book a seat on a flight." This is a many-to-many relationship between the entities "passenger" and "flight." The related ER diagram fragment is shown in Figure 4-7.

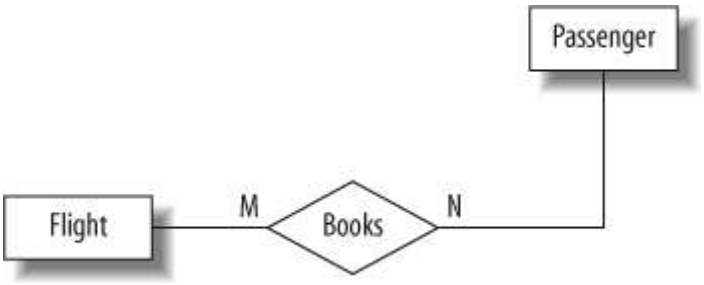

Figure 4-7. A passenger participates in an M:N relationship with flight

However, let's look at this from both sides of the relationship:

- Any given flight can have many passengers with a booking.

ships, one each way. This points us to the existence of a hidden intermediate entity, the booking, between the flight and the passenger entities. The requirement could be better worded as: "A passenger can make a booking for a seat on a flight." The related ER diagram fragment is shown in Figure 4-8.

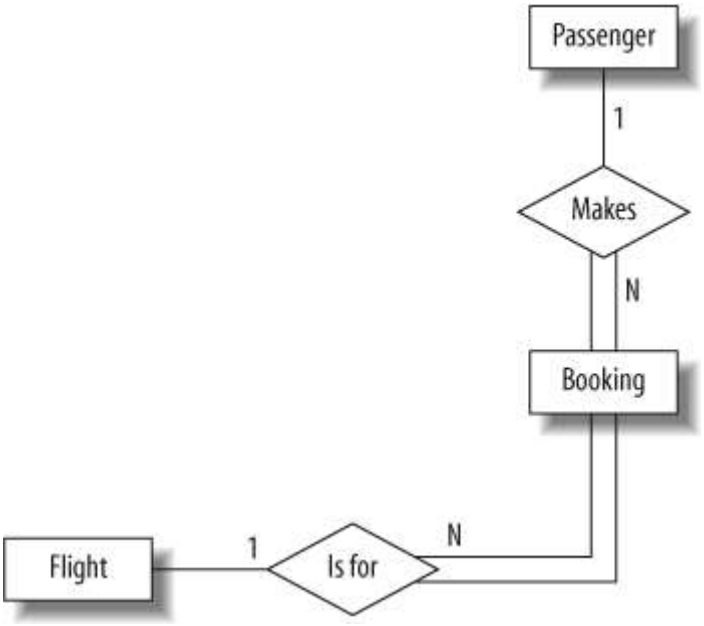

Figure 4-8. The intermediate booking entity between the passenger and flight entities

Each passenger can be involved in multiple bookings, but each booking belongs to a single passenger, so the cardinality of this relationship is 1:N. Similarly, there can be many bookings for a given flight, but each booking is for a single flight, so this relationship also has cardinality 1:N. Since each booking must be associated with a particular passenger and flight, the booking entity participates totally in the relationships with these entities. This total participation could not be captured effectively in the representation in Figure 4-7. (We described partial and total participation earlier in Partial and Total Participation.")

Weak and Strong Entities

Context is very important in our daily interactions; if we know the context, we can work with a much smaller amount of information. For example, we generally call family members by only their first name or nickname. Where ambiguity exists, we add further information such as the

tify it in the context of its parent. We could simply list a child's first name on the assumption that a customer will never have several children with the same first name. Here, the child entity is a *weak* entity, and its relationship with the customer entity is called an *identifying relationship*. Weak entities participate totally in the identifying relationship, since they can't exist in the database independently of their owning entity.

In the ER diagram, we show weak entities and identifying relationships with double lines, and the partial key of a weak entity with a dashed underline, as in Figure 4-9. A weak entity is uniquely identified in the context of its regular (or *strong*) entity, and so the full key for a weak entity is the combination of its own (partial) key with the key of its owning entity. To uniquely identify a child in our example, we need the first name of the child and the email address of the child's parent.

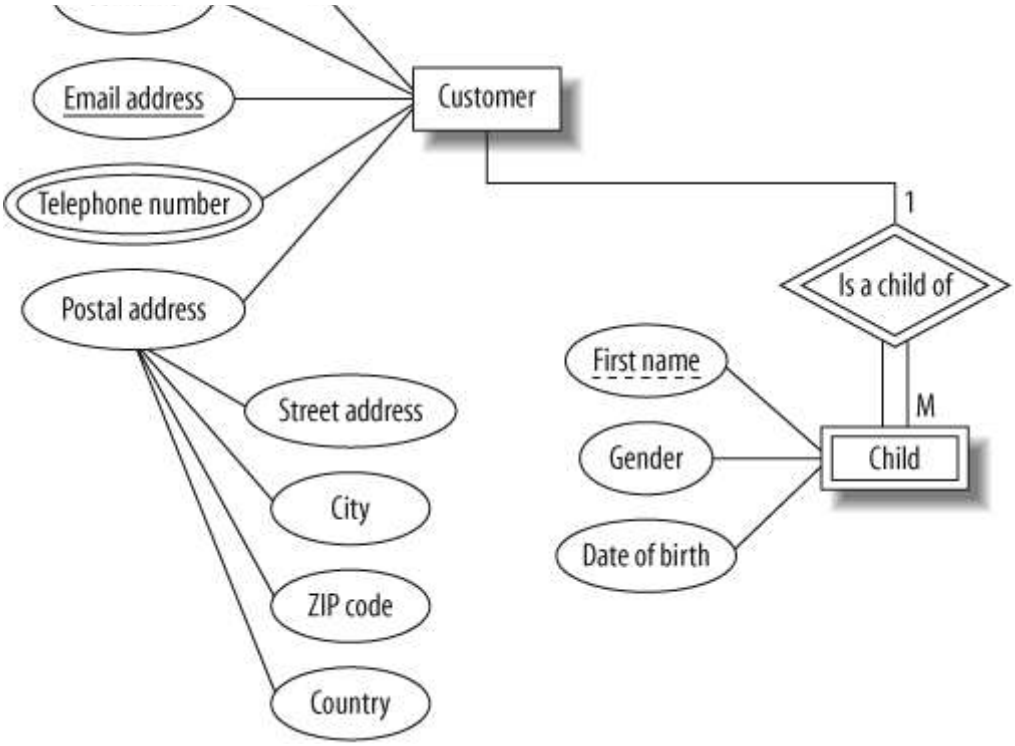

Figure 4-9. The ER diagram representation of a weak entity

Figure 4-10 shows a summary of the symbols we've explained for ER diagrams.

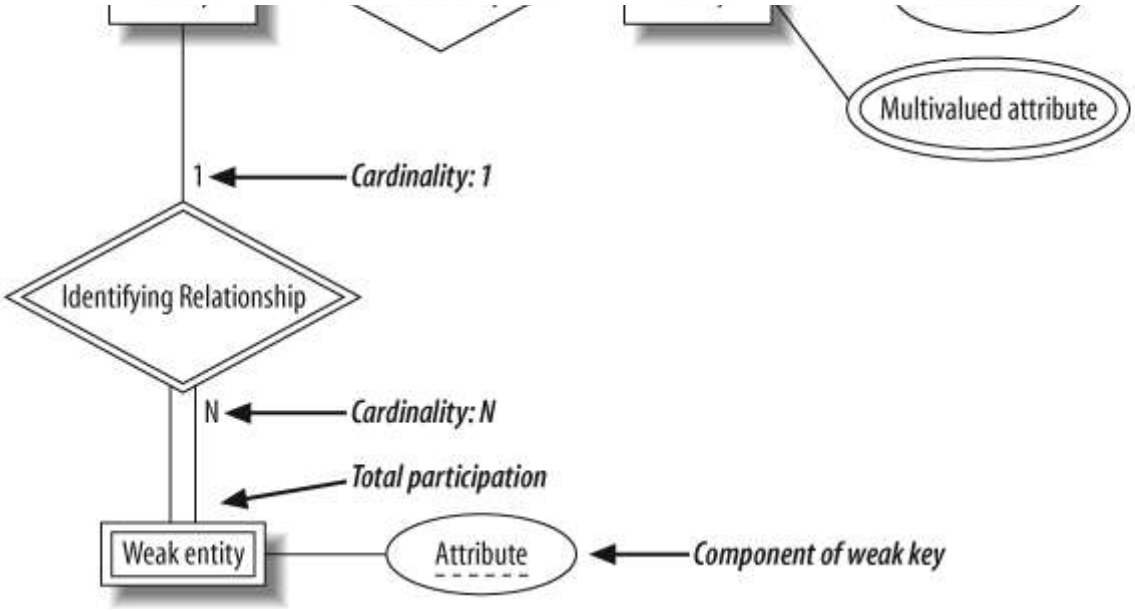

Figure 4-10. Quick summary of the ER diagram symbols

Diversity

O'Reilly for marketers

SUPPORT

Contact us

Newsletters

Privacy policy

WATCH ON YOUR BIG SCREEN

View all O'Reilly videos, Superstream events, and Meet the Expert sessions on your home TV.

DO NOT SELL MY PERSONAL INFORMATION

INTERNATIONAL

Australia & New Zealand

Hong Kong & Taiwan

India

Indonesia

Japan