4

The Basic ER Diagram
A Data Modeling Schema

4.1 INTRODUCTION

This chapter begins by describing a data modeling approach and then introduces entity relationship (ER) diagrams. The concepts of entities, attributes, relationships, and keys are introduced. The first three steps in an ER design methodology are developed. Step 1 begins by building a one-entity diagram. Step 2 concentrates on using structured English to describe a database. Step 3, the last section in this chapter, discusses mapping the ER diagram to a relational database. These concepts—the diagram, structured English, and mapping—evolve together as the book progresses. At the end of the chapter, we also begin a running case study, which is continued in the following chapters.

4.2 WHAT IS A DATA MODELING SCHEMA?

A data modeling schema is a method that allows us to model or illustrate a database. This is often in the form of a graphic diagram, but other means of communication are also desirable; people who are not in the computer field may or may not understand diagrams and graphics. The ER diagram is a graphic tool that facilitates data modeling. ER diagrams are a subset of “semantic models” in database parlance. Semantic models refer to models that intend to elicit meaning from data. ER diagrams are not the only semantic modeling tools, but are common and popular.
When we begin to discuss the contents of a database, the data model helps us to decide which piece of data goes with which other piece(s) of data on a conceptual level. An early concept concerning the database is to recognize that there are levels of abstraction we can use in discussing databases. For example, if we were to discuss the filing of “names,” we could discuss (a) abstractly, that is, “We will file names of people we know”; or (b) concretely, that is, “We will file first, middle, and last names (20 characters each) of people we know, so that we can retrieve the names in alphabetical order by last name, and we will put this data in a spreadsheet format in package x.”

If a person is designing a database, the first step is to abstract, then refine the abstraction. The longer one stays away from the concrete details of logical models (relational, hierarchical, network) and physical realizations (fields [how many characters, the data type, … ] and files [relative, spreadsheet, … ]), the easier it is to change the model and to decide how the data will eventually be physically realized (stored). When we use the term field or file, we will be referring to physical data as opposed to conceptual data.

**Mapping** is the process of choosing a logical model and then moving to a physical database file system from a conceptual model (the ER diagram). A physical file loaded with data is necessary to actually obtain data from a database. Mapping is the bridge between the design concept and physical reality. In this book, we concentrate on the relational database model due to its ubiquitousness in the contemporary database models.

### 4.2.1 So, What Is an Entity Relationship Diagram?

The **ER diagram** is a semantic data modeling tool used to accomplish the goal of abstractly describing or portraying data. Abstractly described data is called a conceptual model. Our conceptual model will lead us to a “schema.” A schema implies a permanent, fixed description of the structure of the data. Therefore, when we agree that we have captured the correct depiction of reality within our conceptual model, our ER diagram, we can call it a schema.

An ER diagram could also be used to document an existing database by reverse engineering it, but in introducing the subject, we focus on the idea of using an ER diagram to model a to-be-created database, and we deal with reverse engineering in further discussion.
4.3 DEFINING A DATABASE— SOME DEFINITIONS: ENTITY, RELATIONSHIP, ATTRIBUTE

As the name implies, an entity relationship diagram models data as entities and relationships. An entity is a thing about which we store data (e.g., a person, a bank account, a building). In the original presentation, Chen (1976) described an entity as a “thing which can be distinctly identified.” So an entity may be a person, place, object, event, or concept about which we wish to store data. A relationship is a connection between entities. An attribute is the category of data that describes an entity or relationship.

An entity represents a type or class of something and should be named accordingly. The following are some examples of entities:

- Examples of a person entity would be EMPLOYEE, VET, or STUDENT.
- Examples of a place entity would be STATE or COUNTRY.
- Examples of an object entity would be BUILDING, AUTO, or PRODUCT.
- An example of an event entity would be SALES, RETURNS, or REGISTRATION.
- An example of a concept entity would be ACCOUNT or DEPARTMENT.

The name of an entity should be generic. The name should be able to accommodate changes “over time.” For example, if we were modeling a business and the business made donuts, we might consider creating an entity called DONUT. But, how long will it be before this business evolves into making more generic pastry? If it is anticipated that the business will involve pastry of all kinds rather than just donuts, perhaps it would be better to create an entity called PASTRY—it may be more applicable over time. In this case, an entity “business” is too generic because you want to record data about donuts or pastry—components of the business.

In older data-processing circles, we would have referred to an entity as a “record,” but the term record is too physical and too confining; record gives us a mental picture of a physical thing, and to work at the conceptual level, we want to avoid device-oriented terms. In a database context, it is unusual to store information about one entity, so we think of storing collections of data about entities; such collections are called entity sets. Entity sets correspond to the older concept of “files,” but a file usually connotes a
physical thing, and hence we abstract the concept of the file (entity set) as well as the concept of a record (entity). As an example, suppose we have a company that has customers. You would imagine that the company had a customer entity set with individual customer entities in it.

An entity may be very broad (e.g., a person), or it may be narrowed by the application for which data is being prepared (like a student or a customer). “Broad” entities, which cover a whole class of objects, are sometimes called *generalizations* (e.g., person), and “narrower” entities are sometimes called *specializations* (e.g., student). In further diagrams (in this book), we revisit generalizations and specializations, but for now, we concern ourselves with an application level at which there are no subgroups (specializations) or supergroups (generalizations) of entities.

When we speak of capturing data about a particular entity, we refer to this as an *instance*. An entity instance is a single occurrence of an entity. For example, if we create an entity called TOOL, and if we choose to record data about a screwdriver, then the screwdriver “record” is an instance of TOOL. Each instance of an entity must be uniquely identifiable so that each instance is separate and distinctly identifiable from all other instances of that type of entity. In a customer entity set, you might imagine that the company would assign a unique customer number, for example. This unique identifier is called a *key*.

A *relationship* is a link or association between entities. Relationships are usually denoted by verb phrases. We begin by expanding the notion of an entity (in this chapter and the next), and then we come back to the notion of a relationship (in Chapter 6) once we have established the concept of an entity.

An *attribute* is a property or characteristic of an entity. For example, an entity, AUTOMOBILE, has attributes type, color, vehicle_id, and so on.

### 4.3.1 A Beginning Methodology

Database modeling begins with a description of “what is to be stored.” Such a description may come from anyone; we will call the describer the “user.” For example, Ms. Smith of Acme Parts Company comes to you asking that you design a database of parts for her company. Ms. Smith is the user. You are the database designer. What Ms. Smith tells you about the parts will be the database description.

As a starting point in dealing with a to-be-created database, we identify a central, “primary” entity—a category about which we will store data. For example,
if we wanted to create a database about students and their environment, then one entity would be STUDENT (our characterization of an entity will always be in the singular). Having chosen one first primary entity, STUDENT, we then search for information to be recorded about our STUDENT (attributes). This methodology of selecting one primary entity from a data description is our first step in drawing an ER diagram and hence the beginning of the requirements phase of software engineering for our database.

Once the primary entity has been chosen, we then ask what information we want to record about our entity. In our STUDENT example, we add some details about the STUDENT—details that will qualify, identify, classify, or express the state of the entity (in this case, the STUDENT entity). These details or contents of entities are called attributes.* Some example attributes of STUDENT would be the student’s name, student number, major, address, and so on—information about the student. Keep in mind that in this process of selecting attributes, the user should be able to tell you what data he or she wishes to keep.

4.3.2 ER Design Methodology

*Step 1. Select one primary entity from the database requirements description and show attributes to be recorded for that entity.*

Requirements definition is the first phase of software engineering in which the systems analyst tries to find out what a user wants. In the case of a database, an information-oriented system, the user will want to store data. Now that we have chosen a primary entity and some attributes, our task will be to (a) draw a diagram of our first impression entity (our primary entity), (b) translate the diagram into English, and (c) present the English (and the diagram) back to the user to see if we have it right and then progress from there.

Step c is called feedback in software engineering. The process of refining via feedback is a normal process in the requirements/specification phases. The feedback loop is essential in arriving at the reality of what one wants to depict from both the user and analyst viewpoints. First, we show how to draw the entity, and then we present guidelines on converting our diagram into English.

---

* C. J. Date (1995), An Introduction to Database Systems, 6th edition, preferred the word “property” to “attribute” because it is more generic and because attribute is used in other contexts. We use attribute because we believe it to be more commonly used.
CHECKPOINT 4.1

1. Of the following items, determine which could be an entity and state why: automobile, college class, student, name of student, book title, number of dependents.
2. Why are entities not called files or records?
3. What are entity sets?
4. Why do we need entity relationship diagrams?
5. What are attributes? List attributes of the entities you found in question 1?
6. What is a relationship?

4.4 A FIRST “ENTITY-ONLY” ER DIAGRAM: AN ENTITY WITH ATTRIBUTES

To recap our example, we have chosen an example with a primary entity from a student information database: the student. Again, note that “a student” is something that we want to store information about (the definition of an entity). In this chapter, we do not concern ourselves with any other entities.

Let us think about some attributes of the entity STUDENT. That is, what are some attributes a student might have? A student has a name, an address, and an educational connection. We call the educational connection a school. We have picked three attributes for the entity STUDENT, and we have also chosen a generic label for each: name, address, school.

We begin our first venture into ER diagrams with a “Chen-like” model. Chen (1976) introduced the idea of the ER diagrams. Chen and others have improved the ER process over the years, and while there is no standard ER diagram model, the Chen-like model and variants thereof are common. After the Chen-like model, we introduce other models. We briefly discuss the Barker/Oracle-like model in Chapter 12. Chen-like models have the advantage that one need not know the underlying logical model to understand the design. Barker models and some other models require a full understanding of the relational model, and the diagrams are affected by relational concepts.

To begin, in the Chen-like model, we will do as Chen originally did and put the entities in boxes and show attributes nearby. One way to depict attributes is to put them in circles or ovals appended to the boxes (refer to Figure 4.1a and Figure 4.1b). Figure 4.1c is an alternative style of depicting attributes. The alternative attribute style (Figure 4.1c) is not as descriptive but is more compact and may be used if Chen-like diagrams become cluttered.
There are several ways of depicting attributes. We have illustrated the model of an “attribute in an oval” (Chen-like model) because it is common and useful. Refer to Figures 4.2a, 4.2b, and 4.2c for some alternate models for attributes. There are benefits to alternate forms for depicting attributes. The standard form of the Chen-like model with ovals and boxes is good for conceptualizing; it is easily changed and very clear regarding which attribute goes where. The concise forms (Figure 4.1c and other variants shown in Figures 4.2a, 4.2b, and 4.2c) are easily created from the standard form and are sometimes more useful for documentation when space is a concern.
74 • Database Design Using Entity-Relationship Diagrams

Figures 4.1b and 4.1c show an ER diagram with one entity, STUDENT, and three attributes: name, address, and school. If more attributes were added to our conceptual model, such as phone and major, they would be appended to the entity (STUDENT is the only entity we have so far), as can be seen in Figure 4.3.

4.5 MORE ABOUT ATTRIBUTES

Attributes are characteristics of entities that provide descriptive detail about the entities. There are several different kinds of attributes: simple or atomic, composite, multivalued, and derived. The properties of an attribute
are its name, description, format, and length, in addition to its atomicity. Some attributes may be considered as unique identifiers for an entity. In this section, we also introduce the idea of a key attribute, a unique identifier for an entity.

4.5.1 The Simple or Atomic Attribute

Simple or atomic attributes cannot be broken down further or subdivided, hence the notion “atomic.” One may examine the domain of values* of an attribute to elicit whether an attribute is simple or not. An example of a simple or atomic attribute would be Social Security number; a person would be expected to have only one, undivided Social Security number.

Other tests of whether an attribute is simple or atomic will depend entirely on the circumstances that the database designer encounters—the desire of the user for which the database is being built. For example, a phone number attribute could be treated as simple in a particular database design, but in another scenario we may want to divide the phone number into two distinct parts, area code and the number. Another example of

---

* The domain of values is the set of values that a given attribute may take on. The domain consists of all the possible legal values that are permitted on an attribute. A data type is a broader term used to describe attributes, but data type includes the idea of which operations are allowable. Since people creating a database are usually more concerned about storage and retrieval, database data types usually just focus on the domain of values.
when the use of the attribute in the database will determine if the attribute is simple or atomic is a birthdate attribute. If we are setting up a database for a veterinary hospital, it may make sense to break a birthdate field up into month, day, and year since it will make a difference in treatment if a young animal is 5 days old versus if it is 5 months or 5 years old. Hence, in this case birthdate would be a composite attribute. For a RACE HORSE database, however, it may not be necessary to break up a birthdate field into month/day/year since all horses are dated only by the year in which they are born. In this case, birthdate, consisting of only the year, would be atomic.

If an attribute is nonatomic, it needs to be depicted as such on the ER diagram. The following sections deal with these more complicated, nonatomic attribute ideas: the composite attribute and the multivalued attribute.

4.5.2 The Composite Attribute

A composite attribute, sometimes called a group attribute, is an attribute that is formed by combining or aggregating related attributes. The names chosen for composite attributes should be descriptive and general. The concept of name is adequate for a general description, but it may be desirable to be more specific about the parts of this attribute. Most data-processing applications divide the name into component parts. Name, then, is called a composite attribute or an aggregate because it is usually composed of a first name, a last name, and a middle initial—subattributes, if you will. The way that composite attributes are shown in ER diagrams in the Chen-like model is illustrated in Figure 4.4. The subattributes, like first name, middle name, and last name, are called simple, atomic, or elementary attributes. The word aggregate is used in a different sense in some database query languages, and to avoid confusion, we do not call composite attributes aggregates; we use the word composite.

The test of whether an attribute will be composite (or not) will depend entirely on the circumstances that the database designer encounters—the desire of the user requesting the database be built. For example, in one database it may not be important to know exactly which city, state, or zip code a person comes from, so an address attribute in that database may not be broken up into its component parts; it may just be called address. In another database, it may be important to know which city and state a person is from, so in this second database we would have to break up the
address attribute into street address, city, state, and zip code, making the address attribute a composite attribute.

4.5.3 The Multivalued Attribute

Another type of nonsimple attribute that has to be managed is called a multivalued attribute. The multivalued attribute, as the name implies, may take on more than one value for a given occurrence of an entity. For example, the attribute school could easily be multivalued if a person attends (or has attended, depending on the context of the database) more than one school. As a counter example, most people go by only one name; hence, the grouping name is not multivalued. The multivalued attribute called school is depicted in Figure 4.5 (Chen-like model) as a double oval, which illustrates the situation for which a database will
store data about students who may have attended more than one school. Although we have chosen to illustrate school as a multivalued attribute, we do not mean to imply that this will always be the case in all databases. In fact, the attribute school may well be single valued in some databases. The idea of school may mean the current (or just previous) school as opposed to all schools attended. If the subjects about whom we are storing data can attend only one school at a time (and that is what we want to depict), then the attribute school may well be a single-valued attribute.

Again, the test of single versus multivalued will depend entirely on the circumstances that the database designer encounters—the desire of the user of the to-be-built database. It is recommended that if the sense of the database is that the attribute school means “current school,” then the attribute should be called “current school” and illustrated as a single-valued attribute. We show a multivalued attribute in Figure 4.5. This diagram implies that multiple schools may be recorded for each student.

### 4.5.4 The Derived Attribute

Derived attributes are attributes that the user may envision but may not be recorded per se. These derived attributes may be calculated from other data in the database. An example of a derived attribute would be an age, which could be calculated once a student’s birth date is entered. In the Chen-like model, a derived attribute is shown in a dashed oval (as shown in Figure 4.5b).

### 4.5.5 Keys

A database is used to store data for retrieval. An attribute that may be used to find a particular entity occurrence is called a **key**. As we model our database with the ER models, we may find that some attributes naturally seem to be keys. If an attribute may be thought of as a unique identifier for an entity, it is called a **candidate key**. When a candidate key is chosen to be the unique identifier, it becomes the **primary key** for the entity.

As an example of keys, suppose we add an attribute called student_number to our STUDENT entity example. We might well consider a student_number to be a unique identifier for the entity—a candidate key because of uniqueness. Name is often unique, but not necessarily so. Members of the same class often share last names. Address may or may not be a unique
identifier and hence is not a likely candidate key. Siblings who take classes together could easily have the same address. The point is that schools often choose to assign a unique student number to each student to be able to find student records—the idea of a key is to provide a unique way to find an entity instance (a particular record).

Some schools also choose to record a Social Security number (SSN) as an attribute. A SSN is also unique and hence a candidate key along with student_number. If both SSN and student_number were recorded, then the designer would have to choose which candidate key would be the primary key. In our case, we choose not to record a SSN. The STUDENT entity with the unique identifier student_number, added as a key, is depicted in Figure 4.6.

FIGURE 4.5
(a) STUDENT entity with a multivalued attribute.
In the Chen-like ER model, attributes, which are *unique identifiers* (candidate keys), are usually underlined (as shown in Figure 4.6). A unique identifier may be an attribute or a combination of attributes. It is not necessary to choose which candidate key will be the primary key at this point, but one could do so. When there is only one candidate key, we will generally speak of it as the primary key simply because it is obvious that the primary key is a candidate key. In Figure 4.6, we also depict the short form of the ER diagram (at the bottom of the figure) with composite attributes and multivalued attributes as well as primary keys. The composite attributes are listed with its component parts, and the multivalued attributes are enclosed in parentheses in the abbreviated form.

Finally, while on the subject of keys, we will have situations in the ER diagram (in the Chen-like model) for which no key is obvious or intended. Entities that have at least one identified key are called *strong* entities. In Chen’s (1976) original article, strong entities were called *regular* entities. Some entities will be discovered that depend on other entities for their being

---

**FIGURE 4.5**
(b) STUDENT entity with a derived attribute: age.
(and hence their identification). Chen called those entities that rely on other entities for their existence weak entities. We often are able to recognize these weak entities because they may not have candidate keys, although the actual meaning of a weak entity is “one that depends on another for existence.” As Chen did, we follow the Chen-like ER notation and call such entities weak entities—weak because they will have to depend on some other entity to furnish a unique identifier to give the entity a reason to be recorded.

Although a weak entity may have a candidate key, it would not be a strong entity. We depict weak entities in the Chen-like ER diagrams with double boxes (see Figure 4.7). For now, we concentrate on those entities...
that have keys, the strong entities, and we will reconsider situations for which no key is obvious, the weak entities, later.

**CHECKPOINT 4.2**

1. Describe the basic types of data representation schemas used in ER modeling.
2. What notation is used to diagrammatically show an entity in the Chen-like ER model?
3. How do we diagrammatically show attributes in the Chen-like ER model?
4. How do we show composite attributes in the Chen-like ER model?
5. Draw an entity representation for the entity “building,” with the following attributes: building name, occupancy, and whether or not it has an elevator (yes/no).
6. Embellish the building entity to include the building superintendent’s name (first, middle, and last). Does this have to be a composite attribute? Why or why not?
7. Embellish the building entity to include the address of the building, which will be the primary key.
8. Again, embellish the building entity to include names (and only the names) of the janitorial staff.
9. Add a multivalued attribute to the building entity.
10. How many attributes can an entity have?

### 4.6 ENGLISH DESCRIPTION OF THE ENTITY

Now that we have an entity with attributes, we want to prepare the first feedback to the user: the English description. Users will not likely want to study the entity diagram, but they might want to hear what you, the analyst, think you heard. For an English description, we use a “structured” English grammar and substitute the appropriate information from the entity diagram.

#### 4.6.1 The Method

The template for the structured English for single entities is as follows:

Let Entity be the name of the entity and att(j) be the attributes. The order of the attributes is not important, so \( j = 1, 2, \ldots \) is assigned arbitrarily. Suppose that there are \( n \) attributes so far. The generalized English equivalent of our diagram is presented next.

#### 4.6.1.1 The Entity

This database records data about Entity. For each Entity in the database, we record \( att(1), att(2), att(3), \ldots, att(n) \).

#### 4.6.1.2 The Attributes

For atomic attributes, \( att(j) \):

For each Entity, there always will be one and only one \( att(j) \). The value for \( att(j) \) will not be subdivided.

For composite attributes, \( att(j) \):

For each Entity, we will record \( att(j) \), which is composed of \( x, y, z, \ldots \). \((x, y, z)\) are the component parts of \( att(j) \).
For multivalued attributes, \( att(j) \):

- **For each Entity**, we will record \( att(j) \)’s. There may be more than one \( att(j) \) recorded for each Entity.

For derived attributes, \( att(j) \):

- **For each Entity**, there may exist \( att(j) \)’s, which will be derived from the database.

### 4.6.1.3 The Keys

For the key(s):

(a) more than one candidate key (strong entity):

- **For each Entity**, we will have the following candidate keys: \( att(j) \), \( att(k) \), … (where \( j, k \) are candidate key attributes).

(b) one candidate key (strong entity):

- **For each Entity**, we will have the following primary key: \( att(j) \).

(c) no candidate keys (weak entity):

- **For each Entity**, we do not assume that any attribute will be unique enough to identify individual entities without the accompanying reference to Entity1 (i.e., some other entity), the owner Entity.\(^*\)

(d) no candidate keys (intersecting entity): This is discussed next.

### 4.6.2 ER Design Methodology

**Step 2. Use structured English for entities, attributes, and keys to describe the database that has been elicited.**

**Step 3. Show some sample data.**

Sample data also helps describe the database as it is perceived.

### 4.6.3 Examples

We now revisit each of our figures and add an English description to each.

#### 4.6.3.1 Figure 4.3 Example

First, reconsider Figure 4.3. There are no multivalued or composite attributes. \( Entity = \text{STUDENT} \), \( att(1) = \text{name} \), \( att(2) = \text{school} \), and so on (\( j \)

\(^*\) The details of the weak entity/strong entity relationship will become clearer as we introduce relationships in Chapter 5.
assigned arbitrarily). The English “translation” of the entity diagram using
the templates is discussed next.

4.6.3.1.1 The Entity
This database records data about STUDENTS. For each STUDENT in the data-
base, we record a name, a school, an address, a phone number, and a major.

4.6.3.1.2 The Attributes
   For each STUDENT, there will be one and only one name. The value
   for name will not be subdivided (note that in Figure 4.3 we did not
divide name).
   For each STUDENT, there will be one and only one major. The value for
   major will not be subdivided.
   For each STUDENT, there will be one and only one address. The value
   for address will not be subdivided.
   For each STUDENT, there will be one and only one school. The value
   for school will not be subdivided.
   For each STUDENT, there will be one and only one phone. The value for
   phone will not be subdivided.

4.6.3.1.3 The Keys
For each STUDENT, we do not assume that any attribute will be unique
enough to identify individual entities. (Remember that we are describing
Figure 4.3.)

4.6.3.1.4 Sample Data
In addition to these descriptions, some sample data is often helpful in showing
the user what you have proposed. Sample data for Figure 4.3 are as follows:

<table>
<thead>
<tr>
<th></th>
<th>major</th>
<th>address</th>
<th>school</th>
<th>phone number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Smith</td>
<td>Cosc</td>
<td>123 4th St</td>
<td>St. Helens</td>
<td>222-2222</td>
</tr>
<tr>
<td>Jones</td>
<td>Acct</td>
<td>222 2nd St</td>
<td>PS 123</td>
<td>333-3333</td>
</tr>
<tr>
<td>Saha</td>
<td>Eng</td>
<td>284 3rd St</td>
<td>Canton</td>
<td>345-3546</td>
</tr>
<tr>
<td>Kapoor</td>
<td>Math</td>
<td>20 Living Cr</td>
<td>High</td>
<td>435-4534</td>
</tr>
</tbody>
</table>

4.6.3.2 Figure 4.4 Example
Now, consider Figure 4.4. This figure has a composite attribute, name. The
English translation of this entity diagram would be as given next.
4.6.3.2.1 The Entity
This database records data about STUDENTs. For each STUDENT in the database, we record a name, a school, and an address.

4.6.3.2.2 The Attributes
For each STUDENT, there will be one and only one name. The value for name will be subdivided into first name, last name, and middle initial. For each STUDENT, there will be one and only one address. The value for address will not be subdivided. For each STUDENT, there will be one and only one school. The value of the school will not be subdivided.

4.6.3.2.3 The Keys
For each STUDENT, we do not assume that any attribute will be unique enough to identify individual entities.

4.6.3.2.4 Sample Data

<table>
<thead>
<tr>
<th>STUDENT</th>
<th>name.first</th>
<th>name.last</th>
<th>name.mi</th>
<th>school</th>
<th>address</th>
</tr>
</thead>
<tbody>
<tr>
<td>Richard</td>
<td>Earp</td>
<td>W</td>
<td></td>
<td>U. Alabama</td>
<td>222 2nd St</td>
</tr>
<tr>
<td>Boris</td>
<td>Backer</td>
<td></td>
<td></td>
<td>Heidleburg</td>
<td>333 Dreistrasse</td>
</tr>
<tr>
<td>Helga</td>
<td>Hogan</td>
<td>H</td>
<td></td>
<td>U. Hoover</td>
<td>88 Half Moon Ave</td>
</tr>
<tr>
<td>Arpan</td>
<td>Bagui</td>
<td>K</td>
<td></td>
<td>Northern School</td>
<td>33 Bloom Ave</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>South Bend</td>
</tr>
<tr>
<td>Hema</td>
<td>Malini</td>
<td></td>
<td></td>
<td></td>
<td>100 Livingstone</td>
</tr>
</tbody>
</table>

4.6.3.3 Figure 4.5a Example
Next consider Figure 4.5a. This figure has a composite as well as a multivalued attribute. The English translation of this entity diagram is given next.

4.6.3.3.1 The Entity
For the entity, this database records data about STUDENTs. For each STUDENT in the database, we record a name, schools, and an address.

4.6.3.3.2 The Attributes
For each STUDENT, there will be one and only one name. The value for name will be subdivided into first name, last name, and middle initial.
For each STUDENT, there will be one and only one address. The value for address will not be subdivided.
For each STUDENT, we will record schools. There may be more than one school recorded for each student.

4.6.3.3.3 The Keys
For each STUDENT, we do not assume that any attribute will be unique enough to identify individual entities.

4.6.3.3.4 Sample Data

<table>
<thead>
<tr>
<th>STUDENT</th>
<th>name.first</th>
<th>name.last</th>
<th>name.mi</th>
<th>school</th>
<th>address</th>
</tr>
</thead>
<tbody>
<tr>
<td>Richard</td>
<td>Earp</td>
<td>W</td>
<td></td>
<td>U. Alabama, Mountain</td>
<td>222 2nd St</td>
</tr>
<tr>
<td>Boris</td>
<td>Backer</td>
<td></td>
<td></td>
<td>Heidelberg, Volcano</td>
<td>333 Dreistrasse</td>
</tr>
<tr>
<td>Helga</td>
<td>Hogan</td>
<td>H</td>
<td></td>
<td>U. Hoover, St. Helens</td>
<td>88 Half Moon Ave</td>
</tr>
<tr>
<td>Arpan</td>
<td>Bagui</td>
<td>K</td>
<td></td>
<td>Northern School</td>
<td>33 Bloom Ave</td>
</tr>
<tr>
<td>Hema</td>
<td>Malini</td>
<td></td>
<td></td>
<td>South Bend</td>
<td>100 Livingstone</td>
</tr>
</tbody>
</table>

4.6.3.4 Figure 4.6 Example
Consider Figure 4.6. This figure has composite, multivalued, and key attributes. The English translation of this entity diagram is as follows.

4.6.3.4.1 The Entity
This database records data about STUDENTs. For each STUDENT in the database, we record a name, schools, an address, and a student_number.

4.6.3.4.2 The Attributes
For each STUDENT, there will be one and only one name. The value for name will be subdivided into first name, last name, and middle initial.
For each STUDENT, there will be one and only one address. The value for address will not be subdivided.
For each STUDENT, we will record schools. There may be more than one school recorded for each student.
4.6.3.4.3 The Keys
For each STUDENT, there is an attribute—student_number—that will be unique enough to identify individual entities.

4.6.3.5 Figure 4.7 Example
Finally, consider Figure 4.7 (top figure). This figure shows a strong entity. We combine the grammar a little to keep the methodology from being overly repetitive. The English translation of this entity diagram follows.

4.6.3.5.1 The Entity
This database records data about AUTOMOBILEs. For each AUTOMOBILE in the database, we record a make, body_style, year, color, and vehicle_id.

4.6.3.5.2 The Attributes
Each AUTOMOBILE will have one and only one make, body_style, year, color, and vehicle_id. None of these attributes will be subdivided.

4.6.3.5.3 The Keys
For each AUTOMOBILE, the attribute, vehicle_id, will be unique enough to identify individual entities.

The bottom of Figure 4.7 shows a weak entity. The only difference between the strong and weak entity description involves the key phrase, which may not exist in the weak entity.

Before leaving this introductory chapter on ER diagrams, we show the other major component of ER diagrams. Figure 4.8 shows a relationship between two entities, an AUTOMOBILE and a STUDENT. The concept of relationships is discussed elaborately in Chapter 6. A relationship adds action to the diagram. For example, the relationship in Figure 4.8 might be that STUDENTS drive AUTOMOBILEs.

Our ER design methodology has evolved to the following so far:

Step 1. Select one primary entity from the database requirements description and show attributes to be recorded for that entity. Label keys if appropriate.
Step 2. Use structured English for entities, attributes, and keys to describe the database that has been elicited.
Step 3. Show some sample data.
4.7 MAPPING THE ENTITY DIAGRAM TO A RELATIONAL DATABASE

Having illustrated the idea of the entity and the attribute, we now turn to a semiphysical realization of the concepts. We say semiphysical because we are really not concerned with the actual physical file that is stored in memory; rather, we are concerned with placing data into relational tables that we will visualize as a physical organization of data. Basically, a relational database is a database of two-dimensional tables called relations. The tables are composed of rows and columns. The rows are often called
tuples and the columns attributes. In a relational database, all attributes (table columns) must be atomic, and keys must not be null. In addition, in relational databases, it is not usually necessary to know the actual physical location of the data in memory.

The process of converting an ER diagram into a database is called mapping. We concern ourselves only with the relational model; hence, as the chapters in this book develop, we consider mapping rules to map ER diagrams to relational databases.

We start with a rule to map strong entities.

Mapping rule 1—Mapping strong entities. Develop a new table (relation) for each strong entity and make the indicated key of the strong entity the primary key of the table. If more than one candidate key is indicated on the ER diagram, choose one for the primary key.

Next, we have to map the attributes into the strong entity. Mapping rules are different for atomic attributes, composite attributes, and multi-valued attributes. First, we present the mapping rule for mapping atomic attributes.

Mapping rule 2—Mapping atomic attributes. For entities with atomic attributes, map the entities to a table and form columns for each atomic attribute.*

In discussing relational tables, it is common to abbreviate the diagram with a notation like this:

TABLENAME(attribute1, attribute2, ...)

A relational database realization of the entity diagram in Figure 4.3 would look like

STUDENT(name, phone, school, address, major)

And with some sample data:

<table>
<thead>
<tr>
<th>name</th>
<th>phone</th>
<th>school</th>
<th>address</th>
<th>major</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jones</td>
<td>932-5100</td>
<td>U. Alabama</td>
<td>123 4th St</td>
<td>Chemistry</td>
</tr>
<tr>
<td>Smith</td>
<td>932-5101</td>
<td>U. Mississippi</td>
<td>123 5th St</td>
<td>Math</td>
</tr>
</tbody>
</table>

* These mapping rules are adapted from Elmasri and Navathe (2007).
The entity name **STUDENT** would be the name of the relation (table). The attributes in the entity diagram become the column headings. The actual table with data, a realization of a relation, is provided as an example of the type of data you might expect from such a relation. The ordering of the columns is irrelevant to the relational database as long as once the ordering is chosen, we stay with it. Recall that the point of this example is for you, the database analyst, to communicate to the user what you think the database should look like.

What about the composite and multivalued attributes? As we mentioned, it is an axiom of the relational database that all columns be atomic. If we have a nonatomic attribute on our diagram, we have to make it atomic for the mapping to the relational database. For composite attributes, we achieve atomicity by recording only the component parts of the attribute. Our next mapping rule maps composite attributes.

**Mapping rule 3—Mapping composite attributes.** For entities with composite attributes, map entities to a table and form columns of each elementary (atomic) part of the composite attributes.

Refer to Figure 4.4. A relational database, which corresponds to the entity diagram in Figure 4.4, would be

```
STUDENT(name.first, name.last, name.mi, school, address)
```

In this shorthand notation of a relational database, the composite attribute (name) is often included with a dot notation (e.g., `name.first`).

With some sample data,

<table>
<thead>
<tr>
<th>name.first</th>
<th>last</th>
<th>name.mi</th>
<th>school</th>
<th>address</th>
</tr>
</thead>
<tbody>
<tr>
<td>Richard</td>
<td>Earp</td>
<td>W</td>
<td>U. Alabama</td>
<td>222 2nd St</td>
</tr>
<tr>
<td>Boris</td>
<td>Backer</td>
<td></td>
<td>Heidleburg</td>
<td>333 Dreistrasse</td>
</tr>
<tr>
<td>Helga</td>
<td>Hogan</td>
<td>H</td>
<td>U. Hoover</td>
<td>88 Half Moon Ave</td>
</tr>
</tbody>
</table>

**The Basic ER Diagram** • 91
A multivalued attribute is depicted in Figure 4.5a. In this entity diagram, the STUDENT entity has a composite name attribute and a multivalued school attribute. This means that a student may have more than one school recorded for his or her row. Data, which would be represented by this diagram, might look like this:

<table>
<thead>
<tr>
<th>STUDENT</th>
<th>name.first</th>
<th>name.last</th>
<th>name.mi</th>
<th>address</th>
<th>school</th>
</tr>
</thead>
<tbody>
<tr>
<td>Richard</td>
<td>Earp</td>
<td>W</td>
<td></td>
<td>222 2nd St</td>
<td>U. Alabama, St Helens,</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Mountain, Volcano</td>
</tr>
<tr>
<td>Boris</td>
<td>Backer</td>
<td></td>
<td></td>
<td>333 Dreistrasse</td>
<td>Heidleburg, Manatee U.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>UCF, UWF</td>
</tr>
<tr>
<td>Helga</td>
<td>Hogan</td>
<td>H</td>
<td></td>
<td>88 Half Moon Ave</td>
<td>U. Hoover, Mount Union U,</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Manatee U</td>
</tr>
<tr>
<td>Arpan</td>
<td>Bagui</td>
<td>K</td>
<td></td>
<td>33 Bloom Ave</td>
<td>Cambridge, USF, Harvard</td>
</tr>
<tr>
<td>Hema</td>
<td>Malini</td>
<td></td>
<td></td>
<td>100 Livingstone</td>
<td>Fashion U, Milan U</td>
</tr>
</tbody>
</table>

Note that this is not considered a relational table because the school attribute is not atomic. To be a relational table, every attribute has to be atomic. To map this multivalued attribute atomically, we follow the following mapping rule:

**Mapping rule 4—Mapping multivalued attributes.** Form a separate table for the multivalued attribute. Record a row for each value of the multivalued attribute together with the key from the original table. The key of the new table will be the concatenation of the multivalued attribute plus the key of the owner entity. Remove the multivalued attribute from the original table.
As per mapping rule 4, we require a key to map multivalued attributes; hence, we use Figure 4.6 to correctly map the multivalued attribute. Figure 4.6 would be mapped into the following two relations:

**STUDENT**(student_number, name.first, name.last, name.mi, address)

and

**STUDENT_SCHOOL**(student_number, school)

Some sample data would be

### STUDENT

<table>
<thead>
<tr>
<th>student_number</th>
<th>name.first</th>
<th>name.last</th>
<th>name.mi</th>
<th>address</th>
</tr>
</thead>
<tbody>
<tr>
<td>111-11-2222</td>
<td>Richard</td>
<td>Earp</td>
<td>W</td>
<td>222 2nd St</td>
</tr>
<tr>
<td>222-11-2222</td>
<td>Boris</td>
<td>Backer</td>
<td></td>
<td>333</td>
</tr>
<tr>
<td>234-45-4567</td>
<td>Helga</td>
<td>Hogan</td>
<td>H</td>
<td>88 Half Moon Ave</td>
</tr>
<tr>
<td>888-77-9990</td>
<td>Arpan</td>
<td>Bagui</td>
<td>K</td>
<td>33 Bloom Ave</td>
</tr>
<tr>
<td>123-45-4321</td>
<td>Hema</td>
<td>Malini</td>
<td></td>
<td>100 Livingstone</td>
</tr>
</tbody>
</table>

### STUDENT_SCHOOL

<table>
<thead>
<tr>
<th>student_number</th>
<th>school</th>
</tr>
</thead>
<tbody>
<tr>
<td>111-11-2222</td>
<td>U. Alabama</td>
</tr>
<tr>
<td>111-11-2222</td>
<td>St. Helens</td>
</tr>
<tr>
<td>111-11-2222</td>
<td>Mountain</td>
</tr>
<tr>
<td>111-11-2222</td>
<td>Volcano</td>
</tr>
<tr>
<td>222-11-2222</td>
<td>Heidleburg</td>
</tr>
<tr>
<td>222-11-2222</td>
<td>Manatee U</td>
</tr>
<tr>
<td>222-11-2222</td>
<td>UCF</td>
</tr>
<tr>
<td>222-11-2222</td>
<td>UWF</td>
</tr>
<tr>
<td>234-45-4567</td>
<td>U. Hoover</td>
</tr>
<tr>
<td>234-45-4567</td>
<td>Mount Union U</td>
</tr>
<tr>
<td>234-45-4567</td>
<td>Manatee U</td>
</tr>
<tr>
<td>888-77-9990</td>
<td>Cambridge</td>
</tr>
<tr>
<td>888-77-9990</td>
<td>USF</td>
</tr>
<tr>
<td>888-77-9990</td>
<td>Harvard</td>
</tr>
<tr>
<td>123-45-4321</td>
<td>Fashion U</td>
</tr>
<tr>
<td>123-45-4321</td>
<td>Milan U</td>
</tr>
</tbody>
</table>

In relational databases, every row of a table contains atomic attributes, and every row is unique. Therefore, a candidate key in any table is always all
of the attributes. Usually, a subset of “all of the attributes” can be found to be a key, but since no two rows are ever the same, it is always true that one candidate key is the collection of all attributes.

CHECKPOINT 4.3
1. How do you map multivalued attributes?
2. How do you map composite attributes?

4.8 CHAPTER SUMMARY
The main focus in this chapter was on developing the concept of the entity and developing a one-entity diagram using the Chen-like model. The concept of attributes was also discussed, and the last section focused on how a one-entity diagram could be mapped to a relational database. The grammar for a one-entity diagram and its attributes was also developed. This grammar is further developed in the following chapters. The next chapter discusses developing a second entity and the relationship between this second entity and the primary entity.

CHAPTER 4 EXERCISES
Note: The user should clarify the assumptions made when reporting the work.

Exercise 4.1
You want to create a database about businesses. Each business will have a name, address, the business phone number, the owner’s phone number, and the first names of the employees who work at the business. Draw the ER diagram using the Chen-like model and then write the English description for your diagrams. Compare the English to your diagrams and state any assumptions you made when drawing the diagrams. Map your diagrams to a relational database.

Which attributes would you consider composite attributes in this database? Which attributes would you consider multivalued attributes in this database? Could there be any derived attributes? What would be good keys?
Exercise 4.2

You want to create a database about the books on your shelf. Each book has authors (assume that only the last name is needed), title, publisher, courses used in (course number only). Draw the ER diagram using the Chen-like model, and then write the English description for your diagrams. Compare the English to your diagrams and state any assumptions you made when drawing the diagrams.

Which attributes would you consider composite attributes in this database? Which attributes would you consider multivalued attributes in this database? Could there be any derived attributes? What would be good keys? Map your diagram to a relational database.

BIBLIOGRAPHY


