Structuring System Data Requirements

Learning Objectives
After studying this chapter, you should be able to:

- Concisely define each of the following key data modeling terms: entity type, attribute, multivalued attribute, relationship, degree, cardinality, business rule, associative entity, trigger, supertype, subtype.
- Draw an entity-relationship (E-R) diagram to represent common business situations.
- Explain the role of conceptual data modeling in the overall analysis and design of an information system.
- Explain the role of prepackaged database models (patterns) in data modeling.
- Distinguish among unary, binary, and ternary relationships and give an example of each.
- Define four basic types of business rules in a conceptual data model.
- Relate data modeling to process and logic modeling as different views of describing an information system.

Introduction

In Chapter 7, you learned how to model and analyze two important views of an information system: (1) the flow of data between manual or automated steps and (2) the decision logic of processing data. None of the techniques discussed so far, however, has concentrated on the data that must be retained in order to support the data flows and processing described. For example, you learned how to show data stores, or data at rest, in a data flow diagram (DFD). The natural structure of data, however, was not shown. DFDs, use cases, and various processing logic techniques show how, where, and when data are used or changed in an information system, but these techniques do not show the definition, structure, and relationships within the data. Data modeling develops these missing, and crucial, pieces of description of a system.

In fact, some systems developers believe that a data model is the most important part of the statement of information system requirements. This belief is based on the following reasons. First, the characteristics of data captured during data modeling are crucial in the design of databases, programs, computer screens, and printed reports. For example, facts such as these—a data element is numeric, a product can be in only one product line at a time, a line item on a customer order can never be moved to another customer order, customer region name is limited to a specified set of values—are all essential pieces of information in ensuring data integrity in an information system.

Second, data, not processes, are the most complex aspects of many modern information systems and hence require a central role in structuring system requirements. Transaction processing systems can have considerable process complexity in validating data, reconciling errors, and coordinating the movement of data to various databases. Current systems development focuses more on management information systems (such as sales tracking), decision support systems (such as short-term cash investment), and business intelligence systems (such as market basket analysis). Such systems are more data intensive. The exact nature of processing is also more ad hoc than with transaction processing systems, so the details of processing steps cannot be anticipated. Thus, the goal is to provide a rich data resource that might support any type of information inquiry, analysis, and summarization.

Third, the characteristics about data (e.g., length, format, and relationships with other data) are reasonably permanent and have significant similarity for different organizations in the same business. In contrast, the paths and design of data flow are quite dynamic. A data model explains the inherent nature of the organization, not its transient form. Therefore, an information system design based on a data orientation, rather than a process or logic orientation, should have a longer useful life and should have common features for the same applications or domains in different organizations. Finally, structural information about data is essential for automatic program generation. For example, the fact that a customer order has many line items on it instead of just one line item affects the automatic design of a computer screen for entry of customer orders. Although a data model specifically documents the file and database requirements for an information system, the business meaning, or semantics, of data included in the data model has a broader effect on the design and construction of a system.

The most common format used for data modeling is entity-relationship (E-R) diagramming. A similar format with object-oriented analysis and design methods is class diagramming, which is included in a special section at the end of this chapter on the object-oriented development approach to data modeling. Data models that use E-R and class diagram notations explain the characteristics and structure of data independent of how the data may be stored in computer memories. A data model is usually developed iteratively, either from scratch or from a purchased data model for the industry or business area to be supported. Information systems (IS) personnel use this preliminary data model to develop an enterprise-wide data model with very broad categories of data and little detail. Next, during the definition of a project, a specific data model is built to help explain the scope of a particular systems analysis and design effort. During requirements structuring, a data model represents conceptual data requirements for a particular system. Then, after system inputs and outputs are fully described during logical design, the data model is refined before it is translated into a logical format (typically a relational data model) from which database design and physical database design are done. A data model represents certain types of business rules that govern the properties of data. Business rules are important statements of business policies that ideally will be enforced through the database and database management system ultimately used for the application you are designing. Thus, you will use E-R and class diagramming in many systems development project steps, and most IS project members need to know how to develop and read data model diagrams. Therefore, mastery of the requirements structuring methods and techniques addressed in this chapter is critical to your success on a systems development project team.
CONCEPTUAL DATA MODELING

A conceptual data model is a representation of organizational data. The purpose of a conceptual data model is to show as many rules about the meaning and interrelationships among data as are possible.

Conceptual data modeling is typically done in parallel with other requirements analysis and structuring steps during systems analysis (see Figure 8-1), as outlined in prior chapters. On larger systems development teams, a subset of the project team concentrates on data modeling while other team members focus attention on process or logic modeling. Analysts develop (or use from prior systems development) a conceptual data model for the current system and then build or refine a purchased conceptual data model that supports the scope and requirements for the proposed or enhanced system.

The work of all team members is coordinated and shared through the project dictionary or repository. This repository is often maintained by a common Computer-Aided Software Engineering (CASE) or data modeling software tool, but some organizations still use manual documentation. Whether automated or manual, it is essential that the process, logic, and data model descriptions of a system be consistent and complete because each describes different, but complementary, views of the same information system. For example, the names of data stores on the primitive-level DFDs often correspond to the names of data entities in E-R diagrams, and the data elements associated with data flows on DFDs must be accounted for by attributes of entities and relationships in E-R diagrams.

The Conceptual Data Modeling Process

The process of conceptual data modeling begins with developing a conceptual data model for the system being replaced, if a system already exists. This is essential for planning the conversion of the current files or database into the database of the new system. Further, this is a good, not a perfect, starting point for your understanding of the data requirements of the new system. Then, a new conceptual data model is built (or a standard one is purchased) that includes all of the data requirements for the new system. You discovered these requirements from the fact-finding methods employed during requirements determination. Today, given the popularity of rapid development methodologies, such as the use of predefined patterns, these requirements often evolve through various iterations from some starting point in a purchased application or database design. Even when developed from scratch, data modeling is an iterative process with many checkpoints.

![Figure 8-1: Systems development life cycle with analysis phase highlighted](image)

Conceptual data modeling is one kind of data modeling and database design carried out throughout the systems development process. Figure 8-2 shows the different kinds of data modeling and database design that go on during the overall systems development life cycle (SDLC). The conceptual data modeling methods we discuss in this chapter are suitable for the planning and analysis phases; these methods can be used with either a data model developed from scratch or based on a purchased data model. The planning phase of the SDLC addresses issues of system scope, general requirements, and content independent of technical implementation. E-R and class diagramming are suited for this phase because these diagrams can be translated into a wide variety of technical architectures for data, such as relational, network, and hierarchical architectures. A data model evolves from the early stages of planning through the analysis phase as it becomes more specific and is validated by more detailed analyses of system needs.

In the design phase, the final data model developed in analysis is matched with designs for systems inputs and outputs and is translated into a format from which physical data storage decisions can be made. After specific data storage architectures are selected, then, in implementation, files and databases are defined as the system is coded. Through the use of the project repository, a field in a physical data record can, for example, be traced back to the conceptual data attribute that represents it on a data model diagram. Thus, the data modeling and design steps in each of the SDLC phases are linked through the project repository.

Deliverables and Outcomes

Most organizations today do conceptual data modeling using E-R modeling, which uses a special notation to represent as much meaning about data as possible. Because of the rapidly increasing interest in object-oriented methods, class diagrams using unified modeling language (UML) drawing tools such as IBM’s Rational products or Microsoft Visio are also popular. We will focus first on E-R diagramming and then later show how it differs from class diagramming.

The primary deliverable from the conceptual data modeling step within the analysis phase is an E-R diagram, similar to the one shown in Figure 8-3. This figure shows the major categories of data (rectangles on the diagram) and the business
relationships between them (lines connecting rectangles). For example, Figure 8.3 shows that, for the business represented by this diagram, a SUPPLIER sometimes Supplies ITEMS to the company, and an ITEM is always Supplied by one to four SUPPLIERS. The fact that a supplier only supplies items implies that the business wants to keep track of some suppliers without designating what they can supply. This diagram includes two names on each line so that a relationship can be read in each direction. For simplicity, we will not typically include two names on lines in E-R diagrams in this book; however, this is a standard used in many organizations.

The other deliverable from conceptual data modeling is a full set of entries about data objects that will be stored in the project dictionary, repository, or data modeling software. The repository is the mechanism that links the data, processes, and logic models of an information system. For example, there are explicit links between a data model and an E-R diagram. Some important links are explained briefly here.

- Data elements included in data flows also appear in the data model, and vice versa. You must include in the data model any raw data captured and retained in a data store, and a data model can include only data that have been captured or that have been computed from captured data. Because a data model is a general business picture of data, both manual and automated data stores will be included.
- Each data store in a process model must relate to business objects (what we will call data entities) represented in the data model. For example, in Figure 7.3, the Inventory File data store must correspond to one or several data objects on a data model.

You can use an automated repository to verify these linkages.

GATHERING INFORMATION FOR CONCEPTUAL DATA MODELING

Requirements determination methods must include questions and investigations that take a data, not only a process and logic, focus. For example, during interviews with potential system users—during Joint Application Design (JAD) sessions or through requirements interviews—you must ask specific questions in order to gain the perspective on data that you need to develop or tailor a purchased data model. In later sections of this chapter, we will introduce some specific terminology and constructs used in data modeling. Even without this specific data modeling language, you can begin to understand the kinds of questions that must be answered during requirements determination. These questions relate to understanding the rules and policies by which the area to be supported by the new information system operates. That is, a data model explains what the organization does and what rules govern how work is performed in the organization. You do not, however, need to know (and often can’t fully anticipate) how or when data are processed or used to do data modeling.

You typically do data modeling from a combination of perspectives. The first perspective is generally called the top-down approach. This perspective derives the business rules for a data model from an intimate understanding of the nature of the business, rather than from any specific information requirements in computer displays, reports, or business forms. It is this perspective that is typically the basis for a purchased data model. Several very useful types of questions elicit the business rules needed for data modeling (see Aranow, 1989; Gottesdiener, 1999; and Sandifer and von Halle, 1991a, 1991b). Table 8-1 summarizes a few key questions you should ask system users and business managers so that you can develop an accurate and complete data model tailored to the particular situation. The questions in this table are purposefully posed in business terms. You can ask these questions whether you start with a clean sheet of paper or a purchased data model, but typically the questions are more obvious and thorough when you begin the data modeling project with a purchased data model for the industry or application under development. In this chapter, you will learn the more technical terms included in bold at the end of each set of questions. Of course, these technical terms do not mean much to a business manager, so you must learn how to frame your questions in business terms for your investigation.

<table>
<thead>
<tr>
<th>TABLE 8-1 Requirements Determination Questions for Data Modeling</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. What are the subjects/objects of the business? What types of people, places, things, materials, events, etc., are used or interact in this business, about which data must be maintained? How many instances of each object might exist? Data entities and their descriptions.</td>
</tr>
<tr>
<td>2. What unique characteristic (or characteristics) distinguishes each object from other objects of the same type? Might this distinguishing feature change over time or is it permanent? Might this characteristic of an object be missing even though we know the object exists? Primary key</td>
</tr>
<tr>
<td>3. What characteristics describe each object? On what basis are objects referenced, selected, qualified, sorted, and categorized? What must we know about each object in order to run the business? Attributes and secondary keys.</td>
</tr>
<tr>
<td>4. How do you do these data? That is, are you the source of the data for the organization, do you refer to the data, do you modify it, and do you destroy it? Who is not permitted to use these data? Who is responsible for establishing legitimate values for these data? Security controls and understanding who really knows the meaning of data.</td>
</tr>
<tr>
<td>5. Over what time period of time are you interested in these data? Do you need historical trends, current &quot;on-hand&quot; values, and/or estimates or projections? If a characteristic of an object changes over time, must you know the absolute values? Cardinality and time dimensions of data.</td>
</tr>
<tr>
<td>6. Are all instances of each object the same? That is, are there special kinds of each kind? Are any descriptions or handled differently by the organization? Are any objects summaries or combinations of more detailed objects? Super types, subtypes, and aggregations.</td>
</tr>
<tr>
<td>7. What events occur that imply associations among various objects? What natural activities or transactions of the business involve handling data about several objects of the same or a different type? Relationships, and their cardinality and degree.</td>
</tr>
<tr>
<td>8. Is each activity or event always handled the same way or are there special circumstances? Can events occur with only some of the associated objects, or must all objects be involved? Can the associations between objects change over time (for example, employees change departments)? Are values for data characteristics limited in any way? Integrity rules, minimum and maximum cardinality, time dimensions of data.</td>
</tr>
</tbody>
</table>
You can also gather the information you need for data modeling by reviewing specific business documents—computer displays, reports, and business forms—handled within the system. This process of gaining an understanding of data is often called a bottom-up approach. These items will appear as data flows on DFDs and will show the data processed by the system and, hence, probably the data that must be maintained in the system’s database. Consider, for example, Figure 8-4, which shows a customer order form used at Pine Valley Furniture (PVF). From this form, we determine that the following data must be kept in the database:

<table>
<thead>
<tr>
<th>PRODUCT NO</th>
<th>DESCRIPTION</th>
<th>QUANTITY ORDERED</th>
<th>UNIT PRICE</th>
</tr>
</thead>
<tbody>
<tr>
<td>M126</td>
<td>Bookcase</td>
<td>4</td>
<td>200.00</td>
</tr>
<tr>
<td>B381</td>
<td>Cabinet</td>
<td>2</td>
<td>180.00</td>
</tr>
<tr>
<td>R210</td>
<td>Table</td>
<td>1</td>
<td>600.00</td>
</tr>
</tbody>
</table>

We also see that each order is from one customer and that an order can have multiple line items, each for one product. We will use this kind of understanding of an organization’s operation to develop data models.

**INTRODUCTION TO E-R MODELING**

The basic E-R modeling notation uses three main constructs: data entities, relationships, and their associated attributes. Several different E-R notations exist, and many CASE and E-R drawing software support multiple notations. For simplicity, we have adopted one common notation for this book; this notation uses the so-called crown’s foot symbols and places data attribute names within entity rectangles. This notation is very similar to that used by many E-R drawing tools, including Microsoft Visio®. If you use another notation in courses or at work, you should be able to easily translate between notations.

An entity-relationship data model (E-R model) is a detailed, logical representation of the data for an organization or for a business area. The E-R model is expressed in terms of entities in the business environment, the relationships or associations among those entities, and the attributes or properties of both the entities and their relationships. An E-R model is normally expressed as an entity-relationship diagram (E-R diagram), which is a graphical representation of an E-R model. The notation we will use for E-R diagrams appears in Figure 8-5, and subsequent sections explain this notation.

**Entities**

An entity (see the first question in Table 8-1) is a person, place, object, event, or concept in the user environment about which the organization wishes to maintain data. An entity has its own identity that distinguishes it from each other entity. Some examples of entities follow:

- **Person**: EMPLOYEE, STUDENT, PATIENT
- **Place**: STORE, WAREHOUSE, STATE
- **Object**: MACHINE, BUILDING, AUTOMOBILE, PRODUCT
- **Event**: SALE, REGISTRATION, RENEWAL
- **Concept**: ACCOUNT, COURSE, WORK CENTER

There is an important distinction between entity type and entity instance. An entity type (sometimes called an entity class) is a collection of entities that share common properties or characteristics. Each entity type in an E-R model is given a name. Because the name represents a class or set, it is singular. Also, because an entity is an object, we use a simple noun to name an entity type. We use capital letters in naming an entity type; in an E-R diagram, the name is placed inside a rectangle representing the entity, as shown in Figure 8-6a.

An entity instance (also known simply as an instance) is a single occurrence of an entity type. An entity instance is described just once in a data model, whereas many instances of that entity type may be represented by data stored in the database. For example, there is one EMPLOYEE entity type in most organizations, but there may be hundreds (or even thousands) of instances of this entity type stored in the database.

A common mistake many people make when they are just learning to draw E-R diagrams is to confuse data entities with sources/sinks or system outputs and relationships with data flows. A simple rule to avoid such confusion is that a true data entity will have many
possible instances, each with a distinguishing characteristic, as well as one or more other descriptive pieces of data. Consider the entity types that might be associated with a sorority expense system, as represented in Figure 8.4b. In this situation, the sorority treasurer manages accounts and records expense transactions against each account. However, do we need to keep track of data about the treasurer and her supervision of accounts as part of this accounting system? The treasurer is the person entering data about accounts and expenses and making inquiries about account balances and expense transactions by category. Because there is only one treasurer, TREASURER data do not need to be kept. On the other hand, if each account has an account manager (e.g., a sorority officer) who is responsible for assigned accounts, then we may wish to have an ACCOUNT MANAGER entity type with pertinent attributes as well as relationships to other entity types.

In this same situation, is an expense report an entity type? Because an expense report is computed from expense transactions and account balances, it is a data flow, not an entity type. Even though there will be multiple instances of expense reports over time, the report contents are already represented by the ACCOUNT and EXPENSE entity types.

Often when we refer to entity types in subsequent sections, we will simply say entity. This is common among data modelers. We will clarify that we mean an entity by the term entity instance.

**Naming and Defining Entity Types**

Clearly naming and defining data, such as entity types, are important tasks during requirements determination and structuring. When naming and defining entity types, you should use the following guidelines:

- **An entity type name is a singular noun** (such as CUSTOMER, STUDENT, or AUTOMOBILE).
- An entity type name should be descriptive and specific to the organization. For example, a PURCHASE ORDER for orders placed with suppliers is distinct from CUSTOMER ORDER for orders placed by customers. Both of these entity types cannot be named ORDER.
- An entity type name should be concise; for example, in a university database, use REGISTRATION for the event of a student registering for a class rather than STUDENT REGISTRATION FOR CLASS.
- Event entity types should be named for the result of the event, not the activity or process of the event. For example, the event of a project manager assigning an employee to work on a project results in an ASSIGNMENT.

Some specific guidelines for defining entity types follow:

- An entity type definition should include a statement of what the unique characteristic(s) is (are) for each instance of the entity type.
- An entity type definition should make clear what entity instances are included and not included in the entity type. For example, “A customer is a person or organization that has placed an order for a product from us or that we have contacted to advertise or promote our products. A customer does not include persons or organizations that buy our products only through our customers, distributors, or agents.”
- An entity type definition often includes a description of when an instance of the entity type is created and deleted.

**Attributes**

Each entity type has a set of attributes (see the third question in Table 8.1) associated with it. An attribute is a property or characteristic of an entity that is of interest to the organization (relationships may also have attributes, as we will see in the section on relationships). Following are some typical entity types and associated attributes:

- **STUDENT:** Student_ID, Student_Name, Home_Address, Phone_Number, Major
- **AUTOMOBILE:** Vehicle_ID, Color, Weight, Horsepower
- **EMPLOYEE:** Employee_ID, Employee_Name, Payroll_Address, Skill

We use an initial capital letter, followed by lowercase letters, and nouns in naming an attribute, underscores may or may not be used to separate words. In E-R diagrams, we represent an attribute by placing its name inside the rectangle for the associated entity (see Figure 8.5). We use different notations for attributes to distinguish between different types of attributes, which we describe below. Our notation is similar to that used by many CASE and E-R drawing tools, such as Microsoft Visio or Oracle’s Designer. Precisely how different types of attributes are shown varies by tool.

**Naming and Defining Attributes**

Often several attributes have approximately the same name and meaning. Thus, it is important to carefully name attributes using the following guidelines:

- An attribute name is a noun (such as Customer_ID, Age, or Product_Price).
- An attribute name should be unique. No two attributes of the same entity type may have the same name, and it is desirable, for clarity, that no two attributes across all entity types have the same name.
- To make an attribute name unique and for clarity, each attribute name should follow a standard format. For example, your university may establish Student_GPA, as opposed to GPA_of_Student, as an example of the standard format for attribute naming.
- Similar attributes of different entity types should use similar but distinguishing names; for example, the city of residence for faculty and students should be, respectively, Faculty_Residence_City_Name and Student_Residence_City_Name.

Some specific guidelines for defining attributes follow:

- An attribute definition states what the attribute is and possibly why it is important.
- An attribute definition should make it clear what is included and what is not included in the attribute’s value; for example, “Employee_Monthly_Salary_Amount is the amount of money paid each month in the currency of the country of residence of the employee exclusive of any benefits, bonuses, reimbursements, or special payments.”
- Any aliases, or alternative names, for the attribute can be specified in the definition.
- It may also be desirable to state in the definition the source of values for the attribute. Stating the source may make the meaning of the data clearer.
- An attribute definition should indicate if a value for the attribute is required or optional. This business rule about an attribute is important for maintaining data integrity.
Candidate Keys and Identifiers

Every entity type must have an attribute or set of attributes that distinguishes one instance from other instances of the same type (see the second question in Table 8.1). A candidate key is an attribute (or combination of attributes) that uniquely identifies each instance of an entity type. A candidate key for a STUDENT entity type might be Student_ID.

Sometimes a combination of attributes is required to identify a unique entity. For example, consider the entity type GAME for a basketball league. The attribute Team_Name is clearly not a candidate key because each team plays several games. If each team plays exactly one home game against each other team, then the combination of the attributes Home_Team and Visiting_Team is a composite candidate key for GAME.

Some entities may have more than one possible candidate key. One candidate key for EMPLOYEE is Employee_ID; a second is the combination of Employee_Name and Address. However, the second key is not a candidate key for EMPLOYEE because the values of Payroll_Address and Employee_Name could easily change during an employee’s term of employment.

Candidate key
A key or combination of attributes that uniquely identifies each instance of an entity type.

Identifier
A candidate key that has been selected as the unique identifying characteristic for an entity type.

Multivalued attribute
An attribute that may take on more than one value for each entity instance.

Candidate Key

- An attribute definition may indicate if a value for the attribute may change once a value is provided and before the entity instance is deleted. This business rule also controls data integrity.
- An attribute definition may also indicate any relationships that an attribute has with other attributes; for example, "Employee_Vacation_Days_Number is the number of days of paid vacation for the employee. If the employee has a value of 'Exempt' for Employee_Type, then the maximum value for Employee_Vacation_Days_Number is determined by a formula involving the number of years of service for the employee."

attributes are common. The first is to list the multivalued attribute along with other attributes, but use a special symbol to indicate that it is multivalued. This is the approach taken in Figure 8.8a, where the multivalued attribute skill is enclosed in curly brackets.

Sometimes a set of data repeats together. For example, consider Figure 8.8b for an employee entity with multivalued attributes for data about each employee’s dependents. In this situation, data such as dependent name, age, and relation to employee (spouse, child, parent, etc.) are multivalued attributes about an employee, and these attributes repeat together (we show this by using one set of curly brackets around the data that repeats together). Several attributes that repeat together are called a repeating group.

Repeating group
A set of two or more multivalued attributes that are logically related.

Required attribute
An attribute that must have a value for every entity instance.

Optional attribute
An attribute that may not have a value for every entity instance.

Composite attribute
An attribute that has meaningful component parts.

Figure 8.7 shows the representation for a STUDENT entity type using our E-R notation. STUDENT has a simple identifier, Student_ID, and three other simple attributes.

Other Attribute Types

A multivalued attribute may take on more than one value for each entity instance. Suppose that Skill is one of the attributes of EMPLOYEE. If each employee can have more than one skill, Skill is a multivalued attribute. Two ways of showing multivalued
Degree of a Relationship

The degree of a relationship determines how strongly two entities are related. This can be expressed as a percentage, with 100% being a complete relationship and 0% being no relationship at all. The degree of a relationship is influenced by various factors, such as the complexity of the relationship and the amount of data available.

Uncertainty Relationship

An uncertainty relationship is a relationship between two entities that is not certain. This type of relationship is often used when the data is incomplete or when there is uncertainty about the relationship.

Relationship Types and Examples

- One-to-one relationship: An example of a one-to-one relationship is the relationship between an employee and their manager. Each employee has exactly one manager, and each manager supervises exactly one employee.
- One-to-many relationship: An example of a one-to-many relationship is the relationship between a course and its students. A course can have many students, but each student can only be enrolled in one course.
- Many-to-one relationship: An example of a many-to-one relationship is the relationship between a department and its employees. Many employees can work in the same department, but each department only has one manager.
- Many-to-many relationship: An example of a many-to-many relationship is the relationship between a client and their orders. A client can place many orders, and an order can be placed by many clients.

Relationships can be defined as a set of rules that determine how two or more entities are related. These relationships can be represented using diagrams, matrices, or other visual aids. The relationships can also be used to create queries and reports, which can help to analyze the data and make informed decisions.
Binary Relationship
A binary relationship is a relationship between instances of two entity types. This is the most common type of relationship encountered in data modeling. Figure 8-11b shows three examples. The first (one-to-one) indicates that an employee is assigned one parking place, and each parking place is assigned to one employee. The second (one-to-many) indicates that a product line may contain several products, and each product belongs to only one product line. The third (many-to-many) shows that a student may register for more than one course, and that each course may have many student registrants.

Ternary Relationship
A ternary relationship is a simultaneous relationship among instances of three entity types. In the example shown in Figure 8-11c, the relationship Supplies tracks the quantity of a given part that is shipped by a particular vendor to a selected warehouse. Each entity may be a one or a many participant in a ternary relationship (in Figure 8-11, all three entities are many participants).

Cardinalities in Relationships
Suppose there are two entity types, A and B, connected by a relationship. The cardinality of a relationship (see the fifth, seventh, and eighth questions in Table 8.1) is the number of instances of entity B that can (or must) be associated with each instance of entity A. For example, consider the relationship for DVDs at a video store shown in Figure 8-13a.

Clearly, a video store may stock more than one DVD of a given movie. In the terminology we have used so far, this example is intuitively a “many” relationship. Yet it is also true that the store may not have a single copy of a particular movie in stock. We need a more precise notation to indicate the range of cardinalities for a relationship. This notation was introduced in Figure 8-5, which you may want to review at this point.

Cardinality
The number of instances of entity B that can (or must) be associated with each instance of entity A.
Minimum and Maximum Cardinalities. The minimum cardinality of a relationship is the minimum number of instances of entity A that may be associated with each instance of entity B. In the preceding example, the minimum number of DVDs available for a movie is zero, in which case we say that DVD is an optional participant in the Is_stocked_as relationship. When the minimum cardinality of a relationship is one, then we say that entity B is a mandatory participant in the relationship. The maximum cardinality is the maximum number of instances. For our example, this maximum is "many" (an unspecified number greater than one). Using the notation from Figure 8-5, we diagram this relationship in Figure 8-13b. The zero through the line near the DVD entity means a minimum cardinality of zero, whereas the crow's foot notation means a "many" maximum cardinality. The double underline of Copy_Number indicates that this attribute is part of the identifier of DVD, but the full composite identifier must also include the identifier of MOVIE, Movie_Name.

Examples of three relationships that show all possible combinations of minimum and maximum cardinalities appear in Figure 8-14. A brief description of each relationship follows:

1. PATIENT Has_recorded PATIENT_HISTORY (Figure 8-14a). Each patient has recorded one or more patient histories (we assume that the initial patient visit is always recorded as an instance of PATIENT_HISTORY). Each instance of PATIENT_HISTORY is a record for exactly one PATIENT.

2. EMPLOYEE Is_assigned_to PROJECT (Figure 8-14b). Each PROJECT has at least one assigned EMPLOYEE (some projects have more than one). Each EMPLOYEE may or (optionally) may not be assigned to any existing PROJECT, or may be assigned to several PROJECTs.

3. PERSON Is_married_to PERSON (Figure 8-14c). This is an optional zero or one cardinality in both directions because a person may or may not be married.

It is possible for the maximum cardinality to be a fixed number, not an arbitrary "many" value. For example, suppose corporate policy states that an employee may work at most five projects at the same time. We could show this business rule by placing a "5" above or below the crow's foot next to the PROJECT entity in Figure 8-14b.

Figure 8-14: Examples of cardinality constraints
(a) Mandatory cardinality
(b) One optional, one mandatory cardinality
(c) Optional cardinalities

Naming and Defining Relationships
Relationships may be the most difficult component of an E-R diagram to understand. Thus, you should use a few special guidelines for naming relationships, such as the following:

- A relationship name is a verb phrase (such as Assigned_to, Supplies, or Teaches). Relationships represent actions, usually in the present tense. A relationship name states the action taken, not the result of the action (e.g., use Assigned_to, not Assignment).

- You should avoid vague names, such as Has or Is_related_to. Use descriptive verb phrases taken from the action verbs found in the definition of the relationship.

Specific guidelines for defining relationships follow:

- A relationship definition explains what action is being taken and possibly why it is important. It may be important to state who or what the action, but it is not important to explain how the action is taken.

- It may be important to give examples to clarify the action. For example, for a relationship Registered_for between student and course, it may be useful to explain that this covers both on-site and online registration and registrations made during the drop/add period.

- The definition should explain any optional participation. You should explain what conditions lead to zero associated instances, whether this can happen only when an entity instance is first created or whether this can happen at any time.

- A relationship definition should also explain the reason for any explicit maximum cardinality other than many.

- A relationship definition should explain any restrictions on participation in the relationship. For example, "Supervised_by links an employee with the other employees he or she supervises and links an employee with the other employee who supervises him or her. An employee cannot supervise him- or herself, and an employee cannot supervise other employees if his or her job classification level is below 4."

- A relationship definition should explain the extent of history that is kept in the relationship.

- A relationship definition should explain whether an entity instance involved in a relationship instance can transfer participation to another relationship instance. For example, "Places links a customer with the orders they have placed with our company. An order is not transferable to another customer."

Associative Entities
As seen in the examples of the Supplies relationship in Figure 8-11 and the Has_components relationship of Figure 8-12, attributes may be associated with a many-to-many relationship as well as with an entity. For example, suppose that the organization wishes to record the date (month and year) that an employee completes each course. Some sample data follow:

<table>
<thead>
<tr>
<th>Employee_ID</th>
<th>Course_Name</th>
<th>Date_Completed</th>
</tr>
</thead>
<tbody>
<tr>
<td>549-25-1948</td>
<td>Basic Algebra</td>
<td>March 2014</td>
</tr>
<tr>
<td>629-16-4497</td>
<td>Software Quality</td>
<td>June 2014</td>
</tr>
<tr>
<td>816-30-458</td>
<td>Software Quality</td>
<td>February 2014</td>
</tr>
<tr>
<td>549-25-1948</td>
<td>C Programming</td>
<td>May 2014</td>
</tr>
</tbody>
</table>

From these knitted data, you can conclude that the attribute Date_Completed is not a property of the entity EMPLOYEE because a given employee, 549-25-1948, has completed courses on different dates. Nor is Date_Completed a property of COURSE because a particular course (Software Quality) may be completed on different dates. Instead, Date_Completed is a property of the relationship between EMPLOYEE and COURSE. The attribute is associated with the relationship and diagrammed in Figure 8-15.
Because many-to-many and one-to-one relationships may have associated attributes, the E-R data model poses an interesting dilemma: Is a many-to-many relationship actually an entity in disguise? Often the distinction between entity and relationship is simply a matter of how you view the data. An associative entity (sometimes called a group) is a relationship that the data modeler chooses to model as an entity type. Figure 8.15b shows the E-R notation for representing the Completes relationship as an associative entity and Figure 8.15c shows how this would be modeled using Microsoft Visio. The lines from CERTIFICATE to the two entities are not two separate binary relationships, so they do not have labels. Note that EMPLOYEE and COURSE have mandatory one cardinality because an instance of Completes must have an associated EMPLOYEE and COURSE. The labels A and B show where the cardinalities from the Completes relation now appear. We have created an identifier for CERTIFICATE of Certificate_Number, rather than use the implied combination of the identifiers of EMPLOYEE and COURSE, Employee_ID and Course_ID, respectively.

An example of the use of an associative entity for a ternary relationship appears in Figure 8.16. This figure shows an alternative (and more explicit) representation of the ternary Supplies relationship shown in Figure 8.11. In Figure 8.16, the ternary type (associative entity) SHIPMENT SCHEDULE replaces the Supplies relationship from Figure 8.11. Each instance of SHIPMENT SCHEDULE represents a real-world shipment by a given vendor of a particular part to a selected warehouse. The Shipment_Mode and Unit_Cost are attributes of SHIPMENT SCHEDULE. We have not designated an identifier for SHIPMENT SCHEDULE, so implicitly it would be a composite identifier of the identifiers of the three related entities. Business rules about participation of vendors, parts, and warehouses in supplies relationships are shown via the cardinalities next to SUPPLY SCHEDULE. Remember, these are not three separate relationships, as with any associative entity.

One situation in which a relationship must be turned into an associative entity is when the associative entity has other relationships with entities besides the relationship that caused its creation. For example, consider the E-R diagram in Figure 8.17a that represents price quotes from different vendors for purchased parts stocked by PVE. Now, suppose that we also need to know which price quote is in effect for each part shipment received. This additional data requirement necessitates that the Quotes.Price relationship be transformed into an associative entity, as shown in Figure 8.17b.

In this use, PRICE QUOTE is not a ternary relationship. Rather, PRICE QUOTE is a binary many-to-many relationship (associative entity) between VENDOR and PART. In addition, each PART RECEIPT, based on Amount, has an applicable, negotiated Price. Each PART RECEIPT is for a given PART from a specific VENDOR, and the Amount of the receipt dictates the purchase price in effect by matching the Quantity attribute. Because the PRICE QUOTE pertains to a given PART and a given VENDOR, PART RECEIPT does not need direct relationships with these entities.
Summary of Conceptual Data Modeling with E-R Diagrams

The purpose of E-R diagramming is to capture the richest possible understanding of the meaning of data necessary for an information system or organization. Besides the aspects shown in this chapter, there are many other semantics about data that E-R diagramming can represent. Some of these more advanced capabilities are explained in Hoffer et al. (2011). You can also find some general guidelines for effective conceptual data modeling in Moody (1996). The following section presents one final aspect of conceptual data modeling: capturing the relationship between similar entity types.

REPRESENTING SUPERTYPES AND SUBTYPES

Often two or more entity types seem very similar (maybe they have almost the same name), but there are a few differences. That is, these entity types share common properties but also have one or more distinct attributes or relationships. To address this situation, the E-R model has been extended to include supertype/subtype relationships. A supertype is a subgrouping of the entities in an entity type that is meaningful to the organization. For example, STUDENT is an entity type in a university. Two subtypes of STUDENT are GRADUATE STUDENT and UNDERGRADUATE STUDENT. A supertype is a generic entity type that has a relationship with one or more subtypes.

An example illustrating the basic notation used for supertype/subtype relationships appears in Figure 8-18. The supertype PATIENT is connected with a line to a subtype, which in turn is connected by a line to each of the two subtypes, OUTPATIENT and RESIDENT PATIENT. Attributes that are shared by all patients (including the identifier) are associated with the supertype; attributes that are unique to a particular subtype (e.g., Checkback_Date for OUTPATIENT) are associated with that subtype. Relationships in which all types of patients participate (Is_cared_for) are associated with the supertype; relationships in which only one type of patient participates (Is_assigned_for RESIDENT PATIENTs) are associated only with the relevant subtype.

Several important business rules govern supertype/subtype relationships. The total specialization rule specifies that each entity instance of the supertype must be a member of some supertype, it cannot simultaneously be a member of any other supertype. The overlap rule specifies that an entity instance can simultaneously be a member of two (or more) subtypes. Disjoint versus overlap is shown by an "X" or an "O" in the circle.

Figure 8.19 illustrates several combinations of these rules for a hierarchy of supertypes and subtypes in a university database. In this example,

• A PERSON must be (total specialization) an EMPLOYEE, an ALUMNUS, or a STUDENT, or any combination of these subtypes (overlap).
• An EMPLOYEE must be a FACULTY or a STAFF (disjoint), or may be just an EMPLOYEE (partial specialization).
• A STUDENT can only be a GRADUATE STUDENT or an UNDERGRADUATE STUDENT and nothing else (total specialization and disjoint).

BUSINESS RULES

Conceptual data modeling is a step-by-step process for documenting information requirements, and it is concerned with both the structure of data and with rules about the integrity of those data (see the eighth question in Table 8.1). Business rules are specifications that preserve the integrity of the logical data model. Four basic types of business rules are as follows:

1. Entity integrity. Each instance of an entity type must have a unique identifier that is not null.
2. Referential integrity constraints. Rules concerning the relationships between entity types.
4. Triggering operations. Other business rules that protect the validity of attribute values.
The E-R model that we have described in this chapter is concerned primarily with the structure of data rather than with expressing business rules (although some elementary rules are implied in the E-R model). Generally, the business rules are captured during requirements determination and stored in the CASE repository as they are documented. Entity integrity was described earlier in this chapter, and referential integrity is described in Chapter 9 because it applies to database design. In this section, we briefly describe two types of rules: domains and triggering operations. These rules are illustrated with a simple example from a banking environment, shown in Figure 8-20. In this example, an ACCOUNT entity has a relationship (ls_for) with a WITHDRAWAL entity.

**Domains**

A domain is the set of all data types and ranges of values that attributes may assume (Fleming and von Halle, 1990). Domain definitions typically specify some (or all) of the following characteristics of attributes: data type, length, format, range, allowable values, meaning, uniqueness, and null support (whether an attribute value may or may not be null).

Figure 8-20b shows two domain definitions for the banking example. The first definition is for Account_Number. Because Account_Number is an identifier attribute, the definition specifies that Account_Number must be unique and also must not be null (these specifications are true of all identifiers). The definition specifies that the attribute data type is character and that the format is non-nnnn. Thus, any attempt to enter a value for this attribute that does not conform to its character type or format will be rejected, and an error message will be displayed.

The domain definition for the Amount attribute (dollar amount of the requested withdrawal) also may not be null, but is not unique. The format allows for two decimal places to accommodate a currency field. The range of values has a lower limit of zero (to prevent negative values) and an upper limit of 10,000. The latter is an arbitrary upper limit for a single withdrawal transaction.

The use of domains offers several advantages:

- Domains verify that the values for an attribute (stored by insert or update operations) are valid.
- Domains ensure that various data manipulation operations (such as joins or union in a relational database system) are logical.
- Domains help conserve effort in describing attribute characteristics.

Domains can conserve effort because we can define domains and then associate each attribute in the data model with an appropriate domain. To illustrate, suppose that a bank has three types of accounts, with the following identifiers:

- **Account Type**: Identifier
- **CHECKING**: Checking_Account_Number
- **SAVINGS**: Savings_Account_Number
- **LOAN**: Loan_Account_Number

If domains are not used, the characteristics for each of the three identifier attributes must be described separately. Suppose, however, that the characteristics for all three of the attributes are identical. Having defined the domain Account_Number once (as shown in Figure 8-13b), we simply associate each of these three attributes with Account_Number. Other common domains such as Date, Social_Security_Number, and Telephone_Number also need to be defined just once in the model.

**Triggering Operations**

A triggering operation (also called a trigger) is an assertion or rule that governs the validity of data manipulation operations such as insert, update, and delete. The scope of triggering operations may be limited to attributes within one entity or it may extend to attributes in two or more entities. Complex business rules may often be stated as triggering operations.

A triggering operation normally includes the following components:

1. **User rule**: A concise statement of the business rule to be enforced by the triggering operation.
2. **Event**: The data manipulation operation (insert, delete, or update) that initiates the operation.
3. **Entity name**: The name of the entity being accessed and/or modified.
4. **Condition**: The condition that causes the operation to be triggered.
5. **Action**: The action taken when the operation is triggered.

Figure 8-20c shows an example of a triggering operation for the banking situation. The business rule is a simple (and familiar) one: the amount of an attempted withdrawal may not exceed the current account balance. The event of interest is an attempted insert of an instance of the WITHDRAWAL entity type (perhaps from an automated teller machine). The condition is

Amount (of the withdrawal) . ACCOUNT Balance

When this condition is triggered, the action taken is to reject the transaction. You should note two things about this triggering operation: first, it spans two entity types; second, the business rule could not be enforced through the use of domains.

The use of triggering operations is an increasingly important component of database strategy. With triggering operations, the responsibility for data integrity lies within the scope of the database management system rather than with application programs or human operators. In the banking example, tellers could conceivably check the account balance before processing each withdrawal. Human operators would be subject to human error and, in any event, manual processing would not
work with automated teller machines. Alternatively, the logic of integrity checks could be built into the appropriate application programs, but integrity checks would require duplicating the logic in each program. There is no assurance that the logic would be consistent (because the application programs may have been developed at different times by different people) or that the application programs will be kept up to date as conditions change.

As stated earlier, business rules should be documented in the CASE repository. Ideally, these rules will then be checked automatically by database software. Removing business rules from application programs and incorporating them in the repository (in the form of domains, referential integrity constraints, and triggering operations) has several important advantages; specifically, incorporating business rules in the repository:

1. Provides for faster application development with fewer errors because these rules can be generated into programs or enforced by the database management system
2. Reduces maintenance effort and expenditures
3. Provides for faster response to business changes
4. Facilitates end-user involvement in developing new systems and manipulating data
5. Provides for consistent application of integrity constraints
6. Reduces the time and effort required to train application programmers
7. Promotes ease of use of a database

For a more thorough treatment of business rules, see Hoffer et al. (2011).

ROLE OF PACKAGED CONCEPTUAL DATA MODELS—DATABASE PATTERNS

Fortunately, the art and science of data modeling has progressed to the point where it is seldom necessary for an organization to develop its data models from scratch in its entirety. Instead, common database patterns for different business situations are available in packaged data models (or model components) that can be purchased at comparably low cost and, after suitable customization, assembled into full-scale data models. These generic data models are developed by industry specialists, consultants, and database technology vendors based on their expertise and experience in dozens of organizations across multiple industry types. The models are typically provided as the contents of a data modeling software package, such as ERWin from Computer Associates. The software is able to produce E-R diagrams, maintain all metadata about the data model, and produce a variety of reports that help in the process of tailoring the data model to the specific situation, such as customizing data names, changing relationship characteristics, or adding data unique to your environment. The software can then generate the code to define the database to a database management system once the design is fully customized to the local situation. Some simple and limited generic data models can be found in books or on the Internet.

There are two principal types of packaged data models: universal data models applicable to nearly any business or organization and industry-specific data models. We discuss each of these types briefly and provide references for each type.

Universal Data Models

Numerous core subject areas are common to many (or even most) organizations, such as customers, products, accounts, documents, and projects. Although they differ in detail, the underlying data structures are often quite similar for these subjects. Further, there are core business functions such as purchasing, accounting, receiving, and project management that follow common patterns. Universal data models are templates for one or more of these subject areas and/or functions. All of the expected components of data models are generally included: entities, relationships, attributes, primary and foreign keys, and even sample data. Two examples of universal data model sets are provided by Hay (1996) and Silverton (2001a).

Industry-Specific Data Models

Industry-specific data models are generic data models that are designed to be used by organizations within specific industries. Data models are available for nearly every major industry group, including health care, telecommunications, discrete manufacturing, process manufacturing, banking, insurance, and higher education. These models are based on the premise that data model patterns for organizations are very similar within a particular industry ("a bank is a bank"). However, the data models for one industry (such as banking) are quite different from those for another (such as hospitals). Prominent examples of industry-specific data models are provided by Silverton (2001b), Kimball and Ross (2002), and Inmon (1999) at www.billinmon.com.

Benefits of Database Patterns and Packaged Data Models

Most people find the data modeling field refer to a purchased universal or industry-specific database pattern as a logical data model (LDM). Technically, the term logical data model means a conceptual data model with some additional properties associated with the most popular type of database technology—relational databases. The type of data planning and analysis we have covered in this chapter can, in fact, be done using either a conceptual or a logical data model. The process is the same; only the starting point is different.

LDMs are the database version of patterns, components, and prepackaged applications that have been discussed in prior chapters as ways to more quickly and reliably build a new application. An advantage of LDMs is that a packaged data model may exist for almost every industry and application area, for specific operational systems to enterprise systems, such as Enterprise Resource Planning (ERP) and data warehouses. They are available from database software vendors, application software providers, and consulting firms. The use of a prepackaged data model does not eliminate the need for the methods and techniques we discuss in this chapter; they only change the context in which these methods and techniques are used.

It is now important that you consider purchasing a prepackaged data model even when an application is to be built from scratch. Consider the following benefits of starting with and then tailoring a purchased data model:

- **Validated.** Purchased models are proven from extensive experience.
- **Cost reduction.** Projects with purchased models take less time and cost less because the initial discovery steps are no longer necessary, leaving only iterative tailoring and refinement to the local situation.
- **Anticipate the future, not just initial requirements.** Purchased models anticipate future needs not just those recognized during the first version of an application. Thus, their benefits are recurring, not one-time, because the database design does not require structural change, which can have costly ramifications for re-programming the applications using the database.
- **Facilitates system analysis.** The purchased model actually facilitates database planning and analysis by providing a first data model, which you can use to generate specific analysis questions and concrete, not hypothetical or abstract examples of what might be in the appropriate database.
- **Consistent and complete.** The purchased data models are very general, covering almost all options employed by the associated functional area or industry. Thus, they provide a structure that, when tailored, will be consistent and complete.