# Contents

Chapter 2

# Data Models

**After completing this chapter, you will be able to:**

- Discuss data modeling and why data models are important
- Describe the basic data-modeling building blocks
- Define what business rules are and how they influence database design
- Understand how the major data models evolved
- List emerging alternative data models and the needs they fulfill
- Explain how data models can be classified by their level of abstraction

## Preview

This chapter examines data modeling. Data modeling is the first step in the database design journey, serving as a bridge between real-world objects and the computer database.

One of the most vexing problems of database design is that designers, programmers, and end users see data in different ways. Consequently, different views of the same data can lead to database designs that do not reflect an organization's actual operation, thus failing to meet end-user needs and data efficiency requirements. To avoid such failures, database designers must obtain a precise description of the data's nature and many uses within the organization. Communication among database designers, programmers, and end users should be frequent and clear. Data modeling clarifies such communication by reducing the complexities of database design to more easily understood abstractions that define entities, relations, and data transformations.

First, you will learn some basic data-modeling concepts and how current data models have developed from earlier models. Tracing the development of those database models will help you understand the database design and implementation issues that are addressed in the rest of this book. In chronological order, you will be introduced to the hierarchical and network models, the relational model, and the entity relationship (ER) model. You will also learn about the use of the entity relationship diagram (ERD) as a data-modeling tool and the different notations used for ER diagrams. Next, you will be introduced to the object-oriented (OO) model and the object/relational model. Then, you will learn about the emerging NoSQL data model and how it is being used to fulfill the current need to manage very large social media data sets efficiently and effectively. Finally, you will learn how various degrees of data abstraction help reconcile varying views of the same data.

## Data Files and Available Formats

| | MS Access | Oracle | MS SQL | My SQL |
|---|---|---|---|---|
| CH02_InsureCo | ✓ | ✓ | ✓ | ✓ |
| CH02_DealCo | ✓ | ✓ | ✓ | ✓ |
| CH02_TinyCollege | ✓ | ✓ | ✓ | ✓ |

*Data Files Available on cengagebrain.com*

## 2-1 Data Modeling and Data Models

Database design focuses on how the database structure will be used to store and manage end-user data. **Data modeling**, the first step in designing a database, refers to the process of creating a specific data model for a determined problem domain. (A *problem domain* is a clearly defined area within the real-world environment, with a well-defined scope and boundaries that will be systematically addressed.) A **data model** is a relatively simple representation, usually graphical, of more complex real-world data structures. In general terms, a *model* is an abstraction of a more complex real-world object or event. A model's main function is to help you understand the complexities of the real-world environment. Within the database environment, a data model represents data structures and their characteristics, relations, constraints, transformations, and other constructs with the purpose of supporting a specific problem domain.

**data modeling**
The process of creating a specific data model for a determined problem domain.

**data model**
A representation, usually graphic, of a complex "real-world" data structure. Data models are used in the database design phase of the Database Life Cycle.

### Note

The terms *data model* and *database model* are often used interchangeably. In this book, the term *database model* is used to refer to the implementation of a *data model* in a specific database system

Data modeling is an iterative, progressive process. You start with a simple understanding of the problem domain, and as your understanding increases, so does the level of detail of the data model. When done properly, the final data model effectively is a "blueprint" with all the instructions to build a database that will meet all end-user requirements. This blueprint is narrative and graphical in nature, meaning that it contains both text descriptions in plain, unambiguous language and clear, useful diagrams depicting the main data elements.

### Note

An implementation-ready data model should contain at least the following components:

- A description of the data structure that will store the end-user data
- A set of enforceable rules to guarantee the integrity of the data
- A data manipulation methodology to support the real-world data transformations

Traditionally, database designers relied on good judgment to help them develop a good data model. Unfortunately, good judgment is often in the eye of the beholder, and it often develops after much trial and error. For example, if each student in this class has to create a data model for a video store, it is very likely that each will come up with a different model. Which one would be correct? The simple answer is "the one that meets all the end-user requirements," and there may be more than one correct solution! Fortunately, database designers make use of existing data-modeling constructs and powerful database design tools that substantially diminish the potential for errors in database modeling. In the following sections, you will learn how existing data models are used to represent real-world data and how the different degrees of data abstraction facilitate data modeling.

## 2-2 The Importance of Data Models

Data models can facilitate interaction among the designer, the applications programmer, and the end user. A well-developed data model can even foster improved understanding of the organization for which the database design is developed. In short, data models are a communication tool. This important aspect of data modeling was summed up neatly by a client whose reaction was as follows: "I created this business, I worked with this business for years, and this is the first time I've really understood how all the pieces really fit together."

The importance of data modeling cannot be overstated. Data constitutes the most basic information employed by a system. Applications are created to manage data and to help transform data into information, but data is viewed in different ways by different people. For example, contrast the view of a company manager with that of a company clerk. Although both work for the same company, the manager is more likely to have an enterprise-wide view of company data than the clerk.

Even different managers view data differently. For example, a company president is likely to take a universal view of the data because he or she must be able to tie the company's divisions to a common (database) vision. A purchasing manager in the same company is likely to have a more restricted view of the data, as is the company's inventory manager. In effect, each department manager works with a subset of the company's data. The inventory manager is more concerned about inventory levels, while the purchasing manager is more concerned about the cost of items and about relationships with the suppliers of those items.

Applications programmers have yet another view of data, being more concerned with data location, formatting, and specific reporting requirements. Basically, applications programmers translate company policies and procedures from a variety of sources into appropriate interfaces, reports, and query screens.

The different users and producers of data and information often reflect the fable of the blind people and the elephant: the blind person who felt the elephant's trunk had quite a different view from the one who felt the elephant's leg or tail. A view of the whole elephant is needed. Similarly, a house is not a random collection of rooms; to build a house, a person should first have the overall view that is provided by blueprints. Likewise, a sound data environment requires an overall database blueprint based on an appropriate data model.

When a good database blueprint is available, it does not matter that an applications programmer's view of the data is different from that of the manager or the end user. Conversely, when a good database blueprint is not available, problems are likely to ensue. For instance, an inventory management program and an order entry system may use conflicting product-numbering schemes, thereby costing the company thousands or even millions of dollars.

Keep in mind that a house blueprint is an abstraction; you cannot live in the blueprint. Similarly, the data model is an abstraction; you cannot draw the required data out of the data model. Just as you are not likely to build a good house without a blueprint, you are equally unlikely to create a good database without first creating an appropriate data model.

**entity**
A person, place, thing, concept, or event for which data can be stored. See also *attribute*.

## 2-3 Data Model Basic Building Blocks

The basic building blocks of all data models are entities, attributes, relationships, and constraints. An **entity** is a person, place, thing, or event about which data will be collected and stored. An entity represents a particular type of object in the real world, which means an entity is "distinguishable"—that is, each entity occurrence is unique and distinct. For example, a CUSTOMER entity would have many distinguishable customer occurrences, such as John Smith, Pedro Dinamita, and Tom Strickland. Entities may be physical objects, such as customers or products, but entities may also be abstractions, such as flight routes or musical concerts.

An **attribute** is a characteristic of an entity. For example, a CUSTOMER entity would be described by attributes such as customer last name, customer first name, customer phone number, customer address, and customer credit limit. Attributes are the equivalent of fields in file systems.

A **relationship** describes an association among entities. For example, a relationship exists between customers and agents that can be described as follows: an agent can serve many customers, and each customer may be served by one agent. Data models use three types of relationships: one-to-many, many-to-many, and one-to-one. Database designers usually use the shorthand notations 1:M or 1..*, M:N or *..*, and 1:1 or 1..1, respectively. (Although the M:N notation is a standard label for the many-to-many relationship, the label M:M may also be used.) The following examples illustrate the distinctions among the three relationships.

- **One-to-many (1:M or 1..*) relationship.** A painter creates many different paintings, but each is painted by only one painter. Thus, the painter (the "one") is related to the paintings (the "many"). Therefore, database designers label the relationship "PAINTER paints PAINTING" as 1:M. Note that entity names are often capitalized as a convention, so they are easily identified. Similarly, a customer (the "one") may generate many invoices, but each invoice (the "many") is generated by only a single customer. The "CUSTOMER generates INVOICE" relationship would also be labeled 1:M.
- **Many-to-many (M:N or *..*) relationship**. An employee may learn many job skills, and each job skill may be learned by many employees. Database designers label the relationship "EMPLOYEE learns SKILL" as M:N. Similarly, a student can take many classes and each class can be taken by many students, thus yielding the M:N label for the relationship expressed by "STUDENT takes CLASS."
- **One-to-one (1:1 or 1..1) relationship**. A retail company's management structure may require that each of its stores be managed by a single employee. In turn, each store manager, who is an employee, manages only a single store. Therefore, the relationship "EMPLOYEE manages STORE" is labeled 1:1.

The preceding discussion identified each relationship in both directions; that is, relationships are bidirectional:

- *One* CUSTOMER can generate *many* INVOICEs.
- Each of the *many* INVOICEs is generated by only *one* CUSTOMER.

A **constraint** is a restriction placed on the data. Constraints are important because they help to ensure data integrity. Constraints are normally expressed in the form of rules:

- An employee's salary must have values that are between 6,000 and 350,000.
- A student's GPA must be between 0.00 and 4.00.
- Each class must have one and only one teacher.

How do you properly identify entities, attributes, relationships, and constraints? The first step is to clearly identify the business rules for the problem domain you are modeling.

**attribute**
A characteristic of an entity or object. An attribute has a name and a data type.

**relationship**
An association between entities.

**one-to-many (1:M or 1..*) relationship**
Associations among two or more entities that are used by data models. In a 1:M relationship, one entity instance is associated with many instances of the related entity.

**many-to-many (M:N or *..*) relationship**
Association among two or more entities in which one occurrence of an entity is associated with many occurrences of a related entity and one occurrence of the related entity is associated with many occurrences of the first entity.

**one-to-one (1:1 or 1..1) relationship**
Associations among two or more entities that are used by data models. In a 1:1 relationship, one entity instance is associated with only one instance of the related entity.

**constraint**
A restriction placed on data, usually expressed in the form of rules. For example, "A student's GPA must be between 0.00 and 4.00."

## 2-4 Business Rules

When database designers go about selecting or determining the entities, attributes, and relationships that will be used to build a data model, they might start by gaining a thorough understanding of what types of data exist in an organization, how the data is used, and in what time frames it is used. But such data and information do not, by themselves, yield the required understanding of the total business. From a database point of view, the collection of data becomes meaningful only when it reflects properly defined *business rules*. A **business rule** is a brief, precise, and unambiguous description of a policy, procedure, or principle within a specific organization. In a sense, business rules are misnamed: they apply to *any* organization, large or small—a business, a government unit, a religious group, or a research laboratory—that stores and uses data to generate information.

Business rules derived from a detailed description of an organization's operations help to create and enforce actions within that organization's environment. Business rules must be rendered in writing and updated to reflect any change in the organization's operational environment.

Properly written business rules are used to define entities, attributes, relationships, and constraints. Any time you see relationship statements such as "an agent can serve many customers, and each customer can be served by only one agent," business rules are at work. You will see the application of business rules throughout this book, especially in the chapters devoted to data modeling and database design.

To be effective, business rules must be easy to understand and widely disseminated to ensure that every person in the organization shares a common interpretation of the rules. Business rules describe, in simple language, the main and distinguishing characteristics of the data *as viewed by the company*. Examples of business rules are as follows:

- A customer may generate many invoices.
- An invoice is generated by only one customer.
- A training session cannot be scheduled for fewer than 10 employees or for more than 30 employees.

Note that those business rules establish entities, relationships, and constraints. For example, the first two business rules establish two entities (CUSTOMER and INVOICE) and a 1:M relationship between those two entities. The third business rule establishes a constraint (no fewer than 10 people and no more than 30 people) and two entities (EMPLOYEE and TRAINING), and also implies a relationship between EMPLOYEE and TRAINING.

### 2-4a Discovering Business Rules

The main sources of business rules are company managers, policy makers, department managers, and written documentation such as a company's procedures, standards, and operations manuals. A faster and more direct source of business rules is direct interviews with end users. Unfortunately, because perceptions differ, end users are sometimes a less reliable source when it comes to specifying business rules. For example, a maintenance department mechanic might believe that any mechanic can initiate a maintenance procedure, when actually only mechanics with inspection authorization can perform such a task. Such a distinction might seem trivial, but it can have major legal consequences. Although end users are crucial contributors to the development of business rules, *it pays to verify end-user perceptions*. Too often, interviews with several people who perform

**business rule**
A description of a policy, procedure, or principle within an organization. For example, a pilot cannot be on duty for more than 10 hours during a 24-hour period, or a professor may teach up to four classes during a semester.

the same job yield very different perceptions of what the job components are. While such a discovery may point to "management problems," that general diagnosis does not help the database designer. The database designer's job is to reconcile such differences and verify the results of the reconciliation to ensure that the business rules are appropriate and accurate.

The process of identifying and documenting business rules is essential to database design for several reasons:

- It helps to standardize the company's view of data.
- It can be a communication tool between users and designers.
- It allows the designer to understand the nature, role, and scope of the data.
- It allows the designer to understand business processes.
- It allows the designer to develop appropriate relationship participation rules and constraints and to create an accurate data model.

Of course, not all business rules can be modeled. For example, a business rule that specifies "no pilot can fly more than 10 hours within any 24-hour period" cannot be modeled in the database model directly. However, such a business rule can be represented and enforced by application software.

## 2-4b Translating Business Rules into Data Model Components

Business rules set the stage for the proper identification of entities, attributes, relationships, and constraints. In the real world, names are used to identify objects. If the business environment wants to keep track of the objects, there will be specific business rules for the objects. As a general rule, a noun in a business rule will translate into an entity in the model, and a verb (active or passive) that associates the nouns will translate into a relationship among the entities. For example, the business rule "a customer may generate many invoices" contains two nouns (*customer* and *invoices*) and a verb (*generate*) that associates the nouns. From this business rule, you could deduce the following:

- *Customer* and *invoice* are objects of interest for the environment and should be represented by their respective entities.
- There is a *generate* relationship between customer and invoice.

To properly identify the type of relationship, you should consider that relationships are bidirectional; that is, they go both ways. For example, the business rule "a customer may generate many invoices" is complemented by the business rule "an invoice is generated by only one customer." In that case, the relationship is one-to-many (1:M). Customer is the "1" side, and invoice is the "many" side.

To properly identify the relationship type, you should generally ask two questions:

- How many instances of B are related to one instance of A?
- How many instances of A are related to one instance of B?

For example, you can assess the relationship between student and class by asking two questions:

- In how many classes can one student enroll? Answer: many classes.
- How many students can enroll in one class? Answer: many students.

Therefore, the relationship between student and class is many-to-many (M:N). You will have many opportunities to determine the relationships between entities as you proceed through this book, and soon the process will become second nature.

**Online Content**

The hierarchical and network models are largely of historical interest, yet they do contain some elements and features that interest current database professionals. The technical details of those two models are discussed in Appendixes K and L, respectively, which are available at *www.cengagebrain.com*. Appendix G is devoted to the object-oriented (OO) model. However, given the dominant market presence of the relational model, most of the book focuses on the relational model.

### 2-4c Naming Conventions

During the translation of business rules to data model components, you identify entities, attributes, relationships, and constraints. This identification process includes naming the object in a way that makes it unique and distinguishable from other objects in the problem domain. Therefore, it is important to pay special attention to how you name the objects you are discovering.

Entity names should be descriptive of the objects in the business environment and use terminology that is familiar to the users. An attribute name should also be descriptive of the data represented by that attribute. It is also a good practice to prefix the name of an attribute with the name or abbreviation of the entity in which it occurs. For example, in the CUSTOMER entity, the customer's credit limit may be called CUS_CREDIT_LIMIT. The CUS indicates that the attribute is descriptive of the CUSTOMER entity, while CREDIT_LIMIT makes it easy to recognize the data that will be contained in the attribute. This will become increasingly important in later chapters when you learn about the need to use common attributes to specify relationships between entities. The use of a proper naming convention will improve the data model's ability to facilitate communication among the designer, application programmer, and the end user. In fact, a proper naming convention can go a long way toward making your model self-documenting.

## 2-5 The Evolution of Data Models

The quest for better data management has led to several models that attempt to resolve the previous model's critical shortcomings and to provide solutions to ever-evolving data management needs. These models represent schools of thought as to what a database is, what it should do, the types of structures that it should employ, and the technology that would be used to implement these structures. Perhaps confusingly, these models are called data models, as are the graphical data models discussed earlier in this chapter. This section gives an overview of the major data models in roughly chronological order. You will discover that many of the "new" database concepts and structures bear a remarkable resemblance to some of the "old" data model concepts and structures. Table 2.1 traces the evolution of the major data models.

**hierarchical model**
An early database model whose basic concepts and characteristics formed the basis for subsequent database development. This model is based on an upside-down tree structure in which each record is called a segment. The top record is the root segment. Each segment has a 1:M relationship to the segment directly below it.

**segment**
In the hierarchical data model, the equivalent of a file system's record type.

**network model**
An early data model that represented data as a collection of record types in 1:M relationships.

### 2-5a Hierarchical and Network Models

The **hierarchical model** was developed in the 1960s to manage large amounts of data for complex manufacturing projects, such as the Apollo rocket that landed on the moon in 1969. The model's basic logical structure is represented by an upside-down tree. The hierarchical structure contains levels, or segments. A **segment** is the equivalent of a file system's record type. Within the hierarchy, a higher layer is perceived as the parent of the segment directly beneath it, which is called the child. The hierarchical model depicts a set of one-to-many (1:M) relationships between a parent and its children segments. (Each parent can have many children, but each child has only one parent.)

The **network model** was created to represent complex data relationships more effectively than the hierarchical model, to improve database performance, and to impose a database standard. In the network model, the user perceives the network database as a collection of records in 1:M relationships. However, unlike the hierarchical model, the

TABLE 2.1

**EVOLUTION OF MAJOR DATA MODELS**

| GENERATION | TIME | DATA MODEL | EXAMPLES | COMMENTS |
|---|---|---|---|---|
| First | 1960s–1970s | File system | VMS/VSAM | Used mainly on IBM mainframe systems<br>Managed records, not relationships |
| Second | 1970s | Hierarchical and network | IMS, ADABAS, IDS-II | Early database systems<br>Navigational access |
| Third | Mid-1970s | Relational | DB2<br>Oracle<br>MS<br>SQL Server<br>MySQL | Conceptual simplicity<br>Entity relationship (ER) modeling and support for relational data modeling |
| Fourth | Mid-1980s | Object-oriented<br>Object/relational (O/R) | Versant<br>Objectivity/DB<br>DB2 UDB<br>Oracle 12c | Object/relational supports object data types<br>Star Schema support for data warehousing<br>Web databases become common |
| Fifth | Mid-1990s | XML Hybrid DBMS | dbXML<br>Tamino<br>DB2 UDB<br>Oracle 12c<br>MS SQL Server | Unstructured data support<br>O/R model supports XML documents<br>Hybrid DBMS adds object front end to relational databases<br>Support large databases (terabyte size) |
| Emerging Models: NoSQL | Early 2000s to present | Key-value store<br>Column store | SimpleDB (Amazon)<br>BigTable (Google)<br>Cassandra (Apache)<br>MongoDB<br>Riak | Distributed, highly scalable<br>High performance, fault tolerant<br>Very large storage (petabytes)<br>Suited for sparse data<br>Proprietary application programming interface (API) |

network model allows a record to have more than one parent. While the network database model is generally not used today, the definitions of standard database *concepts* that emerged with the network model are still used by modern data models:

- The **schema** is the conceptual organization of the entire database as viewed by the database administrator.
- The **subschema** defines the portion of the database "seen" by the application programs that actually produce the desired information from the data within the database.
- A **data manipulation language (DML)** defines the environment in which data can be managed and is used to work with the data in the database.
- A schema **data definition language (DDL)** enables the database administrator to define the schema components.

As information needs grew and more sophisticated databases and applications were required, the network model became too cumbersome. The lack of ad hoc query capability put heavy pressure on programmers to generate the code required to produce even the simplest reports. Although the existing databases provided limited data independence, any structural change in the database could still produce havoc in all application programs that drew data from the database. Because of the disadvantages of the hierarchical and network models, they were largely replaced by the relational data model in the 1980s.

**schema**
A logical grouping of database objects, such as tables, indexes, views, and queries, that are related to each other.

**subschema**
The portion of the database that interacts with application programs.

**data manipulation language (DML)**
The set of commands that allows an end user to manipulate the data in the database, such as SELECT, INSERT, UPDATE, DELETE, COMMIT, and ROLLBACK.

**data definition language (DDL)**
The language that allows a database administrator to define the database structure, schema, and subschema.

## 2-5b The Relational Model

The **relational model** was introduced in 1970 by E. F. Codd of IBM in his landmark paper "A Relational Model of Data for Large Shared Databanks" (*Communications of the ACM*, June 1970, pp. 377–387). The relational model represented a major breakthrough for both users and designers. To use an analogy, the relational model produced an "automatic transmission" database to replace the "standard transmission" databases that preceded it. Its conceptual simplicity set the stage for a genuine database revolution.

### Note

The relational database model presented in this chapter is an introduction and an overview. A more detailed discussion is in Chapter 3, The Relational Database Model. In fact, the relational model is so important that it will serve as the basis for discussions in most of the remaining chapters.

**relational model**
Developed by E. F. Codd of IBM in 1970, the relational model is based on mathematical set theory and represents data as independent relations. Each relation (table) is conceptually represented as a two-dimensional structure of intersecting rows and columns. The relations are related to each other through the sharing of common entity characteristics (values in columns).

**table (relation)**
A logical construct perceived to be a two-dimensional structure composed of intersecting rows (entities) and columns (attributes) that represents an entity set in the relational model.

**tuple**
In the relational model, a table row.

**relational database management system (RDBMS)**
A collection of programs that manages a relational database. The RDBMS software translates a user's logical requests (queries) into commands that physically locate and retrieve the requested data.

The relational model's foundation is a mathematical concept known as a relation. To avoid the complexity of abstract mathematical theory, you can think of a **relation** (sometimes called a **table**) as a two-dimensional structure composed of intersecting rows and columns. Each row in a relation is called a **tuple**. Each column represents an attribute. The relational model also describes a precise set of data manipulation constructs based on advanced mathematical concepts.

In 1970, Codd's work was considered ingenious but impractical. The relational model's conceptual simplicity was bought at the expense of computer overhead; computers at that time lacked the power to implement the relational model. Fortunately, computer power grew exponentially, as did operating system efficiency. Better yet, the cost of computers diminished rapidly as their power grew. Today, even PCs, which cost a fraction of what their mainframe ancestors cost, can run sophisticated relational database software such as Oracle, DB2, Microsoft SQL Server, MySQL, and other mainframe relational software.

The relational data model is implemented through a very sophisticated **relational database management system (RDBMS)**. The RDBMS performs the same basic functions provided by the hierarchical and network DBMS systems, in addition to a host of other functions that make the relational data model easier to understand and implement (as outlined in Chapter 1, in the DBMS Functions section).

Arguably the most important advantage of the RDBMS is its ability to hide the complexities of the relational model from the user. The RDBMS manages all of the physical details, while the user sees the relational database as a collection of tables in which data is stored. The user can manipulate and query the data in a way that seems intuitive and logical.

Tables are related to each other through the sharing of a common attribute (a value in a column). For example, the CUSTOMER table in Figure 2.1 might contain a sales agent's number that is also contained in the AGENT table.

The common link between the CUSTOMER and AGENT tables enables you to match the customer to his or her sales agent, even though the customer data is stored in one table and the sales representative data is stored in another table. For example, you can easily determine that customer Dunne's agent is Alex Alby because for customer Dunne, the CUSTOMER table's AGENT_CODE is 501, which matches the AGENT table's AGENT_CODE for Alex Alby. Although the tables are independent of one another, you

## FIGURE 2.1 LINKING RELATIONAL TABLES

**Table name: AGENT (first six attributes)**

**Database name: Ch02_InsureCo**

| AGENT_CODE | AGENT_LNAME | AGENT_FNAME | AGENT_INITIAL | AGENT_AREACODE | AGENT_PHONE |
|---|---|---|---|---|---|
| 501 | Alby | Alex | B | 713 | 228-1249 |
| 502 | Hahn | Leah | F | 615 | 882-1244 |
| 503 | Okon | John | T | 615 | 123-5589 |

**Link through AGENT_CODE**

**Table name: CUSTOMER**

| CUS_CODE | CUS_LNAME | CUS_FNAME | CUS_INITIAL | CUS_AREACODE | CUS_PHONE | CUS_INSURE_TYPE | CUS_INSURE_AMT | CUS_RENEW_DATE | AGENT_CODE |
|---|---|---|---|---|---|---|---|---|---|
| 10010 | Ramas | Alfred | A | 615 | 844-2573 | T1 | 100.00 | 05-Apr-2018 | 502 |
| 10011 | Dunne | Leona | K | 713 | 894-1238 | T1 | 250.00 | 16-Jun-2018 | 501 |
| 10012 | Smith | Kathy | W | 615 | 894-2285 | S2 | 150.00 | 29-Jan-2019 | 502 |
| 10013 | Olowski | Paul | F | 615 | 894-2180 | S1 | 300.00 | 14-Oct-2018 | 502 |
| 10014 | Orlando | Myron | | 615 | 222-1672 | T1 | 100.00 | 28-Dec-2019 | 501 |
| 10015 | O'Brian | Amy | B | 713 | 442-3381 | T2 | 850.00 | 22-Sep-2018 | 503 |
| 10016 | Brown | James | G | 615 | 297-1228 | S1 | 120.00 | 25-Mar-2019 | 502 |
| 10017 | Williams | George | | 615 | 290-2556 | S1 | 250.00 | 17-Jul-2018 | 503 |
| 10018 | Farriss | Anne | G | 713 | 382-7185 | T2 | 100.00 | 03-Dec-2018 | 501 |
| 10019 | Smith | Olette | K | 615 | 297-3809 | S2 | 500.00 | 14-Mar-2019 | 503 |

can easily associate the data between tables. The relational model provides a minimum level of controlled redundancy to eliminate most of the redundancies commonly found in file systems.

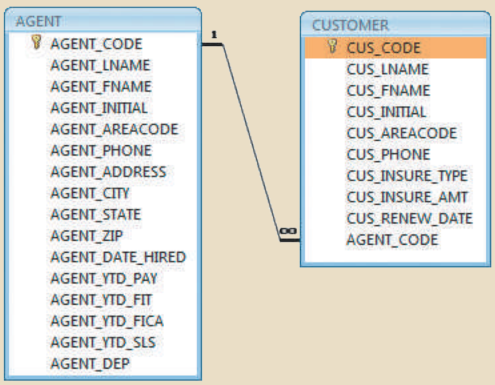
The relationship type (1:1, 1:M, or M:N) is often shown in a relational schema, an example of which is shown in Figure 2.2. A **relational diagram** is a representation of the relational database's entities, the attributes within those entities, and the relationships between those entities.

In Figure 2.2, the relational diagram shows the connecting fields (in this case, AGENT_CODE) and the relationship type (1:M). Microsoft Access, the database software application used to generate Figure 2.2, employs the infinity symbol (∞) to indicate the "many" side. In this example, the CUSTOMER represents the "many" side because an AGENT can have many CUSTOMERs. The AGENT represents the "1" side because each CUSTOMER has only one AGENT.

**Online Content**

This chapter's databases are available at *www.cengagebrain.com*. For example, the contents of the AGENT and CUSTOMER tables shown in Figure 2.1 are in the database named Ch02_InsureCo.

## FIGURE 2.2 A RELATIONAL DIAGRAM




**relational diagram**
A graphical representation of a relational database's entities, the attributes within those entities, and the relationships among the entities.

A relational table stores a collection of related entities. In this respect, the relational database table resembles a file, but there is a crucial difference between a table and a file: a table yields complete data and structural independence because it is a purely logical structure. How the data is physically stored in the database is of no concern to the user or the designer; the perception is what counts. This property of the relational data model, which is explored in depth in the next chapter, became the source of a real database revolution.

Another reason for the relational data model's rise to dominance is its powerful and flexible query language. Most relational database software uses Structured Query Language (SQL), which allows the user to specify what must be done without specifying how. The RDBMS uses SQL to translate user queries into instructions for retrieving the requested data. SQL makes it possible to retrieve data with far less effort than any other database or file environment.

From an end-user perspective, any SQL-based relational database application involves three parts: a user interface, a set of tables stored in the database, and the SQL "engine." Each of these parts is explained as follows:

- *The end-user interface*. Basically, the interface allows the end user to interact with the data (by automatically generating SQL code). Each interface is a product of the software vendor's idea of meaningful interaction with the data. You can also design your own customized interface with the help of application generators that are now standard fare in the database software arena.
- *A collection of tables stored in the database*. In a relational database, all data is perceived to be stored in tables. The tables simply "present" the data to the end user in a way that is easy to understand. Each table is independent. Rows in different tables are related by common values in common attributes.
- *SQL engine*. Largely hidden from the end user, the SQL engine executes all queries, or data requests. Keep in mind that the SQL engine is part of the DBMS software. The end user uses SQL to create table structures and to perform data access and table maintenance. The SQL engine processes all user requests—largely behind the scenes and without the end user's knowledge. Hence, SQL is said to be a declarative language that tells what must be done but not how. (You will learn more about the SQL engine in Chapter 11, Database Performance Tuning and Query Optimization.)

Because the RDBMS performs some tasks behind the scenes, it is not necessary to focus on the physical aspects of the database. Instead, the following chapters concentrate on the logical portion of the relational database and its design. Furthermore, SQL is covered in detail in Chapter 7, Introduction to Structured Query Language (SQL), and in Chapter 8, Advanced SQL.

## 2-5c The Entity Relationship Model

The conceptual simplicity of relational database technology triggered the demand for RDBMSs. In turn, the rapidly increasing requirements for transaction and information created the need for more complex database implementation structures, thus creating the need for more effective database design tools. (Building a skyscraper requires more detailed design activities than building a doghouse, for example.)

Complex design activities require conceptual simplicity to yield successful results. Although the relational model was a vast improvement over the hierarchical and network models, it still lacked the features that would make it an effective database *design* tool. Because it is easier to examine structures graphically than to describe them in text, database designers prefer to use a graphical tool in which entities and their relationships are pictured. Thus, the **entity relationship (ER) model,** or **ERM**, has become a widely accepted standard for data modeling.

Peter Chen first introduced the ER data model in 1976; the graphical representation of entities and their relationships in a database structure quickly became popular because it *complemented* the relational data model concepts. The relational data model and ERM combined to provide the foundation for tightly structured database design. ER models are normally represented in an **entity relationship diagram (ERD)**, which uses graphical representations to model database components. You will learn how to use ERDs to design databases in Chapter 4, Entity Relationship (ER) Modeling.

The ER model is based on the following components:

- *Entity*. Earlier in this chapter, an entity was defined as anything about which data will be collected and stored. An entity is represented in the ERD by a rectangle, also known as an entity box. The name of the entity, a noun, is written in the center of the rectangle. The entity name is generally written in capital letters and in singular form: PAINTER rather than PAINTERS, and EMPLOYEE rather than EMPLOYEES. Usually, when applying the ERD to the relational model, an entity is mapped to a relational table. Each row in the relational table is known as an **entity instance** or **entity occurrence** in the ER model. A collection of like entities is known as an **entity set**. For example, you can think of the AGENT file in Figure 2.1 as a collection of three agents (*entities*) in the AGENT *entity* set. Technically speaking, the ERD depicts entity sets. Unfortunately, ERD designers use the word *entity* as a substitute for *entity set*, and this book will conform to that established practice when discussing any ERD and its components.
- Each entity consists of a set of *attributes* that describes particular characteristics of the entity. For example, the entity EMPLOYEE will have attributes such as a Social Security number, a last name, and a first name. (Chapter 4 explains how attributes are included in the ERD.)
- *Relationships*. Relationships describe associations among data. Most relationships describe associations between two entities. When the basic data model components were introduced, three types of data relationships were illustrated: one-to-many (1:M), many-to-many (M:N), and one-to-one (1:1). The ER model uses the term **connectivity** to label the relationship types. The name of the relationship is usually an active or passive verb. For example, a PAINTER *paints* many PAINTINGs, an EMPLOYEE *learns* many SKILLs, and an EMPLOYEE *manages* a STORE.

Figure 2.3 shows the different types of relationships using three ER notations: the original **Chen notation**, the **Crow's Foot notation**, and the newer **class diagram notation**, which is part of the Unified Modeling Language (UML).

The left side of the ER diagram shows the Chen notation, based on Peter Chen's landmark paper. In this notation, the connectivities are written next to each entity box. Relationships are represented by a diamond connected to the related entities through a relationship line. The relationship name is written inside the diamond.

The middle of Figure 2.3 illustrates the Crow's Foot notation. The name *Crow's Foot* is derived from the three-pronged symbol used to represent the "many" side of the relationship. As you examine the basic Crow's Foot ERD in Figure 2.3, note that the connectivities are represented by symbols. For example, the "1" is represented by a short line segment, and the "M" is represented by the three-pronged "crow's foot." In this example, the relationship name is written above the relationship line.

The right side of Figure 2.3 shows the UML notation (also known as the UML class notation). Note that the connectivities are represented by lines with symbols (1..1, 1..*).

**entity relationship (ER) model (ERM)**
A data model that describes relationships (1:1, 1:M, and M:N) among entities at the conceptual level with the help of ER diagrams.

**entity relationship diagram (ERD)**
A diagram that depicts an entity relationship model's entities, attributes, and relations.

**entity instance (entity occurrence)**
A row in a relational table.

**entity set**
A collection of like entities.

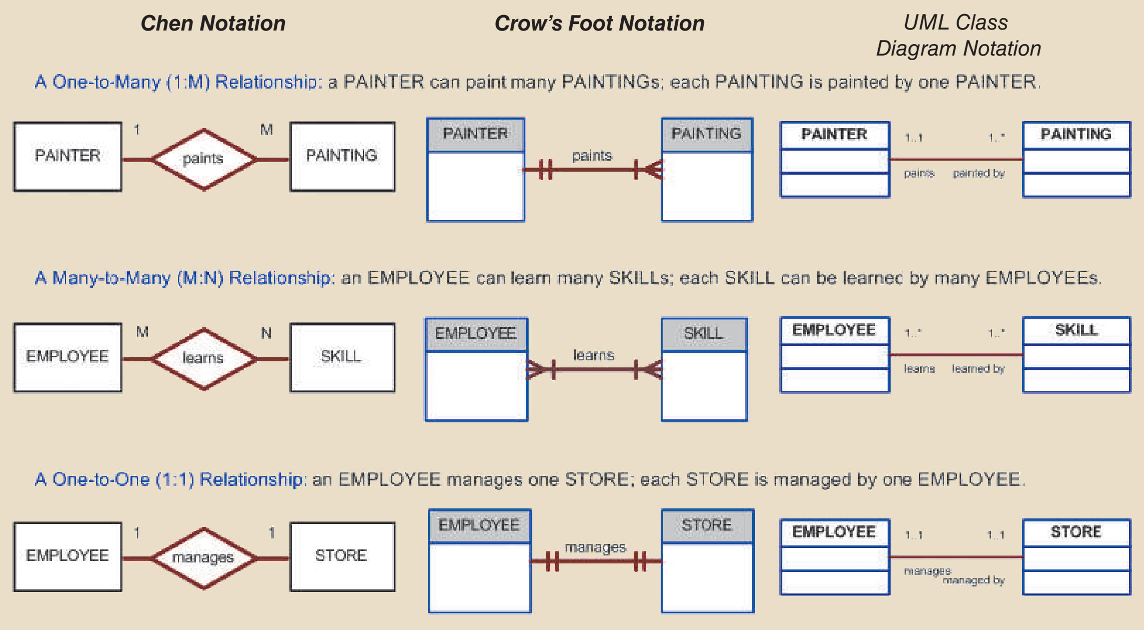
**connectivity**
The type of relationship between entities. Classifications include 1:1, 1:M, and M:N.

**Chen notation**
See *entity relationship (ER) model*.

**Crow's Foot notation**
A representation of the entity relationship diagram that uses a three-pronged symbol to represent the "many" sides of the relationship.

**class diagram notation**
The set of symbols used in the creation of class diagrams.

FIGURE 2.3 THE ER MODEL NOTATIONS




Also, the UML notation uses names in both sides of the relationship. For example, to read the relationship between PAINTER and PAINTING, note the following:

- A PAINTER "paints" one to many PAINTINGs, as indicated by the 1..* symbol.
- A PAINTING is "painted by" one and only one PAINTER, as indicated by the 1..1 symbol.

## Note

Many-to-many (M:N) relationships exist at a conceptual level, and you should know how to recognize them. However, you will learn in Chapter 3 that M:N relationships are not appropriate in a relational model. For that reason, Microsoft Visio does not support the M:N relationship directly. Therefore, to illustrate the existence of an M:N relationship using Visio, you have to change the line style of the connector (see Appendix A, Designing Databases with Visio Professional: A Tutorial, at *www.cengagebrain.com*).

In Figure 2.3, entities and relationships are shown in a horizontal format, but they may also be oriented vertically. The entity location and the order in which the entities are presented are immaterial; just remember to read a 1:M relationship from the "1" side to the "M" side.

The Crow's Foot notation is used as the design standard in this book. However, the Chen notation is used to illustrate some of the ER modeling concepts whenever necessary. Most data modeling tools let you select the Crow's Foot or UML class diagram notation. Microsoft Visio Professional software was used to generate the Crow's Foot designs you will see in subsequent chapters.

The ER model's exceptional visual simplicity makes it the dominant database modeling and design tool. Nevertheless, the search for better data-modeling tools continues as the data environment continues to evolve.

## 2-5d The Object-Oriented Model

Increasingly complex real-world problems demonstrated a need for a data model that more closely represented the real world. In the **object-oriented data model (OODM)**, both data *and its relationships* are contained in a single structure known as an **object**. In turn, the OODM is the basis for the **object-oriented database management system (OODBMS)**.

An OODM reflects a very different way to define and use entities. Like the relational model's entity, an object is described by its factual content. But, quite *unlike* an entity, an object includes information about relationships between the facts within the object, as well as information about its relationships with other objects. Therefore, the facts within the object are given greater *meaning*. The OODM is said to be a **semantic data model** because *semantic* indicates meaning.

Subsequent OODM development has allowed an object also to contain all *operations* that can be performed on it, such as changing its data values, finding a specific data value, and printing data values. Because objects include data, various types of relationships, and operational procedures, the object becomes self-contained, thus making it—at least potentially—a basic building block for autonomous structures.

The OO data model is based on the following components:

- An object is an abstraction of a real-world entity. In general terms, an object may be considered equivalent to an ER model's entity. More precisely, an object represents only one occurrence of an entity. (The object's semantic content is defined through several of the items in this list.)
- Attributes describe the properties of an object. For example, a PERSON object includes the attributes Name, Social Security Number, and Date of Birth.
- Objects that share similar characteristics are grouped in classes. A **class** is a collection of similar objects with shared structure (attributes) and behavior (methods). In a general sense, a class resembles the ER model's *entity* set. However, a class is different from an entity set in that it contains a set of procedures known as *methods*. A class's **method** represents a real-world action such as *finding* a selected PERSON's name, *changing* a PERSON's name, or *printing* a PERSON's address. In other words, methods are the equivalent of *procedures* in traditional programming languages. In OO terms, methods define an object's *behavior*.
- Classes are organized in a *class hierarchy*. The **class hierarchy** resembles an upside-down tree in which each class has only one parent. For example, the CUSTOMER class and the EMPLOYEE class share a parent PERSON class. (Note the similarity to the hierarchical data model in this respect.)
- **Inheritance** is the ability of an object within the class hierarchy to inherit the attributes and methods of the classes above it. For example, two classes, CUSTOMER and EMPLOYEE, can be created as subclasses from the class PERSON. In this case, CUSTOMER and EMPLOYEE will inherit all attributes and methods from PERSON.
- Object-oriented data models are typically depicted using **Unified Modeling Language (UML)** class diagrams. UML is a language based on OO concepts that describes a set of diagrams and symbols you can use to graphically model a system. UML **class diagrams** are used to represent data and its relationships within the larger UML object-oriented system's modeling language. For a more complete description of UML, see Appendix H, Unified Modeling Language (UML).

### Online Content

This chapter introduces only basic OO concepts. You can examine object-orientation concepts and principles in detail in Appendix G, Object-Oriented Databases, at *www.cengagebrain.com.*

**object-oriented data model (OODM)**
A data model whose basic modeling structure is an object.

**object**
An abstract representation of a real-world entity that has a unique identity, embedded properties, and the ability to interact with other objects and itself.

**object-oriented database management system (OODBMS)**
Data management software used to manage data in an object-oriented database model.

**semantic data model**
The first of a series of data models that models both data and their relationships in a single structure known as an object.

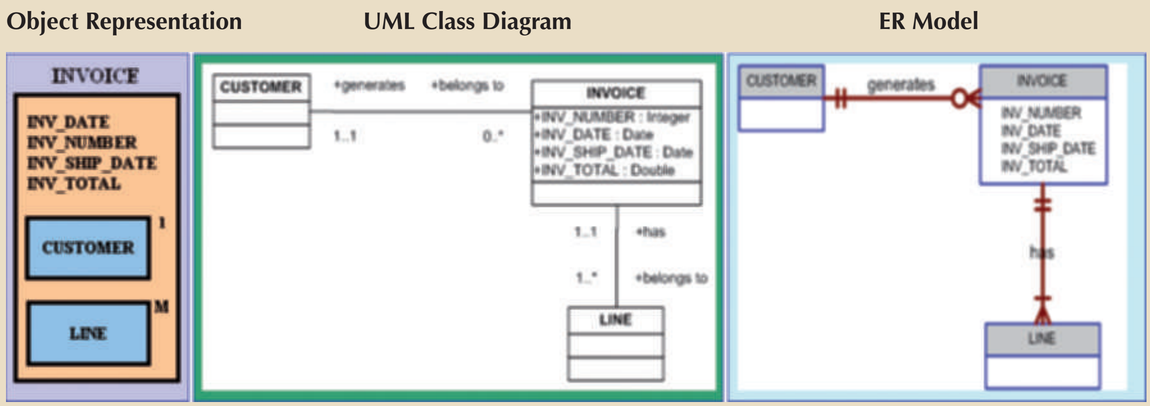
**class**
A collection of similar objects with shared structure (attributes) and behavior (methods). A class encapsulates an object's data representation and a method's implementation.

**method**
In the object-oriented data model, a named set of instructions to perform an action. Methods represent real-world actions.

**class hierarchy**
The organization of classes in a hierarchical tree in which each parent class is a *superclass* and each child class is a *subclass*. See also *inheritance*.

To illustrate the main concepts of the OODM, consider a simple invoicing problem. In this case, invoices are generated by customers, each invoice references one or more lines, and each line represents an item purchased by a customer. Figure 2.4 illustrates the object representation for this simple invoicing problem, as well as the equivalent UML class diagram and ER model. The object representation is a simple way to visualize a single object occurrence.

FIGURE 2.4 A COMPARISON OF THE OO, UML, AND ER MODELS




As you examine Figure 2.4, note the following:

- The object representation of the INVOICE includes all related objects within the *same* object box. Note that the connectivities (1 and M) indicate the relationship of the related objects to the INVOICE. For example, the "1" next to the CUSTOMER object indicates that each INVOICE is related to only one CUSTOMER. The "M" next to the LINE object indicates that each INVOICE contains many LINEs.
- The UML class diagram uses three separate object classes (CUSTOMER, INVOICE, and LINE) and two relationships to represent this simple invoicing problem. Note that the relationship connectivities are represented by the 1..1, 0..*, and 1..* symbols, and that the relationships are named in both ends to represent the different "roles" that the objects play in the relationship.
- The ER model also uses three separate entities and two relationships to represent this simple invoice problem.

**inheritance**
In the object-oriented data model, the ability of an object to inherit the data structure and methods of the classes above it in the class hierarchy. See also *class hierarchy*.

**Unified Modeling Language (UML)**
A language based on object-oriented concepts that provides tools such as diagrams and symbols to graphically model a system.

**class diagram**
A diagram used to represent data and their relationships in UML object notation.

The OODM advances influenced many areas, from system modeling to programming. (Most contemporary programming languages have adopted OO concepts, including Java, Ruby, Perl, C#, and Visual Studio .NET.) The added semantics of the OODM allowed for a richer representation of complex objects. This in turn enabled applications to support increasingly complex objects in innovative ways. As you will see in the next section, such evolutionary advances also affected the relational model.

## 2-5e Object/Relational and XML

Facing the demand to support more complex data representations, the relational model's main vendors evolved the model further and created the **extended relational data**

**model (ERDM)**. The ERDM adds many of the OO model's features within the inherently simpler relational database structure. The ERDM gave birth to a new generation of relational databases that support OO features such as objects (encapsulated data and methods), extensible data types based on classes, and inheritance. That's why a DBMS based on the ERDM is often described as an **object/relational database management system (O/R DBMS)**.

Today, most relational database products can be classified as object/relational, and they represent the dominant market share of OLTP and OLAP database applications. The success of the O/R DBMSs can be attributed to the model's conceptual simplicity, data integrity, easy-to-use query language, high transaction performance, high availability, security, scalability, and expandability. In contrast, the OO DBMS is popular in niche markets such as computer-aided drawing/computer-aided manufacturing (CAD/CAM), geographic information systems (GIS), telecommunications, and multimedia, which require support for more complex objects.

From the start, the OO and relational data models were developed in response to different problems. The OO data model was created to address very specific engineering needs, not the wide-ranging needs of general data management tasks. The relational model was created with a focus on better data management based on a sound mathematical foundation. Given its focus on a smaller set of problem areas, it is not surprising that the OO market has not grown as rapidly as the relational data model market.

The use of complex objects received a boost with the Internet revolution. When organizations integrated their business models with the Internet, they realized its potential to access, distribute, and exchange critical business information. This resulted in the widespread adoption of the Internet as a business communication tool. Within this environment, **Extensible Markup Language (XML)** emerged as the de facto standard for the efficient and effective exchange of structured, semistructured, and unstructured data. Organizations that used XML data soon realized that they needed to manage large amounts of unstructured data such as word-processing documents, webpages, emails, and diagrams. To address this need, XML databases emerged to manage unstructured data within a native XML format (see Chapter 15, Database Connectivity and Web Technologies, for more information about XML). At the same time, O/R DBMSs added support for XML-based documents within their relational data structure. Due to its robust foundation in broadly applicable principles, the relational model is easily extended to include new classes of capabilities, such as objects and XML.

Although relational and object/relational databases address most current data processing needs, a new generation of databases has emerged to address some very specific challenges found in some Internet-era organizations.

## 2-5f Emerging Data Models: Big Data and NoSQL

Deriving usable business information from the mountains of web data that organizations have accumulated over the years has become an imperative need. Web data in the form of browsing patterns, purchasing histories, customer preferences, behavior patterns, and social media data from sources such as Facebook, Twitter, and LinkedIn have inundated organizations with combinations of structured and unstructured data. In addition, mobile technologies such as smartphones and tablets, plus sensors of all types—GPS, RFID systems, weather sensors, biomedical devices, space research probes, car and aviation black boxes—as well as other Internet and cellular-connected devices, have created new ways to automatically collect massive amounts of data in multiple formats (text, pictures, sound, video, etc.). The amount of data being collected grows exponentially every day. According to IBM, "Every day we create 2.5 quintillion bytes of data—so much that 90 percent of the

**extended relational data model (ERDM)**
A model that includes the object-oriented model's best features in an inherently simpler relational database structural environment. See *extended entity relationship model (EERM)*.

**object/relational database management system (O/R DBMS)**
A DBMS based on the extended relational model (ERDM). The ERDM, championed by many relational database researchers, constitutes the relational model's response to the OODM. This model includes many of the object-oriented model's best features within an inherently simpler relational database structure.

**Extensible Markup Language (XML)**
A metalanguage used to represent and manipulate data elements. Unlike other markup languages, XML permits the manipulation of a document's data elements. XML facilitates the exchange of structured documents such as orders and invoices over the Internet.

data in the world today has been created in the last two years alone."[1] According to some studies, the rapid pace of data growth is the top challenge for organizations,[2] with system performance and scalability as the next biggest challenges. Today's information technology (IT) managers are constantly balancing the need to manage this rapidly growing data with shrinking budgets. The need to manage and leverage all these converging trends (rapid data growth, performance, scalability, and lower costs) has triggered a phenomenon called "Big Data." **Big Data** refers to a movement to find new and better ways to manage large amounts of web- and sensor-generated data and derive business insight from it, while simultaneously providing high performance and scalability at a reasonable cost.

The term *Big Data* has been used in many different frameworks, from law to statistics to economics to computing. The term seems to have been first used in a computing framework by John Mashey, a Silicon Graphics scientist in the 1990s.[3] However, it seems to be Douglas Laney, a data analyst from the Gartner Group, who first described the basic characteristics of Big Data databases[4]: volume, velocity, and variety, or the **3 Vs**.

- *Volume* refers to the amounts of data being stored. With the adoption and growth of the Internet and social media, companies have multiplied the ways to reach customers. Over the years, and with the benefit of technological advances, data for millions of e-transactions were being stored daily on company databases. Furthermore, organizations are using multiple technologies to interact with end users and those technologies are generating mountains of data. This ever-growing volume of data quickly reached petabytes in size, and it's still growing.
- *Velocity* refers not only to the speed with which data grows but also to the need to process this data quickly in order to generate information and insight. With the advent of the Internet and social media, business response times have shrunk considerably. Organizations need not only to store large volumes of quickly accumulating data but also need to process such data quickly. The velocity of data growth is also due to the increase in the number of different data streams from which data is being piped to the organization (via the web, e-commerce, Tweets, Facebook posts, emails, sensors, GPS, and so on).
- *Variety* refers to the fact that the data being collected comes in multiple different data formats. A great portion of these data comes in formats not suitable to be handled by the typical operational databases based on the relational model.

The 3 Vs framework illustrates what companies now know, that the amount of data being collected in their databases has been growing exponentially in size and complexity. Traditional relational databases are good at managing structured data but are not well suited to managing and processing the amounts and types of data being collected in today's business environment.

The problem is that the relational approach does not always match the needs of organizations with Big Data challenges.

- It is not always possible to fit unstructured, social media and sensor-generated data into the conventional relational structure of rows and columns.
- Adding millions of rows of multiformat (structured and nonstructured) data on a daily basis will inevitably lead to the need for more storage, processing power, and

**Big Data**
A movement to find new and better ways to manage large amounts of web-generated data and derive business insight from it, while simultaneously providing high performance and scalability at a reasonable cost.

**3 Vs**
Three basic characteristics of Big Data databases: volume, velocity, and variety.

[1] IBM, "What is big data? Bringing big data to the enterprise," http://www-01.ibm.com/software/data/bigdata/, accessed April 2013.
[2] "Gartner survey shows data growth as the largest data center infrastructure challenge," www.gartner.com/it/page.jsp?id=1460213, accessed March 2015.
[3] Steve Lohr, "The origins of 'Big Data': An etymological detective story," *New York Times*, February 1, 2013.
[4] Douglas Laney, "3D data management controlling data volume, velocity and variety," META Group, February 6, 2011.

sophisticated data analysis tools that may not be available in the relational environment. The type of high-volume implementations required in the RDBMS environment for the Big Data problem comes with a hefty price tag for expanding hardware, storage, and software licenses.

- Data analysis based on OLAP tools has proven to be very successful in relational environments with highly structured data. However, mining for usable data in the vast amounts of unstructured data collected from web sources requires a different approach.

There is no "one-size-fits-all" cure to data management needs (although many established database vendors will probably try to sell you on the idea). For some organizations, creating a highly scalable, fault-tolerant infrastructure for Big Data analysis could prove to be a matter of business survival. The business world has many examples of companies that leverage technology to gain a competitive advantage, and others that miss it. Just ask yourself how the business landscape would be different if:

- Blackberry had responded quickly to the emerging Apple smartphone technology.
- MySpace had responded to Facebook's challenge in time.
- Blockbuster had reacted to the Netflix business model sooner.
- Barnes & Noble had developed a viable Internet strategy before Amazon.

Will broadcast television networks be successful in adapting to streaming services such as Hulu, AppleTV, and Roku? Partnerships and mergers will undoubtedly change the landscape of home entertainment as the industry responds to the changing technological possibilities. Will traditional news outlets be able to adapt to the changing news consumption patterns of the millennial generation?

Big Data analytics are being used to create new types of services by all types of companies. For example, Amazon originally competed with "big box" department stores as a low-cost provider. Amazon eventually leveraged storage and processing technologies to begin competing in streaming movie and music service, and more recently, it has leveraged Big Data to create innovative services like predictive shipping. Predictive shipping uses a customer's purchase patterns to predict when a product will be needed and ship it to the customer before the customer even realizes that she needs it! Amazon has also been successful with the sales of products like Amazon Echo that use the Alexa service to perform natural language processing. These "constantly listening" devices are embedded in homes around the world, providing Amazon with unprecedented levels and types of data that it can analyze to improve existing services and support innovation in future services.

In order to create value from their previously unused Big Data stores, companies are using new Big Data technologies. These emerging technologies allow organizations to process massive data stores of multiple formats in cost-effective ways. Some of the most frequently used Big Data technologies are Hadoop, MapReduce, and NoSQL databases.

- **Hadoop** is a Java-based, open-source, high-speed, fault-tolerant distributed storage and computational framework. Hadoop uses low-cost hardware to create clusters of thousands of computer nodes to store and process data. Hadoop originated from Google's work on distributed file systems and parallel processing and is currently supported by the Apache Software Foundation.[5] Hadoop has several modules, but the two main components are Hadoop Distributed File System (HDFS) and MapReduce.
- **Hadoop Distributed File System (HDFS)** is a highly distributed, fault-tolerant file storage system designed to manage large amounts of data at high speeds. In order to achieve high throughput, HDFS uses the write-once, read many model. This means

**Hadoop**
A Java-based, open-source, high-speed, fault-tolerant distributed storage and computational framework. Hadoop uses low-cost hardware to create clusters of thousands of computer nodes to store and process data.

**Hadoop Distributed File System (HDFS)**
A highly distributed, fault-tolerant file storage system designed to manage large amounts of data at high speeds.

[5] For more information about Hadoop, visit hadoop.apache.org.

that once the data is written, it cannot be modified. HDFS uses three types of nodes: a **name node** that stores all the metadata about the file system, a **data node** that stores fixed-size data blocks (that could be replicated to other data nodes), and a **client node** that acts as the interface between the user application and the HDFS.

- **MapReduce** is an open-source application programming interface (API) that provides fast data analytics services. MapReduce distributes the processing of the data among thousands of nodes in parallel. MapReduce works with structured and nonstructured data. The MapReduce framework provides two main functions: Map and Reduce. In general terms, the Map function takes a job and divides it into smaller units of work, and the Reduce function collects all the output results generated from the nodes and integrates them into a single result set. Although MapReduce itself is viewed as fairly limited today, it defined the paradigm for how Big Data is processed.
- **NoSQL** is a large-scale distributed database system that stores structured and unstructured data in efficient ways. NoSQL databases are discussed in more detail in Chapter 14, Big Data and NoSQL.

**name node**
One of three types of nodes used in the Hadoop Distributed File System (HDFS). The name node stores all the metadata about the file system. See also *client node* and *data node*.

**data node**
One of three types of nodes used in the Hadoop Distributed File System (HDFS). The data node stores fixed-size data blocks (that could be replicated to other data nodes). See also *client node* and *name node*.

**client node**
One of three types of nodes used in the Hadoop Distributed File System (HDFS). The client node acts as the interface between the user application and the HDFS. See also *name node* and *data node*.

**MapReduce**
An open-source application programming interface (API) that provides fast data analytics services; one of the main Big Data technologies that allows organizations to process massive data stores.

Hadoop technologies provide a framework for Big Data analytics in which data (structured or unstructured) is distributed, replicated, and processed in parallel using a network of low-cost commodity hardware. Hadoop introduced new ways to store and manage data and Hadoop-related technologies gave rise to a new generation of database systems. Do not be confused: Hadoop and NoSQL databases are often discussed together since they are both components in addressing Big Data issues. However, Hadoop is neither a database nor a data model. It is a distributed file storing and processing model. There is no Hadoop DBMS. NoSQL databases are databases, and the NoSQL model represents a different way of approaching the storage and processing of data in a nonrelational way. NoSQL databases provide distributed, fault-tolerant databases for processing nonstructured data.

With the potential of big gains derived from Big Data analytics, it is not surprising that some organizations are turning to emerging Big Data technologies, such as NoSQL databases, to mine the wealth of information hidden in mountains of web data and gain a competitive advantage.

## Note

Does this mean that relational databases don't have a place in organizations with Big Data challenges? No, relational databases remain the preferred and dominant databases to support most day-to-day transactions and structured data analytics needs. Each DBMS technology has its areas of application, and the best approach is to use the best tool for the job. In perspective, object/relational databases serve 98 percent of operational market needs. For Big Data needs, Hadoop and NoSQL databases are among the options. Chapter 14, Big Data and NoSQL, discusses these options in greater detail.

**NoSQL Databases** Every time you search for a product on Amazon, send messages to friends in Facebook, watch a video on YouTube, or search for directions in Google Maps, you are using a NoSQL database. As with any new technology, the term *NoSQL* can be loosely applied to many different types of technologies. However, this chapter uses *NoSQL* to refer to a new generation of databases that address the specific challenges of the Big Data era and have the following general characteristics:

- They are not based on the relational model and SQL; hence the name NoSQL.
- They support highly distributed database architectures.

**NoSQL**
A new generation of database management systems that is not based on the traditional relational database model.

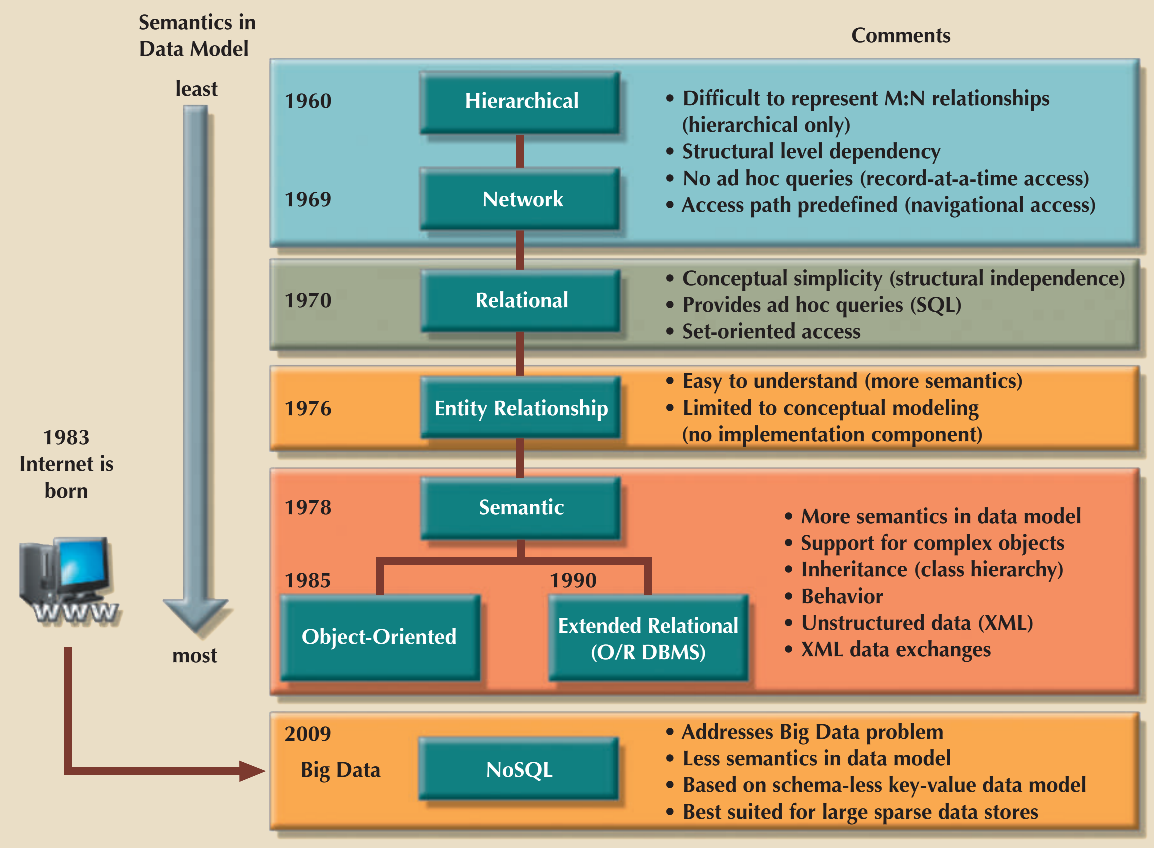
- They provide high scalability, high availability, and fault tolerance.
- They support very large amounts of sparse data (data with a large number of attributes but where the actual number of data instances is low).
- They are geared toward performance rather than transaction consistency.

Unlike the relational model, which provides a very comprehensive and cohesive approach to data storage and manipulation, the NoSQL model is a broad umbrella for a variety of approaches to data storage and manipulation. The most common of those approaches are key-value stores, document databases, columnar databases, and graph databases, as discussed in detail in Chapter 14.

## 2-5g Data Models: A Summary

The evolution of DBMSs has always been driven by the search for new ways of modeling and managing increasingly complex real-world data. A summary of the most commonly recognized data models is shown in Figure 2.5.

FIGURE 2.5 THE EVOLUTION OF DATA MODELS




In the evolution of data models, some common characteristics have made them widely accepted:

- A data model must show some degree of conceptual simplicity without compromising the semantic completeness of the database. *It does not make sense to have a data model that is more difficult to conceptualize than the real world.* At the same time, the model should show clarity and relevance; that is, the data model should be unambiguous and applicable to the problem domain. A data model must represent the real world as closely as possible. This goal is more easily realized by adding more semantics to the model's data representation. (Semantics concern dynamic data behavior, while data representation constitutes the static aspect of the real-world scenario.) In other words, the model should be accurate and complete—all the needed data is included and properly described.

- Representation of the real-world transformations (behavior) must be in compliance with the consistency and integrity characteristics required by the intended use of the data model.

Each new data model addresses the shortcomings of previous models. The network model replaced the hierarchical model because the former made it much easier to represent complex (many-to-many) relationships. In turn, the relational model offers several advantages over the hierarchical and network models through its simpler data representation, superior data independence, and easy-to-use query language; these features have made it the preferred data model for business applications. The OO data model introduced support for complex data within a rich semantic framework. The ERDM added many OO features to the relational model and allowed it to maintain strong market share within the business environment. In recent years, the Big Data phenomenon has stimulated the development of alternative ways to model, store, and manage data that represents a break with traditional data management.

It is important to note that not all data models are created equal; some data models are better suited than others for some tasks. For example, *conceptual* models are better suited for high-level data modeling, while *implementation* models are better for managing stored data for implementation purposes. The ER model is an example of a conceptual model, while the hierarchical and network models are examples of implementation models. At the same time, some models, such as the relational model and the OODM, could be used as both conceptual and implementation models. Table 2.2 summarizes the advantages and disadvantages of the various database models.

### Note

*All* databases assume the use of a common data pool within the database. Therefore, *all* database models promote data sharing, thus reducing the potential problem of islands of information.

Thus far, you have been introduced to the basic constructs of the more prominent data models. Each model uses such constructs to capture the meaning of the real-world data environment. Table 2.3 shows the basic terminology used by the various data models.

## 2-6 Degrees of Data Abstraction

If you ask 10 database designers what a data model is, you will end up with 10 different answers—depending on the degree of data abstraction. To illustrate the meaning of data abstraction, consider the example of automotive design. A car designer begins by drawing the concept of the car to be produced. Next, engineers design the details that help transfer the basic concept into a structure that can be produced. Finally, the engineering drawings are translated into production specifications to be used on the factory floor. As you can see, the process of producing the car begins at a high level of abstraction and proceeds to an ever-increasing level of detail. The factory floor process cannot proceed unless the engineering details are properly specified, and the engineering details cannot exist without the basic conceptual framework created by the designer. Designing a usable database follows the same basic process. That is, a database designer starts with an

TABLE 2.2

## ADVANTAGES AND DISADVANTAGES OF VARIOUS DATABASE MODELS

| DATA MODEL | DATA INDEPENDENCE | STRUCTURAL INDEPENDENCE | ADVANTAGES | DISADVANTAGES |
|---|---|---|---|---|
| Hierarchical | Yes | No | 1. It promotes data sharing.<br>2. Parent/child relationship promotes conceptual simplicity.<br>3. Database security is provided and enforced by DBMS.<br>4. Parent/child relationship promotes data integrity.<br>5. It is efficient with 1:M relationships. | 1. Complex implementation requires knowledge of physical data storage characteristics.<br>2. Navigational system yields complex application development, management, and use; requires knowledge of hierarchical path.<br>3. Changes in structure require changes in all application programs.<br>4. There are implementation limitations (no multiparent or M:N relationships).<br>5. There is no data definition or data manipulation language in the DBMS.<br>6. There is a lack of standards. |
| Network | Yes | No | 1. Conceptual simplicity is at least equal to that of the hierarchical model.<br>2. It handles more relationship types, such as M:N and multiparent.<br>3. Data access is more flexible than in hierarchical and file system models.<br>4. Data owner/member relationship promotes data integrity.<br>5. There is conformance to standards.<br>6. It includes data definition language (DDL) and data manipulation language (DML) in DBMS. | 1. System complexity limits efficiency—still a navigational system.<br>2. Navigational system yields complex implementation, application development, and management.<br>3. Structural changes require changes in all application programs. |
| Relational | Yes | Yes | 1. Structural independence is promoted by the use of independent tables. Changes in a table's structure do not affect data access or application programs.<br>2. Tabular view substantially improves conceptual simplicity, thereby promoting easier database design, implementation, management, and use.<br>3. Ad hoc query capability is based on SQL.<br>4. Powerful RDBMS isolates the end user from physical-level details and improves implementation and management simplicity. | 1. The RDBMS requires substantial hardware and system software overhead.<br>2. Conceptual simplicity gives relatively untrained people the tools to use a good system poorly, and if unchecked, it may produce the same data anomalies found in file systems.<br>3. It may promote islands of information problems as individuals and departments can easily develop their own applications. |
| Entity relationship | Yes | Yes | 1. Visual modeling yields exceptional conceptual simplicity.<br>2. Visual representation makes it an effective communication tool.<br>3. It is integrated with the dominant relational model. | 1. There is limited constraint representation.<br>2. There is limited relationship representation.<br>3. There is no data manipulation language.<br>4. Loss of information content occurs when attributes are removed from entities to avoid crowded displays. (This limitation has been addressed in subsequent graphical versions.) |
| Object-oriented | Yes | Yes | 1. Semantic content is added.<br>2. Visual representation includes semantic content.<br>3. Inheritance promotes data integrity. | 1. Slow development of standards caused vendors to supply their own enhancements, thus eliminating a widely accepted standard.<br>2. It is a complex navigational system.<br>3. There is a steep learning curve.<br>4. High system overhead slows transactions. |
| NoSQL | Yes | Yes | 1. High scalability, availability, and fault tolerance are provided.<br>2. It uses low-cost commodity hardware.<br>3. It supports Big Data.<br>4. Key-value model improves storage efficiency. | 1. Complex programming is required.<br>2. There is no relationship support—only by application code.<br>3. There is no transaction integrity support.<br>4. In terms of data consistency, it provides an eventually consistent model. |

TABLE 2.3

**DATA MODEL BASIC TERMINOLOGY COMPARISON**

| REAL WORLD | EXAMPLE | FILE PROCESSING | HIERARCHICAL MODEL | NETWORK MODEL | RELATIONAL MODEL | ER MODEL | OO MODEL |
|---|---|---|---|---|---|---|---|
| A group of vendors | Vendor file cabinet | File | Segment type | Record type | Table | Entity set | Class |
| A single vendor | Global supplies | Record | Segment occurrence | Current record | Row (tuple) | Entity occurrence | Object instance |
| The contact name | Johnny Ventura | Field | Segment field | Record field | Table attribute | Entity attribute | Object attribute |
| The vendor identifier | G12987 | Index | Sequence field | Record key | Key | Entity identifier | Object identifier |

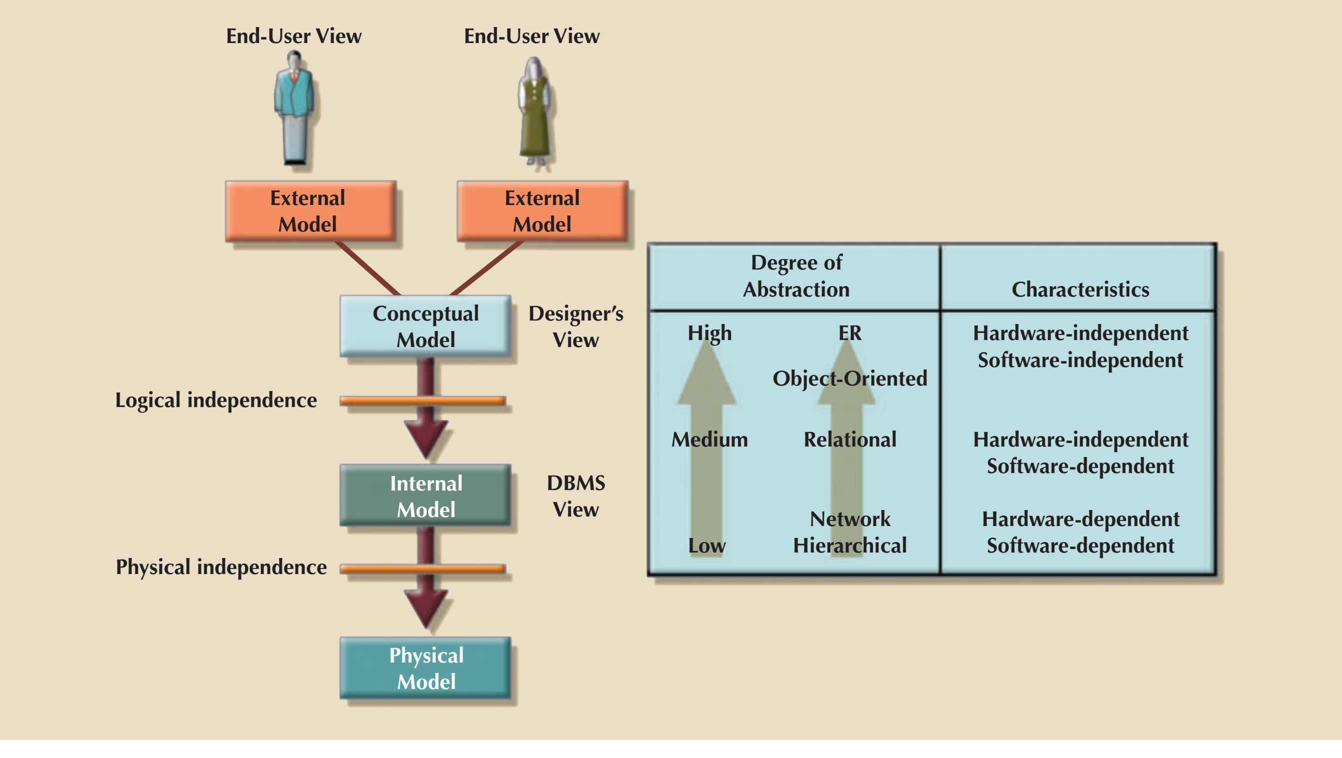
*Note:* For additional information about the terms used in this table, consult the corresponding chapters and online appendixes that accompany this book. For example, if you want to know more about the OO model, refer to Appendix G, Object-Oriented Databases.

**American National Standards Institute (ANSI)**
The group that accepted the DBTG recommendations and augmented database standards in 1975 through its SPARC committee.

abstract view of the overall data environment and adds details as the design comes closer to implementation. Using levels of abstraction can also be very helpful in integrating multiple (and sometimes conflicting) views of data at different levels of an organization.

In the early 1970s, the **American National Standards Institute (ANSI)** Standards Planning and Requirements Committee (SPARC) defined a framework for data modeling based on degrees of data abstraction. The resulting ANSI/SPARC architecture defines three levels of data abstraction: external, conceptual, and internal. You can use this framework to better understand database models, as shown in Figure 2.6. In the figure, the ANSI/SPARC framework has been expanded with the addition of a *physical* model to explicitly address physical-level implementation details of the internal model.

FIGURE 2.6 DATA ABSTRACTION LEVELS



## 2-6a The External Model

The **external model** is the end users' view of the data environment. The term *end users* refers to people who use the application programs to manipulate the data and generate information. End users usually operate in an environment in which an application has a specific business unit focus. Companies are generally divided into several business units, such as sales, finance, and marketing. Each business unit is subject to specific constraints and requirements, and each one uses a subset of the overall data in the organization. Therefore, end users within those business units view their data subsets as separate from or external to other units within the organization.

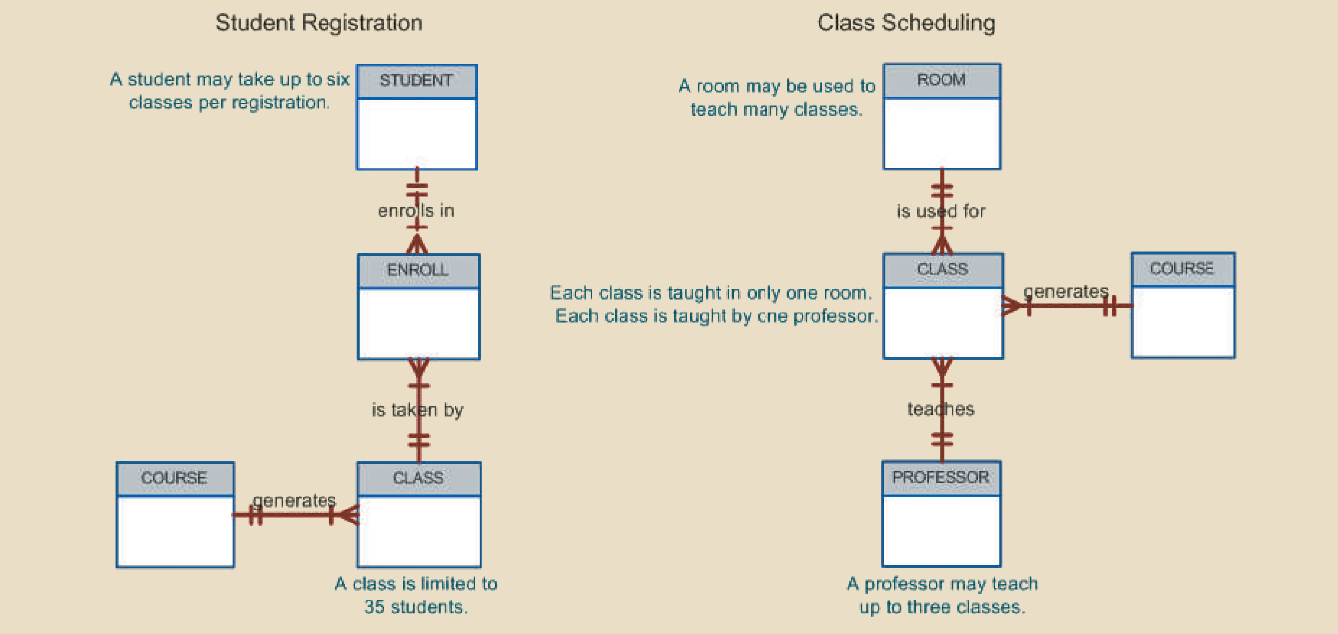
Because data is being modeled, ER diagrams will be used to represent the external views. A specific representation of an external view is known as an **external schema**. To illustrate the external model's view, examine the data environment of Tiny College.

Figure 2.7 presents the external schemas for two Tiny College business units: student registration and class scheduling. Each external schema includes the appropriate entities, relationships, processes, and constraints imposed by the business unit. Also note that *although the application views are isolated from each other, each view shares a common entity with the other view*. For example, the registration and scheduling external schemas share the entities CLASS and COURSE.

**external model**
The application programmer's view of the data environment. Given its business focus, an external model works with a data subset of the global database schema.

**external schema**
The specific representation of an external view; the end user's view of the data environment.

FIGURE 2.7 EXTERNAL MODELS FOR TINY COLLEGE




Note the ERs represented in Figure 2.7:

- A PROFESSOR may teach many CLASSes, and each CLASS is taught by only one PROFESSOR; there is a 1:M relationship between PROFESSOR and CLASS.
- A CLASS may ENROLL many students, and each STUDENT may ENROLL in many CLASSes, thus creating an M:N relationship between STUDENT and CLASS. (You will learn about the precise nature of the ENROLL entity in Chapter 4.)
- Each COURSE may generate many CLASSes, but each CLASS references a single COURSE. For example, there may be several classes (sections) of a database course that have a course code of CIS-420. One of those classes might be offered on MWF from 8:00 a.m. to 8:50 a.m., another might be offered on MWF from 1:00 p.m. to 1:50

p.m., while a third might be offered on Thursdays from 6:00 p.m. to 8:40 p.m. Yet, all three classes have the course code CIS-420.

- Finally, a CLASS requires one ROOM, but a ROOM may be scheduled for many CLASSes. That is, each classroom may be used for several classes: one at 9:00 a.m., one at 11:00 a.m., and one at 1:00 p.m., for example. In other words, there is a 1:M relationship between ROOM and CLASS.

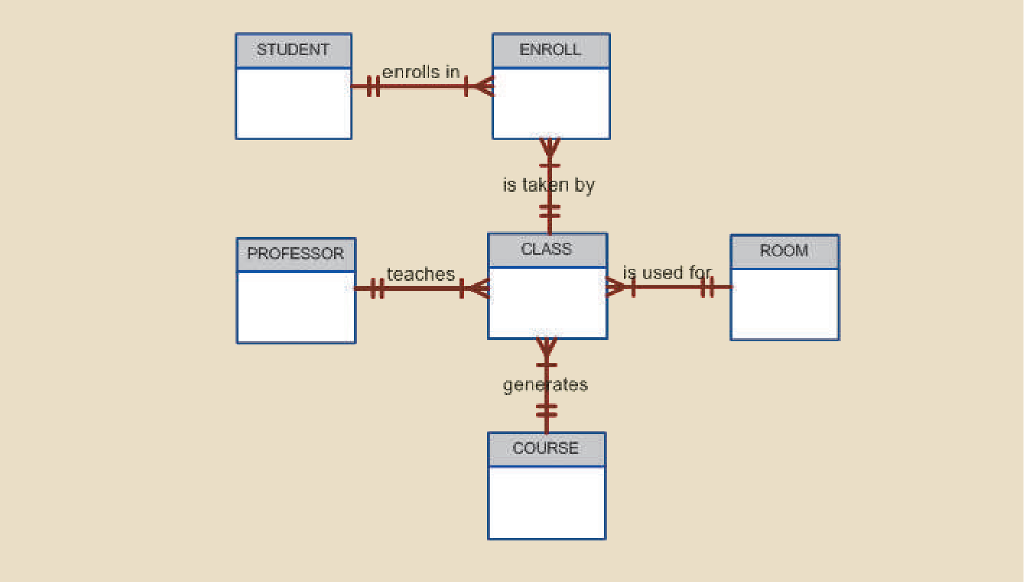
The use of external views that represent subsets of the database has some important advantages:

- It is easy to identify specific data required to support each business unit's operations.
- It makes the designer's job easy by providing feedback about the model's adequacy. Specifically, the model can be checked to ensure that it supports all processes as defined by their external models, as well as all operational requirements and constraints.
- It helps to ensure *security* constraints in the database design. Damaging an entire database is more difficult when each business unit works with only a subset of data.
- It makes application program development much simpler.

## 2-6b The Conceptual Model

The **conceptual model** represents a global view of the entire database by the entire organization. That is, the conceptual model integrates all external views (entities, relationships, constraints, and processes) into a single global view of the data in the enterprise, as shown in Figure 2.8. Also known as a **conceptual schema**, it is the basis for the identification and high-level description of the main data objects (avoiding any database model-specific details).

FIGURE 2.8 A CONCEPTUAL MODEL FOR TINY COLLEGE




The most widely used conceptual model is the ER model. Remember that the ER model is illustrated with the help of the ERD, which is effectively the basic database blueprint. The ERD is used to graphically *represent* the conceptual schema.

The conceptual model yields some important advantages. First, it provides a bird's-eye (macro level) view of the data environment that is relatively easy to understand. For example, you can get a summary of Tiny College's data environment by examining the conceptual model in Figure 2.8.

**conceptual model**
The output of the conceptual design process. The conceptual model provides a global view of an entire database and describes the main data objects, avoiding details.

**conceptual schema**
A representation of the conceptual model, usually expressed graphically. See also *conceptual model*.

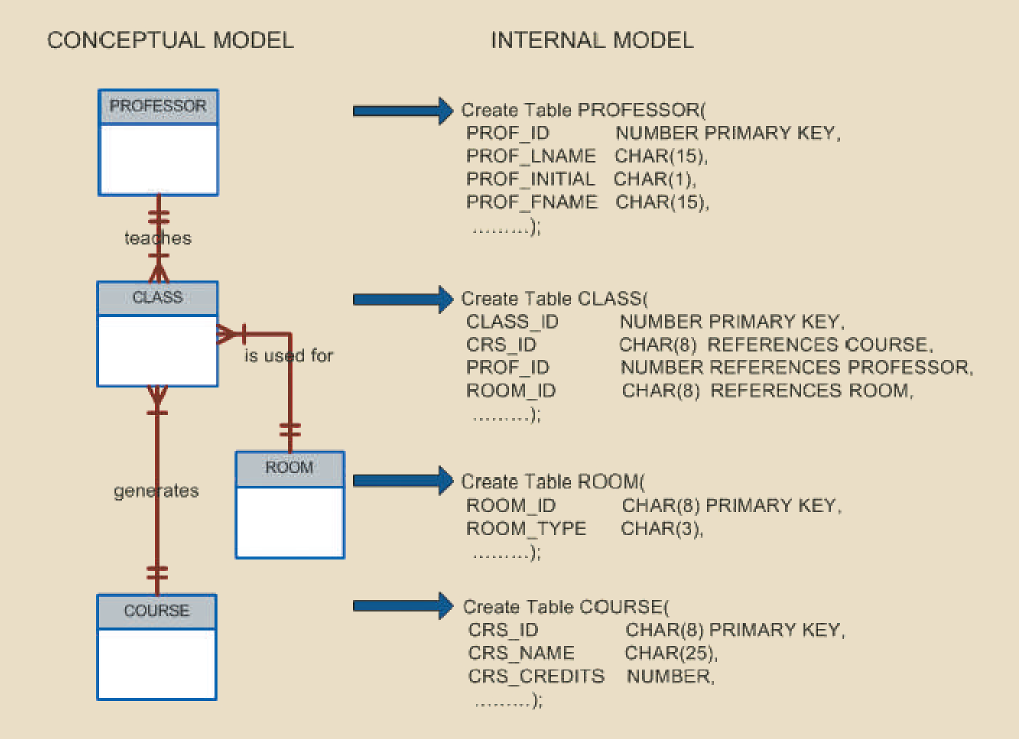
Second, the conceptual model is independent of both software and hardware. **Software independence** means that the model does not depend on the DBMS software used to implement the model. **Hardware independence** means that the model does not depend on the hardware used in the implementation of the model. Therefore, changes in either the hardware or the DBMS software will have no effect on the database design at the conceptual level. Generally, the term **logical design** refers to the task of creating a conceptual data model that could be implemented in any DBMS.

## 2-6c The Internal Model

Once a specific DBMS has been selected, the internal model maps the conceptual model to the DBMS. The **internal model** is the representation of the database as "seen" by the DBMS. In other words, the internal model requires the designer to match the conceptual model's characteristics and constraints to those of the selected implementation model. An **internal schema** depicts a specific representation of an internal model, using the database constructs supported by the chosen database.

Because this book focuses on the relational model, a relational database was chosen to implement the internal model. Therefore, the internal schema should map the conceptual model to the relational model constructs. In particular, the entities in the conceptual model are mapped to tables in the relational model. Likewise, because a relational database has been selected, the internal schema is expressed using SQL, the standard language for relational databases. In the case of the conceptual model for Tiny College depicted in Figure 2.8, the internal model was implemented by creating the tables PROFESSOR, COURSE, CLASS, STUDENT, ENROLL, and ROOM. A simplified version of the internal model for Tiny College is shown in Figure 2.9.

FIGURE 2.9 INTERNAL MODEL FOR TINY COLLEGE




The development of a detailed internal model is especially important to database designers who work with hierarchical or network models because those models require precise specification of data storage location and data access paths. In contrast,

**software independence**
A property of any model or application that does not depend on the software used to implement it.

**hardware independence**
A condition in which a model does not depend on the hardware used in the model's implementation. Therefore, changes in the hardware will have no effect on the database design at the conceptual level.

**logical design**
A stage in the design phase that matches the conceptual design to the requirements of the selected DBMS and is therefore software-dependent. Logical design is used to translate the conceptual design into the internal model for a selected database management system, such as DB2, SQL Server, Oracle, IMS, Informix, Access, or Ingress.

**internal model**
In database modeling, a level of data abstraction that adapts the conceptual model to a specific DBMS model for implementation. The internal model is the representation of a database as "seen" by the DBMS. In other words, the internal model requires a designer to match the conceptual model's characteristics and constraints to those of the selected implementation model.

**internal schema**
A representation of an internal model using the database constructs supported by the chosen database.

the relational model requires less detail in its internal model because most RDBMSs handle data access path definition *transparently*; that is, the designer need not be aware of the data access path details. Nevertheless, even relational database software usually requires specifications of data storage locations, especially in a mainframe environment. For example, DB2 requires that you specify the data storage group, the location of the database within the storage group, and the location of the tables within the database.

Because the internal model depends on specific database software, it is said to be software dependent. Therefore, a change in the DBMS software requires that the internal model be changed to fit the characteristics and requirements of the implementation database model. When you can change the internal model without affecting the conceptual model, you have **logical independence**. However, the internal model is still hardware independent because it is unaffected by the type of computer on which the software is installed. Therefore, a change in storage devices or even a change in operating systems will not affect the internal model.

**logical independence**
A condition in which the internal model can be changed without affecting the conceptual model. (The internal model is hardware-independent because it is unaffected by the computer on which the software is installed. Therefore, a change in storage devices or operating systems will not affect the internal model.)

**physical model**
A model in which physical characteristics such as location, path, and format are described for the data. The physical model is both hardware- and software-dependent. See also *physical design*.

**physical independence**
A condition in which the physical model can be changed without affecting the internal model.

## 2-6d The Physical Model

The **physical model** operates at the lowest level of abstraction, describing the way data is saved on storage media such as magnetic, solid state, or optical media. The physical model requires the definition of both the physical storage devices and the (physical) access methods required to reach the data within those storage devices, making it both software and hardware dependent. The storage structures used are dependent on the software (the DBMS and the operating system) and on the type of storage devices the computer can handle. The precision required in the physical model's definition demands that database designers have a detailed knowledge of the hardware and software used to implement the database design.

Early data models forced the database designer to take the details of the physical model's data storage requirements into account. However, the now-dominant relational model is aimed largely at the logical level rather than at the physical level; therefore, it does not require the physical-level details common to its predecessors.

Although the relational model does not require the designer to be concerned about the data's physical storage characteristics, the *implementation* of a relational model may require physical-level fine-tuning for increased performance. Fine-tuning is especially important when very large databases are installed in a mainframe environment, yet even such performance fine-tuning at the physical level does not require knowledge of physical data storage characteristics.

As noted earlier, the physical model is dependent on the DBMS, methods of accessing files, and types of hardware storage devices supported by the operating system. When you can change the physical model without affecting the internal model, you have **physical independence**. Therefore, a change in storage devices or methods and even a change in operating system will not affect the internal model.

The levels of data abstraction are summarized in Table 2.4.

TABLE 2.4
**LEVELS OF DATA ABSTRACTION**

| MODEL | DEGREE OF ABSTRACTION | FOCUS | INDEPENDENT OF |
|---|---|---|---|
| External | High ↕ Low | End-user views | Hardware and software |
| Conceptual | | Global view of data (database model independent) | Hardware and software |
| Internal | | Specific database model | Hardware |
| Physical | | Storage and access methods | Neither hardware nor software |

## Summary

- A data model is an abstraction of a complex real-world data environment. Database designers use data models to communicate with programmers and end users. The basic data-modeling components are entities, attributes, relationships, and constraints. Business rules are used to identify and define the basic modeling components within a specific real-world environment.
- The hierarchical and network data models were early models that are no longer used, but some of the concepts are found in current data models.
- The relational model is the current database implementation standard. In the relational model, the end user perceives the data as being stored in tables. Tables are related to each other by means of common values in common attributes. The entity relationship (ER) model is a popular graphical tool for data modeling that complements the relational model. The ER model allows database designers to visually present different views of the data—as seen by database designers, programmers, and end users—and to integrate the data into a common framework.
- The object-oriented data model (OODM) uses objects as the basic modeling structure. Like the relational model's entity, an object is described by its factual content. Unlike an entity, however, the object also includes information about relationships between the facts, as well as relationships with other objects, thus giving its data more meaning.
- The relational model has adopted many object-oriented (OO) extensions to become the extended relational data model (ERDM). Object/relational database management systems (O/R DBMS) were developed to implement the ERDM. At this point, the OODM is largely used in specialized engineering and scientific applications, while the ERDM is primarily geared to business applications.
- Big Data technologies such as Hadoop, MapReduce, and NoSQL provide distributed, fault-tolerant, and cost-efficient support for Big Data analytics. NoSQL databases are a new generation of databases that do not use the relational model and are geared to support the very specific needs of Big Data organizations. NoSQL databases offer distributed data stores that provide high scalability, availability, and fault tolerance by sacrificing data consistency and shifting the burden of maintaining relationships and data integrity to the program code.
- Data-modeling requirements are a function of different data views (global versus local) and the level of data abstraction. The American National Standards Institute Standards Planning and Requirements Committee (ANSI/SPARC) describes three levels of data abstraction: external, conceptual, and internal. The fourth and lowest level of data abstraction, called the physical level, is concerned exclusively with physical storage methods.

# Key Terms

3 Vs
American National Standards Institute (ANSI)
attribute
Big Data
business rule
Chen notation
class
class diagram
class diagram notation
class hierarchy
client node
conceptual model
conceptual schema
connectivity
constraint
Crow's Foot notation
data definition language (DDL)
data manipulation language (DML)
data model
data modeling
data node
entity
entity instance
entity occurrence
entity relationship (ER) model (ERM)
entity relationship diagram (ERD)
entity set
extended relational data model (ERDM)
Extensible Markup Language (XML)
external model
external schema
Hadoop
Hadoop Distributed File System (HDFS)
hardware independence
hierarchical model
inheritance
internal model
internal schema
logical design
logical independence
MapReduce
many-to-many (M:N or *..*) relationship
method
name node
network model
NoSQL
object
object/relational database management system (O/R DBMS)
object-oriented data model (OODM)
object-oriented database management system (OODBMS)
one-to-many (1:M or 1..*) relationship
one-to-one (1:1 or 1..1) relationship
physical independence
physical model
relation
relational database management system (RDBMS)
relational diagram
relational model
relationship
schema
segment
semantic data model
software independence
subschema
table
tuple
Unified Modeling Language (UML)

# Review Questions

1. Discuss the importance of data models.
2. What is a business rule, and what is its purpose in data modeling?
3. How do you translate business rules into data model components?
4. Describe the basic features of the relational data model and discuss their importance to the end user and the designer.

5. Explain how the entity relationship (ER) model helped produce a more structured relational database design environment.
6. Consider the scenario described by the statement "A customer can make many payments, but each payment is made by only one customer." Use this scenario as the basis for an entity relationship diagram (ERD) representation.
7. Why is an object said to have greater semantic content than an entity?
8. What is the difference between an object and a class in the object-oriented data model (OODM)?
9. How would you model Question 6 with an OODM? (Use Figure 2.4 as your guide.)
10. What is an ERDM, and what role does it play in the modern (production) database environment?
11. What is a relationship, and what three types of relationships exist?
12. Give an example of each of the three types of relationships.
13. What is a table, and what role does it play in the relational model?
14. What is a relational diagram? Give an example.
15. What is connectivity? (Use a Crow's Foot ERD to illustrate connectivity.)
16. Describe the Big Data phenomenon.
17. What does the term *3 Vs* refer to?
18. What is Hadoop, and what are its basic components?
19. What are the basic characteristics of a NoSQL database?
20. Using the example of a medical clinic with patients and tests, provide a simple representation of how to model this example using the relational model and how it would be represented using the key-value data modeling technique.
21. What is logical independence?
22. What is physical independence?

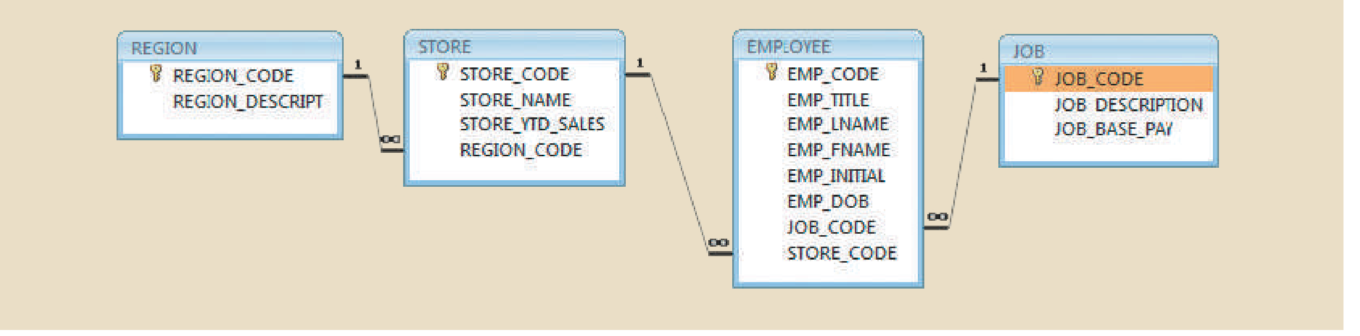
## Problems

Use the contents of Figure 2.1 to work Problems 1–3.

1. Write the business rule(s) that govern the relationship between AGENT and CUSTOMER.
2. Given the business rule(s) you wrote in Problem 1, create the basic Crow's Foot ERD.
3. Using the ERD you drew in Problem 2, create the equivalent object representation and UML class diagram. (Use Figure 2.4 as your guide.)

Using Figure P2.4 as your guide, work Problems 4–5. The DealCo relational diagram shows the initial entities and attributes for the DealCo stores, which are located in two regions of the country.

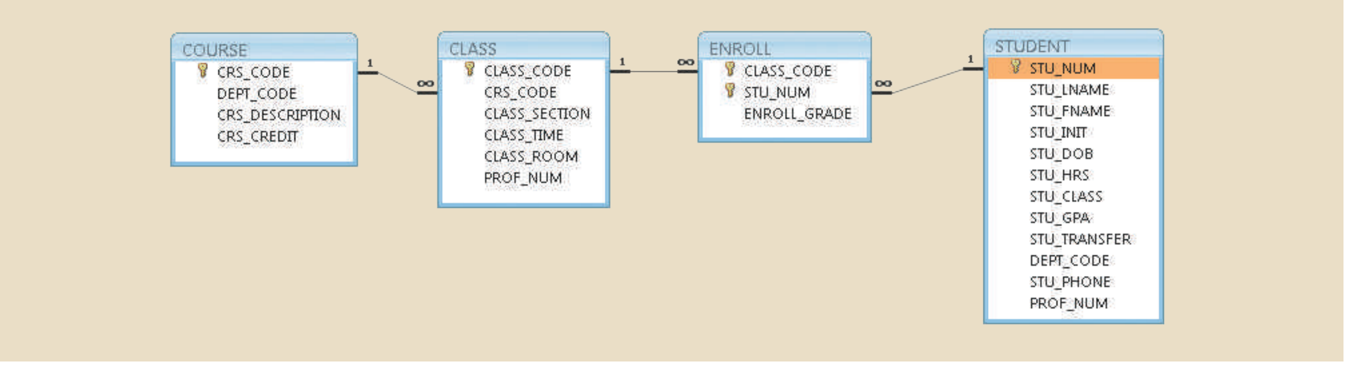
FIGURE P2.4 THE DEALCO RELATIONAL DIAGRAM




4. Identify each relationship type and write all of the business rules.
5. Create the basic Crow's Foot ERD for DealCo.

Using Figure P2.6 as your guide, work Problems 6–8. The Tiny College relational diagram shows the initial entities and attributes for the college.

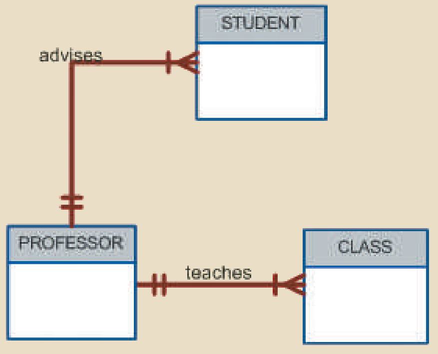
FIGURE P2.6 THE TINY COLLEGE RELATIONAL DIAGRAM




6. Identify each relationship type and write all of the business rules.
7. Create the basic Crow's Foot ERD for Tiny College.
8. Create the UML class diagram that reflects the entities and relationships you identified in the relational diagram.
9. Typically, a hospital patient receives medications that have been ordered by a particular doctor. Because the patient often receives several medications per day, there is a 1:M relationship between PATIENT and ORDER. Similarly, each order can include several medications, creating a 1:M relationship between ORDER and MEDICATION.
   a. Identify the business rules for PATIENT, ORDER, and MEDICATION.
   b. Create a Crow's Foot ERD that depicts a relational database model to capture these business rules.
10. United Broke Artists (UBA) is a broker for not-so-famous artists. UBA maintains a small database to track painters, paintings, and galleries. A painting is created by a particular artist and then exhibited in a particular gallery. A gallery can exhibit many paintings, but each painting can be exhibited in only one gallery. Similarly, a painting is created by a single painter, but each painter can create many paintings. Using PAINTER, PAINTING, and GALLERY, in terms of a relational database:
    a. What tables would you create, and what would the table components be?
    b. How might the (independent) tables be related to one another?

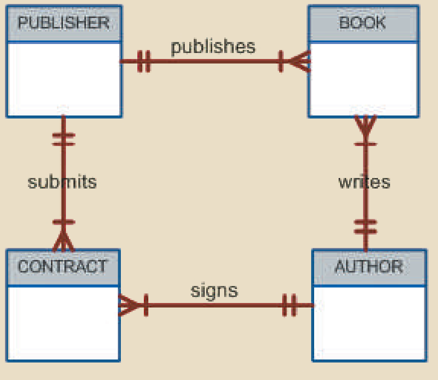
11. Using the ERD from Problem 10, create the relational schema. (Create an appropriate collection of attributes for each of the entities. Make sure you use the appropriate naming conventions to name the attributes.)
12. Convert the ERD from Problem 10 into a corresponding UML class diagram.
13. Describe the relationships (identify the business rules) depicted in the Crow's Foot ERD shown in Figure P2.13.

FIGURE P2.13 THE CROW'S FOOT ERD FOR PROBLEM 13




14. Create a Crow's Foot ERD to include the following business rules for the ProdCo company:
    a. Each sales representative writes many invoices.
    b. Each invoice is written by one sales representative.
    c. Each sales representative is assigned to one department.
    d. Each department has many sales representatives.
    e. Each customer can generate many invoices.
    f. Each invoice is generated by one customer.
15. Write the business rules that are reflected in the ERD shown in Figure P2.15. (Note that the ERD reflects some simplifying assumptions. For example, each book is written by only one author. Also, remember that the ERD is always read from the "1" to the "M" side, regardless of the orientation of the ERD components.)

FIGURE P2.15 THE CROW'S FOOT ERD FOR PROBLEM 15



16. Create a Crow's Foot ERD for each of the following descriptions. (Note that the word *many* merely means *more than one* in the database modeling environment.)
    a. Each of the MegaCo Corporation's divisions is composed of many departments. Each department has many employees assigned to it, but each employee works for only one department. Each department is managed by one employee, and each of those managers can manage only one department at a time.
    b. During some period of time, a customer can download many ebooks from BooksOnline. Each of the ebooks can be downloaded by many customers during that period of time.
    c. An airliner can be assigned to fly many flights, but each flight is flown by only one airliner.
    d. The KwikTite Corporation operates many factories. Each factory is located in a region, and each region can be "home" to many of KwikTite's factories. Each factory has many employees, but each employee is employed by only one factory.
    e. An employee may have earned many degrees, and each degree may have been earned by many employees.
17. Write the business rules that are reflected in the ERD shown in Figure P2.17.

FIGURE P2.17 THE CROW'S FOOT ERD FOR PROBLEM 17