

PART I

DATABASE CONCEPTS

DATABASE SYSTEMS	1
DATA MODELS	2



In 1979, Oracle CEO Larry Ellison, who was closely following the work of researchers at IBM and the University of California at Berkeley, released the first commercial database management system (DBMS) based on the relational model. Relational databases (RDBMSs) burst on the scene in the 1980s and quickly dominated the database market.

For more than 40 years, the RDBMS industry has expanded rapidly, with Oracle maintaining its leading position. In 2010, Gartner, the leading information technology research company, reported that Oracle held a greater share of RDBMS revenue than its five closest competitors combined. In 2010, Oracle's revenue grew 10.9 percent, ahead of the 9.9 percent industry average.¹

In 2008, a group of leading database researchers met in Berkeley and issued a report declaring that the industry had reached an exciting turning point and was on the verge of another database revolution.² Over time, analysts, journalists, and business leaders have seen that new developments, such as the explosion of unstructured data in cyberspace, the growing importance of business intelligence, and the emergence of cloud technologies, may require the development of new database models.

Although relational databases meet rigorous standards for data integrity and consistency, they do not scale unstructured data as well as new database models such as NoSQL.³ As a result, social networking sites such as Twitter and Facebook, which do not require high levels of data consistency and integrity, have adopted NoSQL databases. These types of sites are home to semistructured data such as Web logs and instant messages, giving the NoSQL model an edge in these domains. In addition, as the number of data transactions has skyrocketed in recent years, many "Big Data" companies have turned to these new data models to process their data. When Visa needed to process two years of credit transactions, the company turned to new technologies such as Hadoop to handle the 70 billion transactions; processing time was reduced from one month to 13 minutes.⁴

Another technology that is fast gaining ground is in-memory databases, which store data in main memory rather than secondary storage. In-memory databases take advantage of parallel computing and multicore processors to deliver faster business intelligence.⁵ Additionally, cloud computing provides on-demand access to clusters of computers on the Web at a reasonable cost. Cloud computing does not require that companies invest heavily in massive server infrastructure.⁶ All these technology advances are revolutionizing the database market. For example, in 2009, SAP co-founder Hasso Plattner announced that his company was taking a lead in developing in-memory databases. He said SAP was close to developing an in-memory system that could accommodate a database large enough for Walmart's needs.

However, the relational model is not standing idly by. Today, most large relational database vendors provide in-memory database capabilities. For example, in 2005, Oracle acquired in-memory database technology with the purchase of TimesTen. Cloud computing is also enabling the widespread adoption of relational databases in organizations that could not previously afford a high-availability infrastructure. The relational model has a history of overcoming challenges from other emerging database technologies, which is why the overwhelming majority of existing systems still rely on relational databases. It remains to be seen if the relational model will evolve one more time and adapt to these new challenges. The race is on.

¹ "Oracle Is #1 in the RDBMS Sector for 2010," Oracle Website, www.oracle.com/us/products/database/number-one-database-069037.html.

² Rakesh Agrawal et al., "The Claremont Report on Database Research," <http://db.cs.berkeley.edu/claremont/claremontreport08.pdf>.

³ Guy Harrison, "10 Things You Should Know About NoSQL Databases," *TechRepublic*, August 26, 2010, www.techrepublic.com/blog/10things/10-things-you-should-know-about-nosql-databases/1772.

⁴ David Strom, "NoSQL: Breaking Free of Structured Data," *IT World*, June 9, 2011, www.itworld.com/data-centerservers/172477/nosql-breaking-free-structured-data.

⁵ Mike Simons, "SAP Co-founder Hasso Plattner Promises in-Memory Database Revolution," *CIO*, May 14, 2009, www.cio.com/article/492832/SAP_Co-founder_Hasso_Plattner_Promises_in_Memory_Database_Revolution.

⁶ Joshua Greenbaum, "A Revolution Threatens the Relational Database," *IT Management*, May 13, 2008, <http://itmanagement.earthweb.com/columns/entad/article.php/3746191>.

Business
Vignette

In this chapter, you will learn:

- The difference between data and information
- What a database is, the various types of databases, and why they are valuable assets for decision making
- The importance of database design
- How modern databases evolved from file systems
- About flaws in file system data management
- The main components of the database system
- The main functions of a database management system (DBMS)

Good decisions require good information that is derived from raw facts. These raw facts are known as data. Data are likely to be managed most efficiently when they are stored in a database. In this chapter, you will learn what a database is, what it does, and why it yields better results than other data management methods. You will also learn about various types of databases and why database design is so important.

Databases evolved from computer file systems. Although file system data management is now largely outmoded, understanding the characteristics of file systems is important because file systems are the source of serious data management limitations. In this chapter, you will also learn how the database system approach helps eliminate most of the shortcomings of file system data management.

Preview

1.1 WHY DATABASES?

Imagine trying to operate a business without knowing who your customers are, what products you are selling, who is working for you, who owes you money, and to whom you owe money. All businesses have to keep this type of data and much more; just as importantly, they must have those data available to decision makers when necessary. It can be argued that the ultimate purpose of all business information systems is to help businesses use information as an organizational resource. At the heart of all of these systems are the collection, storage, aggregation, manipulation, dissemination, and management of data.

Depending on the type of information system and the characteristics of the business, these data could vary from a few megabytes on just one or two topics to terabytes covering hundreds of topics within the business's internal and external environment. Telecommunications companies such as Sprint and AT&T are known to have systems that keep data on trillions of phone calls, with new data being added to the system at speeds up to 70,000 calls per second!¹ Not only do these companies have to store and manage immense collections of data, they have to be able to find any given fact in that data quickly. Consider the case of Internet search staple Google. While Google is reluctant to disclose many details about its data storage specifications, it is estimated that the company responds to over 91 million searches per day across a collection of data that is several terabytes in size. Impressively, the results of these searches are available almost instantly.

How can these businesses process this much data? How can they store it all, and then quickly retrieve just the facts that decision makers want to know, just when they want to know it? The answer is that they use databases. Databases, as explained in detail throughout this book, are specialized structures that allow computer-based systems to store, manage, and retrieve data very quickly. Virtually all modern business systems rely on databases; therefore, a good understanding of how these structures are created and their proper use is vital for any information systems professional. Even if your career does not take you down the amazing path of database design and development, databases will be a key component underpinning the systems that you use. In any case, you will probably make decisions in your career based on information generated from data. Thus, it is important that you know the difference between data and information.

1.2 DATA VS. INFORMATION

To understand what drives database design, you must understand the difference between data and information. **Data** are raw facts. The word raw indicates that the facts have not yet been processed to reveal their meaning. For example, suppose that a university tracks data on faculty members for reporting to accrediting bodies. To get the data for each faculty member into the database, you would provide a screen to allow for convenient data entry, complete with drop-down lists, combo boxes, option buttons, and other data-entry validation controls. Figure 1.1, Panel A, shows a simple data-entry form from a software package named Sedona. When the data are entered into the form and saved, they are placed in the underlying database as raw data, as shown in Figure 1.1, Panel B. Although you now have the facts in hand, they are not particularly useful in this format. Reading through hundreds of rows of data for faculty members does not provide much insight into the overall makeup of the faculty. Therefore, you transform the raw data into a data summary like the one shown in Figure 1.1, Panel C. Now you can get quick answers to questions such as "What percentage of the faculty in the Information Systems (INFS) department are adjuncts?" In this case, you can quickly determine that 20% of the INFS faculty members are adjunct faculty. Because graphics can enhance your ability to quickly extract meaning from data, you show the data summary pie chart in Figure 1.1, Panel D.

¹ "Top Ten Largest Databases in the World," *Business Intelligence Lowdown*, February 15, 2007, http://www.businessintelligencelowdown.com/2007/02/top_10_largest_.html.

FIGURE 1.1 Transforming raw data into information

a) Data entry screen

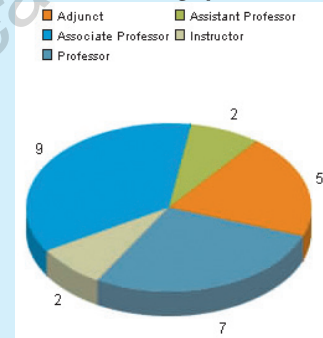
b) Raw data

ID	Last Name	Mid Name	First Name	Dept Code	Office	Email	Rank	Hire Year	Degree
1	Washington	A.	George	MGMT	N135	gswashington@mtsu.edu	Professor	2001	Ph.D.
2	Adams	John	John	FIN	N313	jadams@mtsu.edu	Professor	1984	Ph.D.
3	Jefferson	L.	Thames	ECON		tjefferson@mtsu.edu	Instructor	2002	M.B.A.
4	Madison	D.	James	FIN	N236	jmadison@mtsu.edu	Associate Professor	1984	Ph.D.
5	Monroe	N.	James	ACCT	N411	jmonroe@mtsu.edu	Assistant Professor	1995	Ph.D.
6	Adams	J.	John	ACCT	N418	jadams@mtsu.edu	Associate Professor	1989	Ph.D.
7	Jackson	C.	Andrew	ECON	N303	ajackson@mtsu.edu	Associate Professor	1993	Ph.D.
8	Van Buren	T.	Martin	FIN	N306	mvanburen@mtsu.edu	Professor	1980	Ph.D.
9	Harrison	R.	William	MGMT	N118	wharrison@mtsu.edu	Professor	1994	Ph.D.
10	Tyler	M.	John	MGMT		jtyler@mtsu.edu	Assistant Professor	2000	Ed.D.
11	Polk	Cheryl		MGMT	N143	cpolk@mtsu.edu	Associate Professor	2002	Ph.D.
12	Taylor	G.	Zachary	ACCT	N415	ztaylor@mtsu.edu	Associate Professor	1996	Ph.D.
13	Filmore	G.	Milard	FIN	N219	mfilmore@mtsu.edu	Professor	1992	Ph.D.
14	Pierce	A.	Franklin	MGMT	N359	fpierce@mtsu.edu	Instructor	2005	M.B.A.
15	Buchanan	T.	James	MGMT	N146	jbuchanan@mtsu.edu	Associate Professor	1996	D.B.A.
17	Lincoln	W.	Larry	MGMT	N150	lincoln@mtsu.edu	Associate Professor	1996	Ph.D.
18	Johnson	Andrew		ISYS	N360	ajohnson@mtsu.edu	Professor	1987	Ph.D.
19	Grant	Kate		MGMT	N120	kgrant@mtsu.edu	Assistant Professor	1989	D.B.A.
20	Rutherford	Hayes		ACCT	N408	hrutherford@mtsu.edu	Professor	1992	Ph.D.
21	Grefeld	T.	Denise	ACCT		drefeld@mtsu.edu	Assistant Professor	2018	Ph.D.
22	Arthur	Emily		ACCT	N413	earthur@mtsu.edu	Associate Professor	2003	J.D.
23	Owensland	G.	Robert	ACCT	N401	robertowensland@mtsu.edu	Associate Professor	1987	Ph.D.
24	Harrison	X.	Patricia	BULA	N405	pharrison@mtsu.edu	Associate Professor	2001	J.D.
25	McKinley	B.	Priscilla	ISYS	N363	bmckinley@mtsu.edu	Adjunct	1994	M.S.
26	Roosevelt	F.	Hilary	MGMT	N104	hroosevelt@mtsu.edu	Associate Professor	2002	Ph.D.
27	Wilson	Laure		BCEN	N448	lwilson@mtsu.edu	Professor	1992	Ph.D.
28	Harding	Wendy		MGMT	N114	wharding@mtsu.edu	Professor	1984	Ed.D.
29	Coolidge	Calvin		ECON	N316	ccoolidge@mtsu.edu	Professor	1975	Ph.D.
30	Hoover	Lisa		MGMT		lhoover@mtsu.edu	Adjunct	1978	M.B.A.
31	Truman	Betty		ACCT	N416	btruman@mtsu.edu	Professor	1971	Ed.D.
32	Johnson	Robert		BCEN	N240	rjohnson@mtsu.edu	Professor	2001	Ph.D.

c) Information in summary format

Rank	COUNT	%/INFS	TOT/COL	%/COL. TOT.	%/COL. FAC.
Adjunct	5	20.00%	23	21.74%	3.27%
Assistant Professor	2	8.00%	28	7.14%	1.31%
Associate Professor	9	36.00%	37	24.32%	5.88%
Instructor	2	8.00%	18	11.11%	1.31%
Professor	7	28.00%	47	14.89%	4.58%

d) Information in graphical format



SOURCE: Course Technology/Cengage Learning
Data entry screen courtesy of Sedona Systems, 2011.
Information screens courtesy of JCBDashboard, 2011.

Information is the result of processing raw data to reveal its meaning. Data processing can be as simple as organizing data to reveal patterns or as complex as making forecasts or drawing inferences using statistical modeling. To reveal meaning, information requires *context*. For example, an average temperature reading of 105 degrees does not mean much unless you also know its context: Is this reading in degrees Fahrenheit or Celsius? Is this a machine temperature, a body temperature, or an outside air temperature? Information can be used as the foundation for decision making. For example, the data summary for the faculty can provide accrediting bodies with insights that are useful in determining whether to renew accreditation for the university.

Keep in mind that raw data must be properly *formatted* for storage, processing, and presentation. For example, dates might be stored in Julian calendar formats within the database, but displayed in a variety of formats, such as day-month-year or month/day/year, for different purposes. Respondents' yes/no responses might need to be converted to a Y/N, or 0/1, format for data storage. More complex formatting is required when working with complex data types, such as sounds, videos, or images.

In this "information age," production of accurate, relevant, and timely information is the key to good decision making. In turn, good decision making is the key to business survival in a global market. We are now said to be entering the "knowledge age."² Data are the foundation of information, which is the bedrock of **knowledge**—that is, the body of information and facts about a specific subject. Knowledge implies familiarity, awareness, and understanding of information as it applies to an environment. A key characteristic of knowledge is that "new" knowledge can be derived from "old" knowledge.

² Peter Drucker coined the phrase "knowledge worker" in 1959 in his book *Landmarks of Tomorrow*. In 1994, Esther Dyson, George Keyworth, and Dr. Alvin Toffler introduced the concept of the "knowledge age."

Let's summarize some key points:

- Data constitute the building blocks of information.
- Information is produced by processing data.
- Information is used to reveal the meaning of data.
- Accurate, relevant, and timely information is the key to good decision making.
- Good decision making is the key to organizational survival in a global environment.

Timely and useful information requires accurate data. Such data must be properly generated and stored in a format that is easy to access and process. And, like any basic resource, the data environment must be managed carefully.

Data management is a discipline that focuses on the proper generation, storage, and retrieval of data. Given the crucial role that data play, it should not surprise you that data management is a core activity for any business, government agency, service organization, or charity.

1.3 INTRODUCING THE DATABASE

Efficient data management typically requires the use of a computer database. A **database** is a shared, integrated computer structure that stores a collection of:

- End-user data—that is, raw facts of interest to the end user.
- **Metadata**, or data about data, through which the end-user data are integrated and managed.

The metadata describe the data characteristics and the set of relationships that links the data found within the database. For example, the metadata component stores information such as the name of each data element, the type of values (numeric, dates, or text) stored on each data element, and whether the data element can be left empty. The metadata provide information that complements and expands the value and use of the data. In short, metadata present a more complete picture of the data in the database. Given the characteristics of metadata, you might hear a database described as a “collection of *self-describing* data.”

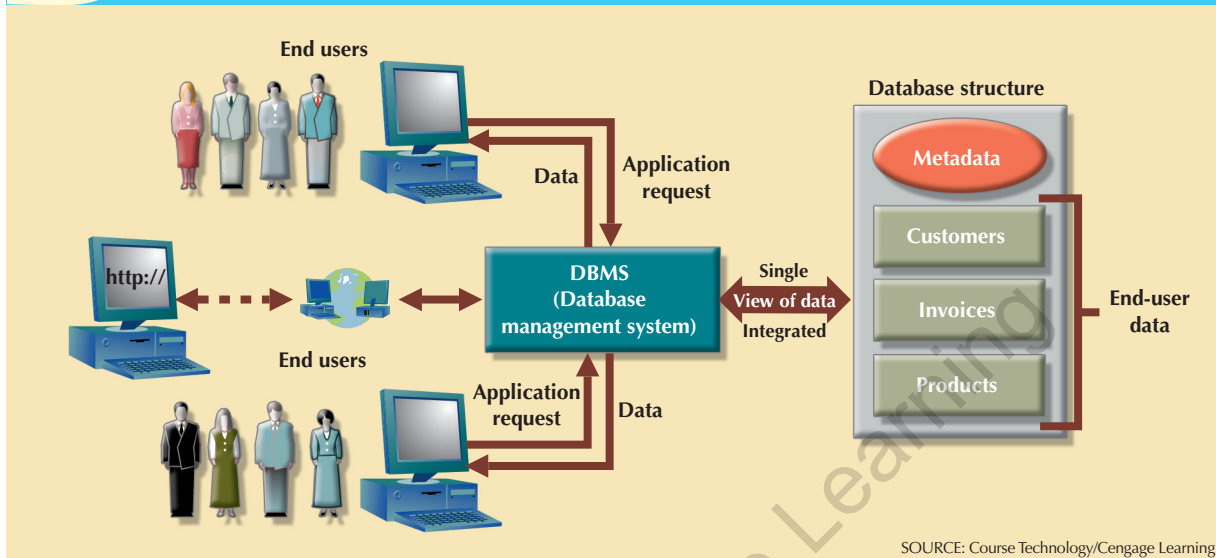
A **database management system (DBMS)** is a collection of programs that manages the database structure and controls access to the data stored in the database. In a sense, a database resembles a very well-organized electronic filing cabinet in which powerful software (the DBMS) helps manage the cabinet's contents.

1.3.1 ROLE AND ADVANTAGES OF THE DBMS

The DBMS serves as the intermediary between the user and the database. The database structure itself is stored as a collection of files, and the only way to access the data in those files is through the DBMS. Figure 1.2 emphasizes the point that the DBMS presents the end user (or application program) with a single, integrated view of the data in the database. The DBMS receives all application requests and translates them into the complex operations required to fulfill those requests. The DBMS hides much of the database's internal complexity from the application programs and users. The application program might be written by a programmer using a programming language such as Visual Basic.NET, Java, or C#, or it might be created through a DBMS utility program.

Having a DBMS between the end user's applications and the database offers some important advantages. First, the DBMS enables the data in the database *to be shared* among multiple applications or users. Second, the DBMS *integrates* the many different users' views of the data into a single all-encompassing data repository.

FIGURE 1.2 The DBMS manages the interaction between the end user and the database



Because data are the crucial raw material from which information is derived, you must have a good method to manage such data. As you will discover in this book, the DBMS helps make data management more efficient and effective. In particular, a DBMS provides advantages such as:

- *Improved data sharing.* The DBMS helps create an environment in which end users have better access to more and better-managed data. Such access makes it possible for end users to respond quickly to changes in their environment.
- *Improved data security.* The more users access the data, the greater the risks of data security breaches. Corporations invest considerable amounts of time, effort, and money to ensure that corporate data are used properly. A DBMS provides a framework for better enforcement of data privacy and security policies.
- *Better data integration.* Wider access to well-managed data promotes an integrated view of the organization's operations and a clearer view of the big picture. It becomes much easier to see how actions in one segment of the company affect other segments.
- *Minimized data inconsistency.* **Data inconsistency** exists when different versions of the same data appear in different places. For example, data inconsistency exists when a company's sales department stores a sales representative's name as Bill Brown and the company's personnel department stores that same person's name as *William G. Brown*, or when the company's regional sales office shows the price of a product as \$45.95 and its national sales office shows the same product's price as \$43.95. The probability of data inconsistency is greatly reduced in a properly designed database.
- *Improved data access.* The DBMS makes it possible to produce quick answers to ad hoc queries. From a database perspective, a **query** is a specific request issued to the DBMS for data manipulation—for example, to read or update the data. Simply put, a query is a question, and an **ad hoc query** is a spur-of-the-moment question. The DBMS sends back an answer (called the **query result set**) to the application. For example, when dealing with large amounts of sales data, end users might want quick answers to questions (ad hoc queries). Some examples include:
 - What was the dollar volume of sales by product during the past six months?
 - What is the sales bonus figure for each of our salespeople during the past three months?
 - How many of our customers have credit balances of \$3,000 or more?

- *Improved decision making.* Better-managed data and improved data access make it possible to generate better-quality information, on which better decisions are based. The quality of the information generated depends on the quality of the underlying data. **Data quality** is a comprehensive approach to promoting the accuracy, validity, and timeliness of the data. While the DBMS does not guarantee data quality, it provides a framework to facilitate data quality initiatives. Data quality concepts will be covered in more detail in Chapter 15, Database Administration and Security.
- *Increased end-user productivity.* The availability of data, combined with the tools that transform data into usable information, empowers end users to make quick, informed decisions that can make the difference between success and failure in the global economy.

The advantages of using a DBMS are not limited to the few just listed. In fact, you will discover many more advantages as you learn more about the technical details of databases and their proper design.

1.3.2 TYPES OF DATABASES

A DBMS can be used to build many different types of databases. Each database stores a particular collection of data and is used for a specific purpose. Over the years, as technology and innovative uses of databases have evolved, different methods have been used to classify databases. For example, databases can be classified by the number of users supported, where the data are located, the type of data stored, the intended data usage, and the degree to which the data are structured.

The number of users determines whether the database is classified as single-user or multiuser. A **single-user database** supports only one user at a time. In other words, if user A is using the database, users B and C must wait until user A is done. A single-user database that runs on a personal computer is called a **desktop database**. In contrast, a **multiuser database** supports multiple users at the same time. When the multiuser database supports a relatively small number of users (usually fewer than 50) or a specific department within an organization, it is called a **workgroup database**. When the database is used by the entire organization and supports many users (more than 50, usually hundreds) across many departments, the database is known as an **enterprise database**.

Location might also be used to classify the database. For example, a database that supports data located at a single site is called a **centralized database**. A database that supports data distributed across several different sites is called a **distributed database**. The extent to which a database can be distributed and the way in which such distribution is managed are addressed in detail in Chapter 12, Distributed Database Management Systems.

In some contexts, such as research environments, a popular way of classifying databases is according to the type of data stored in them. Using this criterion, databases are grouped into two categories: general-purpose and discipline-specific databases. **General-purpose databases** contain a wide variety of data used in multiple disciplines—for example, a census database that contains general demographic data, and the LexisNexis and ProQuest databases that contain newspaper, magazine, and journal articles for a variety of topics. **Discipline-specific databases** contain data focused on specific subject areas. The data in this type of database are used mainly for academic or research purposes within a small set of disciplines. Examples of discipline-specific databases include financial data stored in databases such as CompuStat or CRSP, geographic information system (GIS) databases that store geospatial and other related data, and medical databases that store confidential medical history data.

The most popular way of classifying databases today, however, is based on how they will be used and on the time sensitivity of the information gathered from them. For example, transactions such as product or service sales, payments, and supply purchases reflect critical day-to-day operations. Such transactions must be recorded accurately and immediately. A database that is designed primarily to support a company's day-to-day operations is classified as an **operational database**, also known as an **online transaction processing (OLTP)**, **transactional**, or **production database**. In contrast, **analytical databases** focus primarily on storing historical data and business metrics used exclusively for tactical or strategic decision making. Such analysis typically requires extensive "data massaging" (data

manipulation) to produce information on which to base pricing decisions, sales forecasts, market strategies, and so on. Analytical databases allow the end user to perform advanced data analysis of business data using sophisticated tools.

Typically, analytical databases comprise two main components: a data warehouse and an online analytical processing (OLAP) front end. The **data warehouse** is a specialized database that stores data in a format optimized for decision support. The data warehouse contains historical data obtained from the operational databases as well as data from other external sources. **Online analytical processing (OLAP)** is a set of tools that work together to provide an advanced data analysis environment for retrieving, processing, and modeling data from the data warehouse. In recent times, this area of database application has grown in importance and usage, to the point that it has evolved into its own discipline: business intelligence. The term **business intelligence** describes a comprehensive approach to capture and process business data with the purpose of generating information to support business decision making. Chapter 13, Business Intelligence and Data Warehouses, covers this topic in detail.

Databases can also be classified to reflect the degree to which the data are structured. **Unstructured data** are data that exist in their original (raw) state—that is, in the format in which they were collected. Therefore, unstructured data exist in a format that does not lend itself to the processing that yields information. **Structured data** are the result of formatting unstructured data to facilitate storage, use, and the generation of information. You apply structure (format) based on the type of processing that you intend to perform on the data. Some data might not be ready (unstructured) for some types of processing, but they might be ready (structured) for other types of processing. For example, the data value 37890 might refer to a zip code, a sales value, or a product code. If this value represents a zip code or a product code and is stored as text, you cannot perform mathematical computations with it. On the other hand, if this value represents a sales transaction, it must be formatted as numeric.

To further illustrate the concept of structure, imagine a stack of printed paper invoices. If you want to merely store these invoices as images for future retrieval and display, you can scan them and save them in a graphic format. On the other hand, if you want to derive information such as monthly totals and average sales, such graphic storage would not be useful. Instead, you could store the invoice data in a (structured) spreadsheet format so that you can perform the requisite computations. Actually, most data you encounter are best classified as semistructured. **Semistructured data** have already been processed to some extent. For example, if you look at a typical Web page, the data are presented in a prearranged format to convey some information. The database types mentioned thus far focus on the storage and management of highly structured data. However, corporations are not limited to the use of structured data. They also use semistructured and unstructured data. Just think of the valuable information that can be found on company e-mails, memos, and documents such as procedures, rules, and Web pages. Unstructured and semistructured data storage and management needs are being addressed through a new generation of databases known as XML databases. **Extensible Markup Language (XML)** is a special language used to represent and manipulate data elements in a textual format. An **XML database** supports the storage and management of semistructured XML data.

Table 1.1 compares the features of several well-known database management systems.

TABLE 1.1 Types of Databases

PRODUCT	NUMBER OF USERS			DATA LOCATION		DATA USAGE		XML
	SINGLE USER	MULTIUSER		CENTRALIZED	DISTRIBUTED	OPERATIONAL	ANALYTICAL	
		WORKGROUP	ENTERPRISE					
MS Access	X	X		X		X		
MS SQL Server	X ³	X	X	X	X	X	X	X
IBM DB2	X ³	X	X	X	X	X	X	X
MySQL	X	X	X	X	X	X	X	X
Oracle RDBMS	X ³	X	X	X	X	X	X	X

³ Vendor offers single-user/personal DBMS version.

With the emergence of the World Wide Web and Internet-based technologies as the basis for the new “social media” generation, great amounts of data are being stored and analyzed. **Social media** refers to Web and mobile technologies that enable “anywhere, anytime, always on” human interactions. Websites such as Google, Facebook, MySpace, Twitter, and LinkedIn capture vast amounts of data about end users and consumers. These data grow exponentially and require the use of specialized database systems. Over the past few years, this new breed of specialized database has grown in sophistication and widespread usage. Currently, this new type of database is known as a NoSQL database. The term **NoSQL** (Not only SQL) is generally used to describe a new generation of database management systems that is not based on the traditional relational database model. You will learn more about this type of system in Chapter 2, Data Models.

1.4 WHY DATABASE DESIGN IS IMPORTANT

Database design refers to the activities that focus on the design of the database structure that will be used to store and manage end-user data. A database that meets all user requirements does not just happen; its structure must be designed carefully. In fact, database design is such a crucial aspect of working with databases that most of this book is dedicated to the development of good database design techniques. Even a good DBMS will perform poorly with a badly designed database.

Data is one of an organization’s most valuable assets. Data on customers, employees, orders, and receipts are all vital to the existence of a company. Tracking key growth and performance indicators are also vital to strategic and tactical plans to ensure future success; therefore, an organization’s data must not be handled lightly or carelessly. Thorough planning to ensure that data are properly used and leveraged to give the company the most benefit is just as important as proper financial planning to ensure that the company gets the best use from its financial resources.

Because current-generation DBMSs are easy to use, an unfortunate side effect is that many computer-savvy business users gain a false sense of confidence in their ability to build a functional database. These users can effectively navigate the creation of database objects, but without the proper understanding of database design, they tend to produce flawed, overly simplified structures that prevent the system from correctly storing data that corresponds to business realities, which produces incomplete or erroneous results when the data are retrieved. Consider the data shown in Figure 1.3, which illustrates the storage of data about employees and skills. Some employees have not passed a certification test in any skill, while others have been certified in several skills. Some certified skills are stored for several employees, while other skills have no employees that hold those certifications.

FIGURE 1.3 Employee skill certification in a poor design

ID	EmployeeNumber	EmployeeName	Skill1	Skill1Date	Skill2	Skill2Date	Skill3	Skill3Date
1	02345	Johnny Jones	Basic Database Management	2/14/1993	Advanced Database Management	2/14/1993	Basic Web Design	8/9/1999
2	08273	Marco Bienz	Basic Web Design	3/8/2005	Advanced Process Modeling	8/19/2008		
3	06234	Jasmine Patel	Basic Web Design	8/10/2003	Advanced C# programming	8/10/2003	Basic DB manipulation	1/29/2008
4	03373	Franklin Johnson, Jr.	Advanced Spreadsheets	6/20/2007				
5	13567	Almond, Robert	Basic Process Modeling	9/30/2010	Basic Database Design	5/23/2011		
6	10282	Richardson, Amanda						
7	09382	Jessica Johnson	Basic DB Design	8/2/2008	Basic Database Manipulation	8/2/2008	Advanced DB Manipulation	5/1/2009
8	14311	Duong, Lee	Basic Web Design	9/1/2012				
9			Master Database Programming					
10			Basic Spreadsheets					
11	09002	Ben Joiner	Advanced Spreadsheets	5/16/2009	Basic Web Design	5/16/2009		
12	13383	Raymond F. Matthews	Basic C# Programming	3/12/2010				
13	09283	Chavez, Juan						
14	04893	Patricia Richards	Advanced Database Management	6/11/2002	Advanced Database Manipulation	9/20/2008		
15	13932	Lee, Megan						

SOURCE: Course Technology/Cengage Learning

Based on this storage of the data, notice the following problems.

- It would be difficult, if not impossible, to produce an alphabetical listing of employees based on their last names.
- To determine how many employees are certified in Basic Database Manipulation, you would need a program that counts the number of those certifications recorded in Skill1 and places it in a variable. Then the count of those certifications in Skill2 could be calculated and added to the variable. Finally, the count of those certifications in Skill3 could be calculated and added to the variable to produce the total.
- If you redundantly store the name of a skill with each employee who is certified in that skill, you run the risk of spelling the name differently for different employees. For example, the skill *Basic Database Manipulation* is also entered as *Basic DB Manipulation* for at least one employee in Figure 1.3, which makes it difficult to get an accurate count of employees who have the certification.
- The structure of the database will have to be changed by adding more columns to the table when an employee is certified in a fourth skill. It will have to be modified again if an employee is certified in a fifth skill.

Contrast this poor design with that shown in Figure 1.4, where the design has been improved by decomposing the data into three related tables. These tables contain all of the same data that were represented in Figure 1.3, but they are structured so that you can easily manipulate the data to view it in different ways and answer simple questions.

FIGURE 1.4 Employee skill certifications in a good design

Database name: Ch01_Text

Table name: EMPLOYEE				
Employee_ID	Employee_FName	Employee_LName	Employee_HireDate	Employee_Title
02345	Johnny	Jones	2/14/1993	DBA
03373	Franklin	Johnson	3/15/2000	Purchasing Agent
04893	Patricia	Richards	6/11/2002	DBA
06234	Jasmine	Patel	8/10/2003	Programmer
08273	Marco	Bienz	7/28/2004	Analyst
09002	Ben	Joiner	5/20/2008	Clerk
09283	Juan	Chavez	7/4/2008	Clerk
09382	Jessica	Johnson	8/2/2008	Database Programmer
10282	Amanda	Richardson	4/11/2009	Clerk
13383	Raymond	Matthews	3/12/2010	Programmer
13567	Robert	Almond	9/30/2010	Analyst
13932	Megan	Lee	9/29/2011	Programmer
14311	Lee	Duong	9/1/2012	Programmer

Table name: CERTIFIED		
Employee_ID	Skill_ID	Certified_Date
02345	100	2/14/1993
02345	110	8/9/1999
02345	180	2/14/1993
03373	120	6/20/2007
04893	180	6/11/2002
04893	220	9/20/2008
06234	110	8/10/2003
06234	200	8/10/2003
06234	210	1/29/2008
08273	110	3/8/2005
08273	190	8/19/2008
09002	110	5/16/2009
09002	120	5/16/2009
09382	140	8/2/2008
09382	210	8/2/2008
09382	220	5/1/2009
13383	170	3/12/2010
13567	130	9/30/2010
13567	140	5/23/2011
14311	110	9/1/2012

Table name: SKILL		
Skill_ID	Skill_Name	Skill_Description
100	Basic Database Management	Create and manage database user accounts.
110	Basic Web Design	Create and maintain HTML and CSS documents.
120	Advanced Spreadsheets	Use of advanced functions, user-defined functions, and macroing.
130	Basic Process Modeling	Create core business process models using standard libraries.
140	Basic Database Design	Create simple data models.
150	Master Database Programming	Create integrated trigger and procedure packages for a distributed environment.
160	Basic Spreadsheets	Create single tab worksheets with basic formulas
170	Basic C# Programming	Create single-tier data aware modules.
180	Advanced Database Management	Manage Database Server Clusters.
190	Advanced Process Modeling	Evaluate and Redesign cross-functional internal and external business processes.
200	Advanced C# Programming	Create multi-tier applications using multi-threading
210	Basic Database Manipulation	Create simple data retrieval and manipulation statements in SQL.
220	Advanced Database Manipulation	Use of advanced data manipulation methods for multi-table inserts, set operations, and correlated subqueries.

SOURCE: Course Technology/Cengage Learning

With the improved structure in Figure 1.4, you can use simple commands in a standard data manipulation language to:

- Produce an alphabetical listing of employees by last name:

```
SELECT * FROM EMPLOYEE ORDER BY EMPLOYEE_LNAME;
```
- Determine how many employees are certified in Basic Database Manipulation:

```
SELECT Count(*)  
FROM SKILL JOIN CERTIFIED ON SKILL.SKILL_ID = CERTIFIED.SKILL_ID  
WHERE SKILL_NAME = 'Basic Database Manipulation';
```

You will learn more about these commands in Chapter 7, Introduction to Structured Query Language.

Note that because each skill name is stored only once, the names cannot be spelled or abbreviated differently for different employees. Also, the additional certification of an employee with a fourth or fifth skill does not require changes to the structure of the tables.

Proper database design requires the designer to identify precisely the database's expected use. Designing a transactional database emphasizes accurate and consistent data and operational speed. Designing a data warehouse database emphasizes the use of historical and aggregated data. Designing a database to be used in a centralized, single-user environment requires a different approach from that used in the design of a distributed, multiuser database. This book emphasizes the design of transactional, centralized, single-user, and multiuser databases. Chapters 12 and 13 also examine critical issues confronting the designer of distributed and data warehouse databases.

Designing appropriate data repositories of integrated information using the two-dimensional table structures found in most databases is a process of decomposition. The integrated data must be decomposed properly into its constituent parts, with each part stored in its own table. Further, the relationships between these tables must be carefully considered and implemented so the integrated view of the data can be re-created later as information for the end user. A well-designed database facilitates data management and generates accurate and valuable information. A poorly designed database is likely to become a breeding ground for difficult-to-trace errors that may lead to bad decision making—and bad decision making can lead to the failure of an organization. Database design is simply too important to be left to luck. That's why college students study database design, why organizations of all types and sizes send personnel to database design seminars, and why database design consultants often make an excellent living.

1.5 EVOLUTION OF FILE SYSTEM DATA PROCESSING

Understanding what a database is, what it does, and the proper way to use it can be clarified by considering what a database is not. A brief explanation of the evolution of file system data processing can be helpful in understanding the data access limitations that databases attempt to overcome. Understanding these limitations is relevant to database designers and developers because database technologies do not make these problems magically disappear—database technologies simply make it easier to create solutions that avoid these problems. Creating database designs that avoid the pitfalls of earlier systems requires that the designer understand these problems and how to avoid them; otherwise, the database technologies are no better (and are potentially even worse!) than the technologies and techniques they have replaced.

1.5.1 MANUAL FILE SYSTEMS

To be successful, an organization must develop systems for handling core business tasks. Historically, such systems were often manual, paper-and-pencil systems. The papers within these systems were organized to facilitate the expected use of the data. Typically, this was accomplished through a system of file folders and filing cabinets. As long as a collection of data was relatively small and an organization's business users had few reporting requirements, the manual system served its role well as a data repository. However, as organizations grew and as reporting requirements became more complex, keeping track of data in a manual file system became more difficult. Therefore, companies looked to computer technology for help.

1.5.2 COMPUTERIZED FILE SYSTEMS

Generating reports from manual file systems was slow and cumbersome. In fact, some business managers faced government-imposed reporting requirements that led to weeks of intensive effort each quarter, even when a well-designed manual system was used. Therefore, a **data processing (DP) specialist** was hired to create a computer-based system that would track data and produce required reports.

Initially, the computer files within the file system were similar to the manual files. A simple example of a customer data file for a small insurance company is shown in Figure 1.5. (You will discover later that the file structure shown in Figure 1.5, although typically found in early file systems, is unsatisfactory for a database.)

The description of computer files requires a specialized vocabulary. Every discipline develops its own terminology to enable its practitioners to communicate clearly. The basic file vocabulary shown in Table 1.2 will help you to understand subsequent discussions more easily.

FIGURE 1.5 Contents of the CUSTOMER file

C_NAME	C_PHONE	C_ADDRESS	C_ZIP	A_NAME	A_PHONE	TP	AMT	REN
Alfred A. Ramas	615-844-2573	218 Fork Rd., Babs, TN	36123	Leah F. Hahn	615-882-1244	T1	100.00	05-Apr-2012
Leona K. Dunne	713-894-1238	Box 12A, Fox, KY	25246	Alex B. Alby	713-228-1249	T1	250.00	16-Jun-2012
Kathy W. Smith	615-894-2285	125 Oak Ln, Babs, TN	36123	Leah F. Hahn	615-882-2144	S2	150.00	29-Jan-2013
Paul F. Olowski	615-894-2180	217 Lee Ln., Babs, TN	36123	Leah F. Hahn	615-882-1244	S1	300.00	14-Oct-2012
Myron Orlando	615-222-1672	Box 111, New, TN	36155	Alex B. Alby	713-228-1249	T1	100.00	28-Dec-2012
Amy B. O'Brian	713-442-3381	387 Troll Dr., Fox, KY	25246	John T. Okon	615-123-5589	T2	850.00	22-Sep-2012
James G. Brown	615-297-1228	21 Tye Rd., Nash, TN	37118	Leah F. Hahn	615-882-1244	S1	120.00	25-Mar-2013
George Williams	615-290-2556	155 Maple, Nash, TN	37119	John T. Okon	615-123-5589	S1	250.00	17-Jul-2012
Anne G. Farriss	713-382-7185	2119 Elm, Crew, KY	25432	Alex B. Alby	713-228-1249	T2	100.00	03-Dec-2012
Olette K. Smith	615-297-3809	2782 Main, Nash, TN	37118	John T. Okon	615-123-5589	S2	500.00	14-Mar-2013

C_NAME = Customer name

C_PHONE = Customer phone

C_ADDRESS = Customer address

C_ZIP = Customer zip code

A_NAME = Agent name

A_PHONE = Agent phone

TP = Insurance type

AMT = Insurance policy amount, in thousands of \$

REN = Insurance renewal date

SOURCE: Course Technology/Cengage Learning



ONLINE CONTENT

The databases used in each chapter are available at www.cengagebrain.com. Throughout the book, Online Content boxes highlight material related to chapter content on the Website.

TABLE 1.2 Basic File Terminology

TERM	DEFINITION
Data	Raw facts, such as a telephone number, a birth date, a customer name, and a year-to-date (YTD) sales value. Data have little meaning unless they have been organized in some logical manner.
Field	A character or group of characters (alphabetic or numeric) that has a specific meaning. A field is used to define and store data.
Record	A logically connected set of one or more fields that describes a person, place, or thing. For example, the fields that constitute a customer record might consist of the customer's name, address, phone number, date of birth, credit limit, and unpaid balance.
File	A collection of related records. For example, a file might contain data about the students currently enrolled at Gigantic University.

Using the proper file terminology in Table 1.2, you can identify the file components shown in Figure 1.5. The CUSTOMER file contains 10 records. Each record is composed of nine fields: C_NAME, C_PHONE, C_ADDRESS, C_ZIP, A_NAME, A_PHONE, TP, AMT, and REN. The 10 records are stored in a named file. Because the file in Figure 1.5 contains customer data for the insurance company, its filename is CUSTOMER.

When business users wanted data from the computerized file, they sent requests for the data to the DP specialist. For each request, the DP specialist had to create programs to retrieve the data from the file, manipulate it in whatever manner the user had requested, and present it as a printed report. If a request was for a report that had been run previously, the DP specialist could rerun the existing program and provide the printed results to the user. As other business users saw the new and innovative ways in which customer data were being reported, they wanted to be able to view their data in similar fashions. This generated more requests for the DP specialist to create more computerized files of other business data, which in turn meant that more data management programs had to be created, and more requests for reports. For example, the sales department at the insurance company created a file named SALES, which helped track daily sales efforts. The sales department's success was so obvious that the personnel department manager demanded access to the DP specialist to automate payroll processing and other personnel functions. Consequently, the DP specialist was asked to create the AGENT file shown in Figure 1.6. The data in the AGENT file were used to write checks, keep track of taxes paid, and summarize insurance coverage, among other tasks.

FIGURE 1.6 Contents of the AGENT file

A_NAME	A_PHONE	A_ADDRESS	ZIP	HIRED	YTD_PAY	YTD_FIT	YTD_FICA	YTD_SLS	DEP
Alex B. Alby	713-228-1249	123 Toll, Nash, TN	37119	01-Nov-2000	26566.24	6641.56	2125.30	132737.75	3
Leah F. Hahn	615-882-1244	334 Main, Fox, KY	25246	23-May-1986	32213.78	8053.44	2577.10	138967.35	0
John T. Okon	615-123-5589	452 Elm, New, TN	36155	15-Jun-2005	23198.29	5799.57	1855.86	127093.45	2

A_NAME = Agent name
A_PHONE = Agent phone
A_ADDRESS = Agent address
ZIP = Agent zip code
HIRED = Agent date of hire

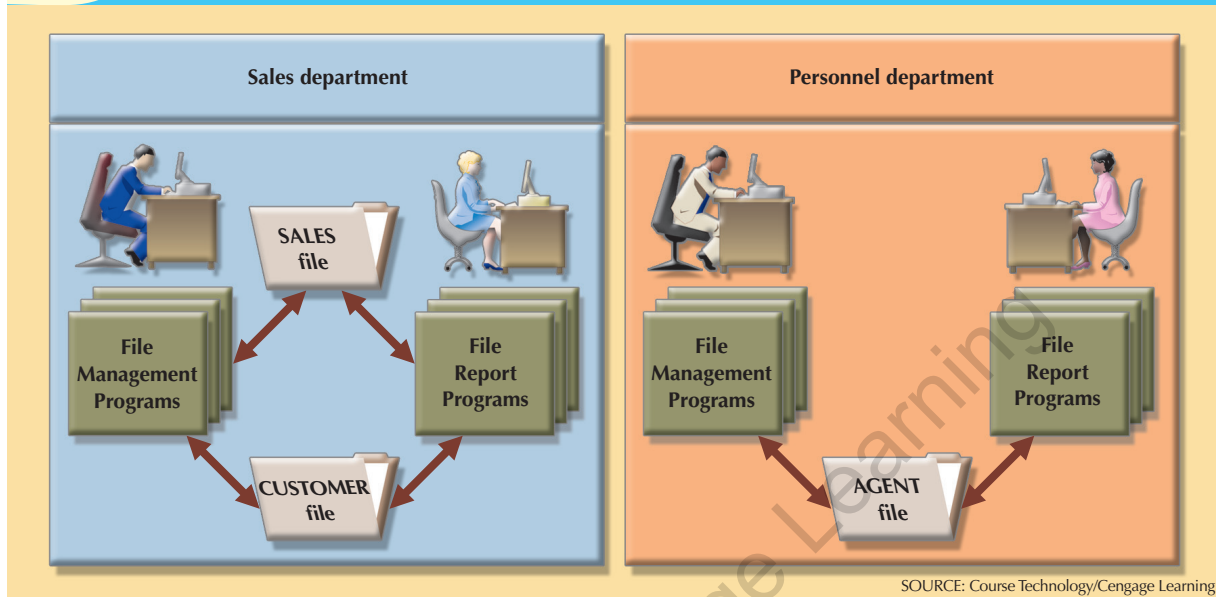
YTD_PAY = Year-to-date pay
YTD_FIT = Year-to-date federal income tax paid
YTD_FICA = Year-to-date Social Security taxes paid
YTD_SLS = Year-to-date sales
DEP = Number of dependents

SOURCE: Course Technology/Cengage Learning

As more and more computerized files were developed, the problems with this type of file system became apparent. While these problems are explored in detail in the next section, the problems basically centered on having many data files that contained related—often overlapping—data with no means of controlling or managing the data consistently across all of the files. As shown in Figure 1.7, each file in the system used its own application program to store, retrieve, and modify data. Also, each file was owned by the individual or the department that commissioned its creation.

The advent of computer files to store company data was significant; it not only established a landmark in the use of computer technologies, it also represented a huge step forward in a business's ability to process data. Previously, users had direct, hands-on access to all of the business data. But they didn't have the tools to convert those data into the information they needed. The creation of computerized file systems gave them improved tools for manipulating the company data that allowed them to create new information. However, it had the additional effect of introducing a schism between the end users and their data. The desire to close the gap between the end users and the data influenced the development of many types of computer technologies, system designs, and uses (and misuses) of many

FIGURE 1.7 A simple file system



technologies and techniques. However, such developments also created a split between the ways DP specialists and end users viewed the data.

- From the DP specialist's perspective, the computer files within the file system were created to be similar to the manual files. Data management programs were created to add to, update, and delete data from the file.
- From the end user's perspective, the systems separated the users from the data. As the users' competitive environment pushed them to make more and more decisions in less time, users became frustrated by the delay between conceiving of a new way to create information from the data and the point when the DP specialist actually created the programs to generate that information.

1.5.3 FILE SYSTEM REDUX: MODERN END-USER PRODUCTIVITY TOOLS

The users' desire for direct, hands-on access to data helped to fuel the adoption of personal computers for business use. Although not directly related to file system evolution, the ubiquitous use of personal productivity tools can introduce the same problems as the old file systems.

Personal computer spreadsheet programs such as Microsoft Excel are widely used by business users, and they allow the user to enter data in a series of rows and columns so the data can be manipulated using a wide range of functions. The popularity of spreadsheet applications has enabled users to conduct sophisticated data analysis that has greatly enhanced their ability to understand the data and make better decisions. Unfortunately, as in the old adage "When the only tool you have is a hammer, every problem looks like a nail," users have become so adept at working with spreadsheets, they tend to use them to complete tasks for which spreadsheets are not appropriate.

A common misuse of spreadsheets is as a substitute for a database. Interestingly, end users often take the limited data to which they have direct access and place it in a spreadsheet format similar to that of the traditional, manual data storage systems—which is precisely what the early DP specialists did when creating computerized data files. Due to the large number of users with spreadsheets, each making separate copies of the data, the resulting "file system" of spreadsheets suffers from the same problems as the file systems created by the early DP specialists, which are outlined in the next section.

1.6 PROBLEMS WITH FILE SYSTEM DATA PROCESSING

The file system method of organizing and managing data was a definite improvement over the manual system, and the file system served a useful purpose in data management for over two decades—a very long time in the computer era. Nonetheless, many problems and limitations became evident in this approach. A critique of the file system method serves two major purposes:

- Understanding the shortcomings of the file system enables you to understand the development of modern databases.
- Many of the problems are not unique to file systems. Failure to understand such problems is likely to lead to their duplication in a database environment, even though database technology makes it easy to avoid them.

The following problems associated with file systems, whether created by DP specialists or through a series of spreadsheets, severely challenge the types of information that can be created from the data as well as the accuracy of the information:

- *Lengthy development times.* The first and most glaring problem with the file system approach is that even the simplest data-retrieval task requires extensive programming. With the older file systems, programmers had to specify what must be done and how to do it. As you will learn in upcoming chapters, modern databases use a nonprocedural data manipulation language that allows the user to specify what must be done without specifying how.
- *Difficulty of getting quick answers.* The need to write programs to produce even the simplest reports makes ad hoc queries impossible. Harried DP specialists who work with mature file systems often receive numerous requests for new reports. They are often forced to say that the report will be ready “next week” or even “next month.” If you need the information now, getting it next week or next month will not serve your information needs.
- *Complex system administration.* System administration becomes more difficult as the number of files in the system expands. Even a simple file system with a few files requires creating and maintaining several file management programs. Each file must have its own file management programs that allow the user to add, modify, and delete records; to list the file contents; and to generate reports. Because ad hoc queries are not possible, the file reporting programs can multiply quickly. The problem is compounded by the fact that each department in the organization “owns” its data by creating its own files.
- *Lack of security and limited data sharing.* Another fault of a file system data repository is a lack of security and limited data sharing. Data sharing and security are closely related. Sharing data among multiple geographically dispersed users introduces a lot of security risks. In terms of spreadsheet data, while many spreadsheet programs provide rudimentary security options, they are not always used, and even when they are, they are insufficient for robust data sharing among users. In terms of creating data management and reporting programs, security and data-sharing features are difficult to program and consequently are often omitted from a file system environment. Such features include effective password protection, the ability to lock out parts of files or parts of the system itself, and other measures designed to safeguard data confidentiality. Even when an attempt is made to improve system and data security, the security devices tend to be limited in scope and effectiveness.
- *Extensive programming.* Making changes to an existing file structure can be difficult in a file system environment. For example, changing just one field in the original CUSTOMER file would require a program that:
 1. Reads a record from the original file.
 2. Transforms the original data to conform to the new structure’s storage requirements.
 3. Writes the transformed data into the new file structure.
 4. Repeats the preceding steps for each record in the original file.

In fact, any change to a file structure, no matter how minor, forces modifications in all of the programs that use the data in that file. Modifications are likely to produce errors (bugs), and additional time is spent using a debugging process to find those errors. Those limitations, in turn, lead to problems of structural and data dependence.

1.6.1 STRUCTURAL AND DATA DEPENDENCE

A file system exhibits **structural dependence**, which means that access to a file is dependent on its structure. For example, adding a customer date-of-birth field to the CUSTOMER file shown in Figure 1.5 would require the four steps described in the previous section. Given this change, none of the previous programs will work with the new CUSTOMER file structure. Therefore, all of the file system programs must be modified to conform to the new file structure. In short, because the file system application programs are affected by changes in the file structure, they exhibit structural dependence. Conversely, **structural independence** exists when you can change the file structure without affecting the application's ability to access the data.

Even changes in the characteristics of data, such as changing a field from integer to decimal, require changes in all the programs that access the file. Because all data access programs are subject to change when any of the file's data storage characteristics change (that is, changing the data type), the file system is said to exhibit **data dependence**. Conversely, **data independence** exists when you can change the data storage characteristics without affecting the program's ability to access the data.

The practical significance of data dependence is the difference between the **logical data format** (how the human being views the data) and the **physical data format** (how the computer must work with the data). Any program that accesses a file system's file must tell the computer not only what to do but how to do it. Consequently, each program must contain lines that specify the opening of a specific file type, its record specification, and its field definitions. Data dependence makes the file system extremely cumbersome from the point of view of a programmer and database manager.

1.6.2 DATA REDUNDANCY

The file system's structure makes it difficult to combine data from multiple sources, and its lack of security renders the file system vulnerable to security breaches. The organizational structure promotes the storage of the same basic data in different locations. (Database professionals use the term **islands of information** for such scattered data locations.) The dispersion of data is exacerbated by the use of spreadsheets to store data. In a file system, the entire sales department would share access to the SALES data file through the data management and reporting programs created by the DP specialist. With the use of spreadsheets, each member of the sales department can create his or her own copy of the sales data. Because data stored in different locations will probably not be updated consistently, the islands of information often contain different versions of the same data. For example, in Figures 1.5 and 1.6, the agent names and phone numbers occur in both the CUSTOMER and the AGENT files. You only need one correct copy of the agent names and phone numbers. Having them occur in more than one place produces data redundancy. **Data redundancy** exists when the same data are stored unnecessarily at different places.

Uncontrolled data redundancy sets the stage for:

- *Poor data security.* Having multiple copies of data increases the chances for a copy of the data to be susceptible to unauthorized access. Chapter 15, Database Administration and Security, explores the issues and techniques associated with securing data.
- *Data inconsistency.* Data inconsistency exists when different and conflicting versions of the same data appear in different places. For example, suppose you change an agent's phone number in the AGENT file. If you forget to make the corresponding change in the CUSTOMER file, the files contain different data for the same agent. Reports will yield inconsistent results that depend on which version of the data is used.

NOTE

Data that display data inconsistency are also said to lack data integrity. **Data integrity** is defined as the condition in which all data in the database are consistent with real-world events and conditions. In other words, data integrity means that:

- Data are *accurate*—there are no data inconsistencies.
- Data are *verifiable*—the data will always yield consistent results.

Data-entry errors are more likely to occur when complex entries (such as 10-digit phone numbers) are made in several different files or recur frequently in one or more files. In fact, the CUSTOMER file shown in Figure 1.5 contains just such an entry error: the third record in the CUSTOMER file has a transposed digit in the agent's phone number (615-882-2144 rather than 615-882-1244).

It is possible to enter a nonexistent sales agent's name and phone number into the CUSTOMER file, but customers are not likely to be impressed if the insurance agency supplies the name and phone number of an agent who does not exist. Should the personnel manager allow a nonexistent agent to accrue bonuses and benefits? In fact, a data-entry error such as an incorrectly spelled name or an incorrect phone number yields the same kind of data integrity problems.

- **Data anomalies.** The dictionary defines *anomaly* as “an abnormality.” Ideally, a field value change should be made in only a single place. Data redundancy, however, fosters an abnormal condition by forcing field value changes in many different locations. Look at the CUSTOMER file in Figure 1.5. If agent Leah F. Hahn decides to get married and move, the agent name, address, and phone number are likely to change. Instead of making these changes in a single file (AGENT), you must also make the change each time that agent's name and phone number occur in the CUSTOMER file. You could be faced with the prospect of making hundreds of corrections, one for each of the customers served by that agent! The same problem occurs when an agent decides to quit. Each customer served by that agent must be assigned a new agent. Any change in any field value must be correctly made in many places to maintain data integrity. A **data anomaly** develops when not all of the required changes in the redundant data are made successfully. The data anomalies found in Figure 1.5 are commonly defined as follows:
 - **Update anomalies.** If agent Leah F. Hahn has a new phone number, it must be entered in each of the CUSTOMER file records in which Ms. Hahn's phone number is shown. In this case, only four changes must be made. In a large file system, such a change might occur in hundreds or even thousands of records. Clearly, the potential for data inconsistencies is great.
 - **Insertion anomalies.** If only the CUSTOMER file existed and you needed to add a new agent, you would also add a dummy customer data entry to reflect the new agent's addition. Again, the potential for creating data inconsistencies would be great.
 - **Deletion anomalies.** If you delete the customers Amy B. O'Brian, George Williams, and Olette K. Smith, you will also delete John T. Okon's agent data. Clearly, this is not desirable.

1.6.3 LACK OF DESIGN AND DATA-MODELING SKILLS

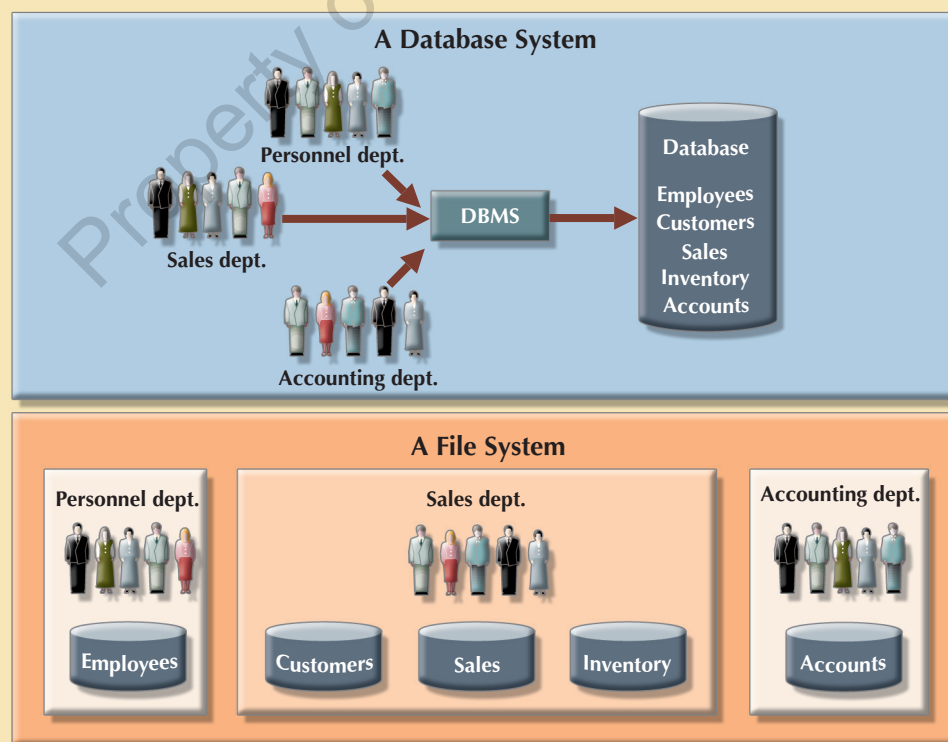
A new problem that has evolved with the use of personal productivity tools (such as spreadsheet and desktop databases) is that users typically lack proper design and data-modeling skills. People naturally have an integrated view of the data in their environment. For example, consider a student's class schedule. The schedule probably contains the student's identification number and name, class code, class description, class credit hours, class instructor name, the class meeting days and times, and the class room number. In the mind of the student, these various data items compose a single unit. If a student organization wanted to keep a record of the schedules of its members, an end user might make a spreadsheet to store the schedule information. Even if the student makes a foray into the realm of desktop databases, he or she is likely to create a structure composed of a single table that mimics the schedule structure. As you will learn in the coming chapters, forcing this type of integrated data into a single two-dimensional table structure is a poor data design that leads to a large degree of redundancy for several data items.

Data-modeling skills are also a vital part of the design process. It is important to document the design properly. Design documentation is necessary to facilitate communication among the database designer, the end user, and the developer. Data modeling, as introduced later in this text, is the most common method of documenting database designs. Using a standardized data-modeling technique ensures that the data model fulfills its role in facilitating communication. The data model also provides an invaluable resource when maintaining or modifying a database as business requirements change. The data designs created by end users are rarely documented and never with an appropriate standardized data-modeling technique. On a positive note, however, this book will help you develop the skills needed to design and model a successful database that avoids the problems listed earlier in this section.

1.7 DATABASE SYSTEMS

The problems inherent in file systems make using a database system very desirable. Unlike the file system, with its many separate and unrelated files, the database system consists of logically related data stored in a single logical data repository. (The “logical” label reflects the fact that the data repository appears to be a single unit to the end user, even though data might be physically distributed among multiple storage facilities and locations.) Because the database’s data repository is a single logical unit, the database represents a major change in the way end-user data are stored, accessed, and managed. The database’s DBMS, shown in Figure 1.8, provides numerous advantages over file system management, shown in Figure 1.7, by making it possible to eliminate most of the file system’s data inconsistency, data anomaly, data dependence, and structural dependence problems. Better yet, the current generation of DBMS software stores not only the data structures, but also the relationships between those structures and the access paths to those structures—all in a central location. The current generation of DBMS software also takes care of defining, storing, and managing all required access paths to those components.

FIGURE 1.8 Contrasting database and file systems



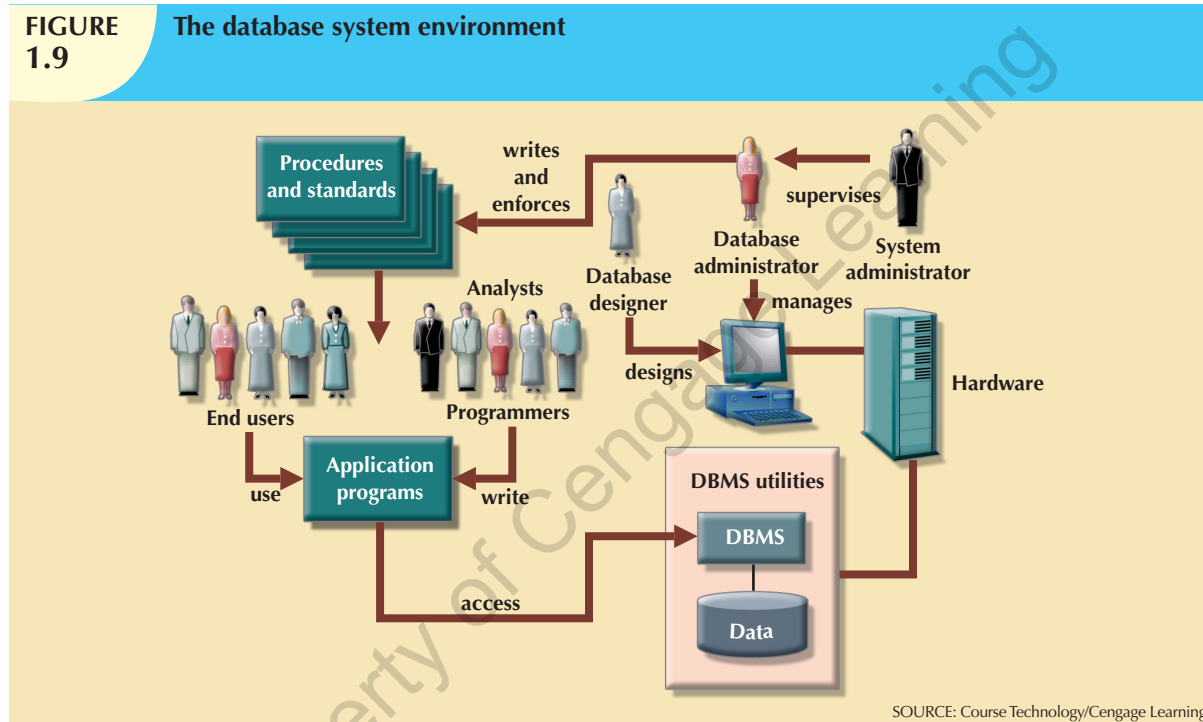
SOURCE: Course Technology/Cengage Learning

Remember that the DBMS is just one of several crucial components of a database system. The DBMS may even be referred to as the database system's heart. However, just as it takes more than a heart to make a human being function, it takes more than a DBMS to make a database system function. In the sections that follow, you'll learn what a database system is, what its components are, and how the DBMS fits into the picture.

1.7.1 THE DATABASE SYSTEM ENVIRONMENT

The term **database system** refers to an organization of components that define and regulate the collection, storage, management, and use of data within a database environment. From a general management point of view, the database system is composed of the five major parts shown in Figure 1.9: hardware, software, people, procedures, and data.

FIGURE 1.9 The database system environment



Let's take a closer look at the five components shown in Figure 1.9:

- **Hardware.** Hardware refers to all of the system's physical devices, including computers (PCs, workstations, servers, and supercomputers), storage devices, printers, network devices (hubs, switches, routers, fiber optics), and other devices (automated teller machines, ID readers, and so on).
- **Software.** Although the most readily identified software is the DBMS itself, three types of software are needed to make the database system function fully: operating system software, DBMS software, and application programs and utilities.
 - **Operating system software** manages all hardware components and makes it possible for all other software to run on the computers. Examples of operating system software include Microsoft Windows, Linux, Mac OS, UNIX, and MVS.
 - **DBMS software** manages the database within the database system. Some examples of DBMS software include Microsoft's SQL Server, Oracle Corporation's Oracle, Sun's MySQL, and IBM's DB2.
 - **Application programs and utility software** are used to access and manipulate data in the DBMS and to manage the computer environment in which data access and manipulation take place. Application programs are most commonly used to access data within the database to generate reports, tabulations, and other information to facilitate decision making. Utilities are the software tools used to help manage

the database system's computer components. For example, all of the major DBMS vendors now provide graphical user interfaces (GUIs) to help create database structures, control database access, and monitor database operations.

- *People.* This component includes all users of the database system. On the basis of primary job functions, five types of users can be identified in a database system: system administrators, database administrators, database designers, system analysts and programmers, and end users. Each user type, described below, performs both unique and complementary functions.
 - *System administrators* oversee the database system's general operations.
 - *Database administrators*, also known as DBAs, manage the DBMS and ensure that the database is functioning properly. The DBA's role is sufficiently important to warrant a detailed exploration in Chapter 15, Database Administration and Security.
 - *Database designers* design the database structure. They are, in effect, the database architects. If the database design is poor, even the best application programmers and the most dedicated DBAs cannot produce a useful database environment. Because organizations strive to optimize their data resources, the database designer's job description has expanded to cover new dimensions and growing responsibilities.
 - *System analysts and programmers* design and implement the application programs. They design and create the data-entry screens, reports, and procedures through which end users access and manipulate the database's data.
 - *End users* are the people who use the application programs to run the organization's daily operations. For example, sales clerks, supervisors, managers, and directors are all classified as end users. High-level end users employ the information obtained from the database to make tactical and strategic business decisions.
- *Procedures.* Procedures are the instructions and rules that govern the design and use of the database system. Procedures are a critical, although occasionally forgotten, component of the system. Procedures play an important role in a company because they enforce the standards by which business is conducted within the organization and with customers. Procedures also help to ensure that companies have an organized way to monitor and audit the data that enter the database and the information generated from those data.
- *Data.* The word *data* covers the collection of facts stored in the database. Because data are the raw material from which information is generated, determining what data to enter into the database and how to organize those data is a vital part of the database designer's job.

A database system adds a new dimension to an organization's management structure. The complexity of this managerial structure depends on the organization's size, its functions, and its corporate culture. Therefore, database systems can be created and managed at different levels of complexity and with varying adherence to precise standards. For example, compare a local movie rental system with a national insurance claims system. The movie rental system may be managed by two people, the hardware used is probably a single PC, the procedures are probably simple, and the data volume tends to be low. The national insurance claims system is likely to have at least one systems administrator, several full-time DBAs, and many designers and programmers; the hardware probably includes several servers at multiple locations throughout the United States; the procedures are likely to be numerous, complex, and rigorous; and the data volume tends to be high.

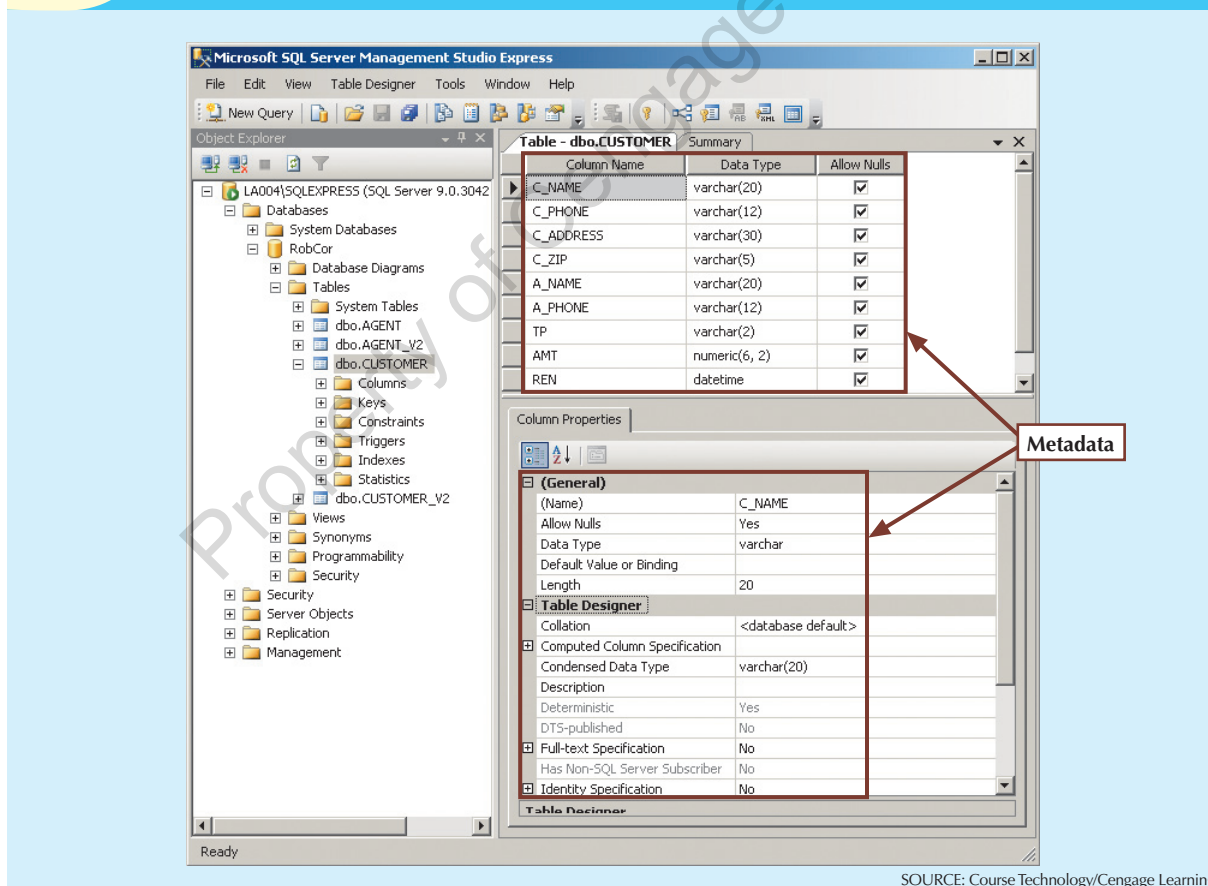
In addition to the different levels of database system complexity, managers must also take another important fact into account: database solutions must be cost-effective as well as tactically and strategically effective. Producing a million-dollar solution to a thousand-dollar problem is hardly an example of good database system selection or of good database design and management. Finally, the database technology already in use is likely to affect the selection of a database system.

1.7.2 DBMS FUNCTIONS

A DBMS performs several important functions that guarantee the integrity and consistency of the data in the database. Most of those functions are transparent to end users, and most can be achieved only through the use of a DBMS. They include data dictionary management, data storage management, data transformation and presentation, security management, multiuser access control, backup and recovery management, data integrity management, database access languages and application programming interfaces, and database communication interfaces. Each of these functions is explained below.

- *Data dictionary management.* The DBMS stores definitions of the data elements and their relationships (meta-data) in a data dictionary. In turn, all programs that access the data in the database work through the DBMS. The DBMS uses the **data dictionary** to look up the required data component structures and relationships, thus relieving you from having to code such complex relationships in each program. Additionally, any changes made in a database structure are automatically recorded in the data dictionary, thereby freeing you from having to modify all of the programs that access the changed structure. In other words, the DBMS provides data abstraction, and it removes structural and data dependence from the system. For example, Figure 1.10 shows how Microsoft SQL Server Express presents the data definition for the CUSTOMER table.

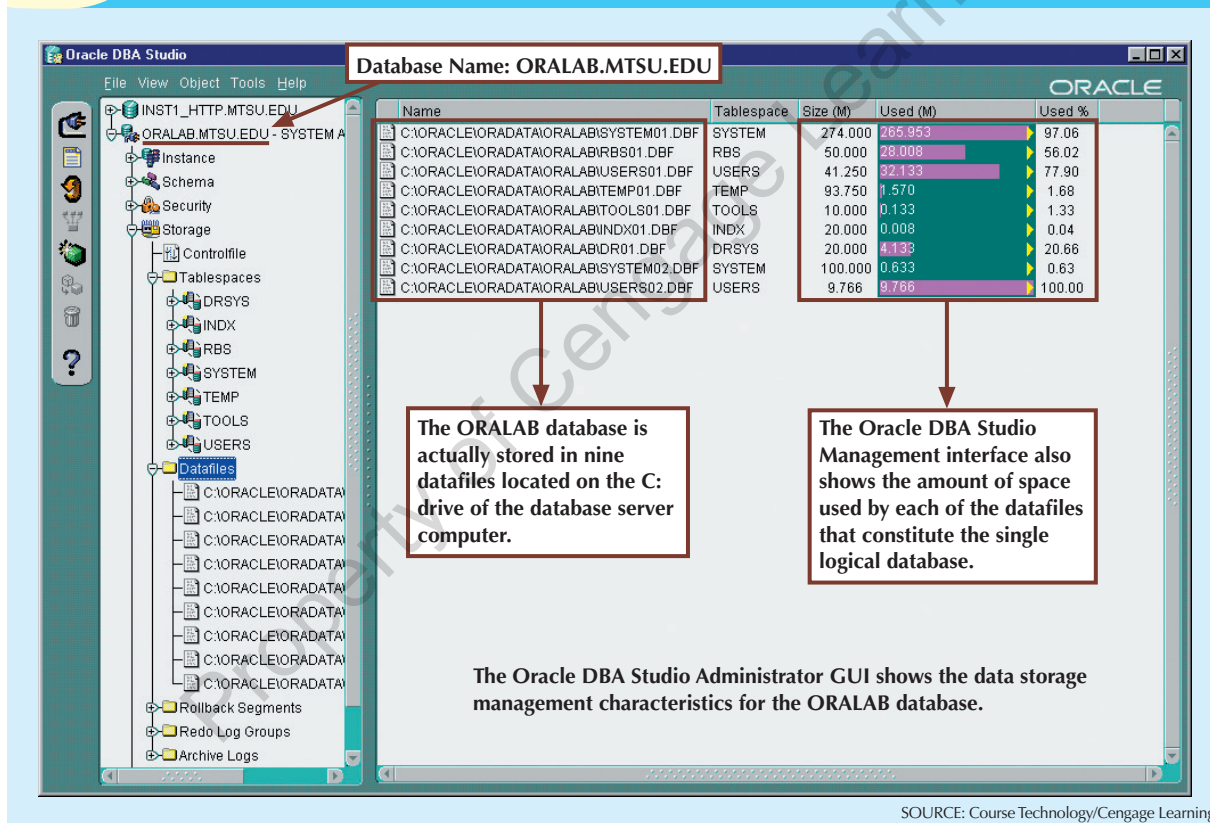
FIGURE 1.10 Illustrating metadata with Microsoft SQL Server Express



SOURCE: Course Technology/Cengage Learning

- Data storage management.** The DBMS creates and manages the complex structures required for data storage, thus relieving you from the difficult task of defining and programming the physical data characteristics. A modern DBMS provides storage not only for the data but for related data-entry forms or screen definitions, report definitions, data validation rules, procedural code, structures to handle video and picture formats, and so on. Data storage management is also important for database performance tuning. **Performance tuning** relates to the activities that make the database perform more efficiently in terms of storage and access speed. Although the user sees the database as a single data storage unit, the DBMS actually stores the database in multiple physical data files. (See Figure 1.11.) Such data files may even be stored on different storage media. Therefore, the DBMS doesn't have to wait for one disk request to finish before the next one starts. In other words, the DBMS can fulfill database requests concurrently. Data storage management and performance tuning issues are addressed in Chapter 11, Database Performance Tuning and Query Optimization.

FIGURE 1.11 Illustrating data storage management with Oracle



- Data transformation and presentation.** The DBMS transforms entered data to conform to required data structures. The DBMS relieves you of the chore of distinguishing between the logical data format and the physical data format. That is, the DBMS formats the physically retrieved data to make it conform to the user's logical expectations. For example, imagine an enterprise database used by a multinational company. An end user in England would expect to enter the date July 11, 2013, as "11/07/2013." In contrast, the same date would be entered in the United States as "07/11/2013." Regardless of the data presentation format, the DBMS must manage the date in the proper format for each country.
- Security management.** The DBMS creates a security system that enforces user security and data privacy. Security rules determine which users can access the database, which data items each user can access, and which data operations (read, add, delete, or modify) the user can perform. This is especially important in

multiuser database systems. Chapter 15, Database Administration and Security, examines data security and privacy issues in greater detail. All database users may be authenticated to the DBMS through a username and password or through biometric authentication such as a fingerprint scan. The DBMS uses this information to assign access privileges to various database components such as queries and reports.

- *Multiuser access control.* To provide data integrity and data consistency, the DBMS uses sophisticated algorithms to ensure that multiple users can access the database concurrently without compromising its integrity. Chapter 10, Transaction Management and Concurrency Control, covers the details of multiuser access control.
- *Backup and recovery management.* The DBMS provides backup and data recovery to ensure data safety and integrity. Current DBMS systems provide special utilities that allow the DBA to perform routine and special backup and restore procedures. Recovery management deals with the recovery of the database after a failure, such as a bad sector in the disk or a power failure. Such capability is critical to preserving the database's integrity. Chapter 15 covers backup and recovery issues.
- *Data integrity management.* The DBMS promotes and enforces integrity rules, thus minimizing data redundancy and maximizing data consistency. The data relationships stored in the data dictionary are used to enforce data integrity. Ensuring data integrity is especially important in transaction-oriented database systems. Data integrity and transaction management issues are addressed in Chapter 7, Introduction to Structured Query Language (SQL), and Chapter 10.
- *Database access languages and application programming interfaces.* The DBMS provides data access through a query language. A **query language** is a nonprocedural language—one that lets the user specify what must be done without having to specify how. **Structured Query Language (SQL)** is the de facto query language and data access standard supported by the majority of DBMS vendors. Chapter 7 and Chapter 8, Advanced SQL, address the use of SQL. The DBMS also provides application programming interfaces to procedural languages such as COBOL, C, Java, Visual Basic.NET, and C#. In addition, the DBMS provides administrative utilities used by the DBA and the database designer to create, implement, monitor, and maintain the database.
- *Database communication interfaces.* A current-generation DBMS accepts end-user requests via multiple, different network environments. For example, the DBMS might provide access to the database via the Internet through the use of Web browsers such as Mozilla Firefox or Microsoft Internet Explorer. In this environment, communications can be accomplished in several ways:
 - End users can generate answers to queries by filling in screen forms through their preferred Web browser.
 - The DBMS can automatically publish predefined reports on a Website.
 - The DBMS can connect to third-party systems to distribute information via e-mail or other productivity applications.

Database communication interfaces are examined in greater detail in Chapter 12, Distributed Database Management Systems; in Chapter 14, Database Connectivity and Web Technologies; and in Appendix I, Databases in Electronic Commerce. (Appendixes are available at www.cengagebrain.com.)

NOTE

Why a Spreadsheet Is Not a Database

While a spreadsheet allows for the creation of multiple tables, it does not support even the most basic database functionality such as support for self-documentation through metadata, enforcement of data types or domains to ensure consistency of data within a column, defined relationships among tables, or constraints to ensure consistency of data across related tables. Most users lack the necessary training to recognize the limitations of spreadsheets for these types of tasks.

1.7.3 MANAGING THE DATABASE SYSTEM: A SHIFT IN FOCUS

The introduction of a database system over the file system provides a framework in which strict procedures and standards can be enforced. Consequently, the role of the human component changes from an emphasis on programming (in the file system) to a focus on the broader aspects of managing the organization's data resources and on the administration of the complex database software itself.

The database system makes it possible to tackle far more sophisticated uses of the data resources, as long as the database is designed to make use of that power. The kinds of data structures created within the database and the extent of the relationships among them play a powerful role in determining the effectiveness of the database system.

Although the database system yields considerable advantages over previous data management approaches, database systems do carry significant disadvantages. For example:

- *Increased costs.* Database systems require sophisticated hardware and software and highly skilled personnel. The cost of maintaining the hardware, software, and personnel required to operate and manage a database system can be substantial. Training, licensing, and regulation compliance costs are often overlooked when database systems are implemented.
- *Management complexity.* Database systems interface with many different technologies and have a significant impact on a company's resources and culture. The changes introduced by the adoption of a database system must be properly managed to ensure that they help advance the company's objectives. Because database systems hold crucial company data that are accessed from multiple sources, security issues must be assessed constantly.
- *Maintaining currency.* To maximize the efficiency of the database system, you must keep your system current. Therefore, you must perform frequent updates and apply the latest patches and security measures to all components. Because database technology advances rapidly, personnel training costs tend to be significant.
- *Vendor dependence.* Given the heavy investment in technology and personnel training, companies might be reluctant to change database vendors. As a consequence, vendors are less likely to offer pricing point advantages to existing customers, and those customers might be limited in their choice of database system components.
- *Frequent upgrade/replacement cycles.* DBMS vendors frequently upgrade their products by adding new functionality. Such new features often come bundled in new upgrade versions of the software. Some of these versions require hardware upgrades. Not only do the upgrades themselves cost money, it also costs money to train database users and administrators to properly use and manage the new features.

Now that you know what a database and DBMS are, and why they are necessary, you are ready to begin developing your career as a database professional.

1.8 PREPARING FOR YOUR DATABASE PROFESSIONAL CAREER

In this chapter, you were introduced to the concepts of data, information, databases, and DBMSs. You also learned that, regardless of what type of database you use (OLTP or OLAP), or what type of database environment you are working in (for example, Oracle, Microsoft, or IBM), the success of a database system greatly depends on how well the database structure is designed.

Throughout this book, you will learn the building blocks that lay the foundation for your career as a database professional. Understanding these building blocks and developing the skills to use them effectively will prepare you to work with databases at many different levels within an organization. A small sample of such career opportunities is shown in Table 1.3.

TABLE 1.3 Database Career Opportunities

JOB TITLE	DESCRIPTION	SAMPLE SKILLS REQUIRED
Database developer	Creates and maintains database-based applications	Programming, database fundamentals, SQL
Database designer	Designs and maintains databases	Systems design, database design, SQL
Database administrator	Manages and maintains DBMS and databases	Database fundamentals, SQL, vendor courses
Database analyst	Develops databases for decision support reporting	SQL, query optimization, data warehouses
Database architect	Designs and implements database environments (conceptual, logical, and physical)	DBMS fundamentals, data modeling, SQL, hardware knowledge
Database consultant	Helps companies leverage database technologies to improve business processes and achieve specific goals	Database fundamentals, data modeling, database design, SQL, DBMS, hardware, vendor-specific technologies
Database security officer	Implements security policies for data administration	DBMS fundamentals, database administration, SQL, data security technologies

As you also learned in this chapter, database technologies are constantly evolving to address new challenges such as large databases, semistructured and unstructured data, increasing processing speed, and lowering costs. While database technologies can change quickly, the fundamental concepts and skills do not. It is our goal that after you learn the database essentials in this book, you will be ready to apply your knowledge and skills to work with traditional OLTP and OLAP systems as well as cutting-edge, complex database technologies such as:

- *Very Large Databases (VLDB).* Many vendors are addressing the need for databases that support large amounts of data, usually in the petabyte range. (A petabyte is more than 1,000 terabytes.) VLDB vendors include Oracle Exadata, IBM's Netezza, Greenplum, HP's Vertica, and Teradata. VLDB are now being overtaken in market interest by Big Data databases.
- *Big Data databases.* Products such as Cassandra (Facebook) and BigTable (Google) are using "columnar-database" technologies to support the needs of database applications that manage large amounts of "nontabular" data. See more about this topic in Chapter 2.
- *In-memory databases.* Most major database vendors also offer some type of in-memory database support to address the need for faster database processing. In-memory databases store most of their data in primary memory (RAM) rather than in slower secondary storage (hard disks). In-memory databases include IBM's solidDB and Oracle's TimesTen.
- *Cloud databases.* Companies can now use cloud database services to quickly add database systems to their environment while simultaneously lowering the total cost of ownership of a new DBMS. A cloud database offers all the advantages of a local DBMS, but instead of residing within your organization's network infrastructure, it resides on the Internet. See more about this topic in Chapter 14.

We address some of these topics in this book, but not all—no single book can cover the entire realm of database technologies. This book's primary focus is to help you learn database fundamentals, develop your database design skills, and master your SQL skills so you will have a head start in becoming a successful database professional. However, you first must learn about the tools at your disposal. In the next chapter, you will learn different approaches to data management and how these approaches influence your designs.

S U M M A R Y

- Data are raw facts. Information is the result of processing data to reveal their meaning. Accurate, relevant, and timely information is the key to good decision making, and good decision making is the key to organizational survival in a global environment.
- Data are usually stored in a database. To implement a database and to manage its contents, you need a database management system (DBMS). The DBMS serves as the intermediary between the user and the database. The database contains the data you have collected and “data about data,” known as metadata.
- Database design defines the database structure. A well-designed database facilitates data management and generates accurate and valuable information. A poorly designed database can lead to bad decision making, and bad decision making can lead to the failure of an organization.
- Databases can be classified according to the number of users supported, where the data are located, the type of data stored, the intended data usage, and the degree to which the data are structured.
- Databases evolved from manual and then computerized file systems. In a file system, data are stored in independent files, each requiring its own data management programs. Although this method of data management is largely outmoded, understanding its characteristics makes database design easier to comprehend.
- Some limitations of file system data management are that it requires extensive programming, system administration can be complex and difficult, making changes to existing structures is difficult, and security features are likely to be inadequate. Also, independent files tend to contain redundant data, leading to problems of structural and data dependence.
- Database management systems were developed to address the file system’s inherent weaknesses. Rather than depositing data in independent files, a DBMS presents the database to the end user as a single data repository. This arrangement promotes data sharing, thus eliminating the potential problem of islands of information. In addition, the DBMS enforces data integrity, eliminates redundancy, and promotes data security.

K E Y T E R M S

ad hoc query	database design	multiuser database
analytical database	database management system (DBMS)	NoSQL
business intelligence	database system	online analytical processing (OLAP)
centralized database	desktop database	online transaction processing (OLTP)
data	discipline-specific database	operational database
data anomaly	distributed database	performance tuning
data dependence	enterprise database	physical data format
data dictionary	Extensible Markup Language (XML)	production database
data inconsistency	field	query
data independence	file	query language
data integrity	general-purpose database	query result set
data management	information	record
data processing (DP) specialist	islands of information	semistructured data
data quality	knowledge	single-user database
data redundancy	logical data format	social media
data warehouse	metadata	structural dependence
database		structural independence

structured data
Structured Query Language (SQL)

transactional database
unstructured data

workgroup database
XML database



ONLINE CONTENT

Flashcards and crossword puzzles for key term practice are available at www.cengagebrain.com.

REVIEW QUESTIONS

1. Define each of the following terms:
 - a. data
 - b. field
 - c. record
 - d. file
2. What is data redundancy, and which characteristics of the file system can lead to it?
3. What is data independence, and why is it lacking in file systems?
4. What is a DBMS, and what are its functions?
5. What is structural independence, and why is it important?
6. Explain the differences among data, information, and a database.
7. What is the role of a DBMS, and what are its advantages? What are its disadvantages?
8. List and describe the different types of databases.
9. What are the main components of a database system?
10. What are metadata?
11. Explain why database design is important.
12. What are the potential costs of implementing a database system?
13. Use examples to compare and contrast unstructured and structured data. Which type is more prevalent in a typical business environment?
14. What are some basic database functions that a spreadsheet cannot perform?
15. What common problems does a collection of spreadsheets created by end users share with the typical file system?
16. Explain the significance of the loss of direct, hands-on access to business data that end users experienced with the advent of computerized data repositories.

P R O B L E M S



O N L I N E C O N T E N T

The file structures you see in this problem set are simulated in a Microsoft Access database named **Ch01_Problems**, which is available at www.cengagebrain.com.

FIGURE P1.1 The file structure for Problems 1–4

PROJECT_CODE	PROJECT_MANAGER	MANAGER_PHONE	MANAGER_ADDRESS	PROJECT_BID_PRICE
21-5Z	Holly B. Parker	904-338-3416	3334 Lee Rd., Gainesville, FL 37123	16833460.00
25-2D	Jane D. Grant	615-898-9909	218 Clark Blvd., Nashville, TN 36362	12500000.00
25-5A	George F. Dorts	615-227-1245	124 River Dr., Franklin, TN 29185	32512420.00
25-9T	Holly B. Parker	904-338-3416	3334 Lee Rd., Gainesville, FL 37123	21563234.00
27-4Q	George F. Dorts	615-227-1245	124 River Dr., Franklin, TN 29185	10314545.00
29-2D	Holly B. Parker	904-338-3416	3334 Lee Rd., Gainesville, FL 37123	25559999.00
31-7P	William K. Moor	904-445-2719	216 Morton Rd., Stetson, FL 30155	56850000.00

SOURCE: Course Technology/Cengage Learning

Given the file structure shown in Figure P1.1, answer Problems 1–4.

1. How many records does the file contain? How many fields are there per record?
2. What problem would you encounter if you wanted to produce a listing by city? How would you solve this problem by altering the file structure?
3. If you wanted to produce a listing of the file contents by last name, area code, city, state, or zip code, how would you alter the file structure?
4. What data redundancies do you detect? How could those redundancies lead to anomalies?

FIGURE P1.5 The file structure for Problems 5–8

PROJ_NUM	PROJ_NAME	EMP_NUM	EMP_NAME	JOB_CODE	JOB_CHG_HOUR	PROJ_HOURS	EMP_PHONE
1	Hurricane	101	John D. Newson	EE	85.00	13.3	653-234-3245
1	Hurricane	105	David F. Schwann	CT	60.00	16.2	653-234-1123
1	Hurricane	110	Anne R. Ramoras	CT	60.00	14.3	615-233-5568
2	Coast	101	John D. Newson	EE	85.00	19.8	653-234-3254
2	Coast	108	June H. Sattlemir	EE	85.00	17.5	905-554-7812
3	Satellite	110	Anne R. Ramoras	CT	62.00	11.6	615-233-5568
3	Satellite	105	David F. Schwann	CT	26.00	23.4	653-234-1123
3	Satellite	123	Mary D. Chen	EE	85.00	19.1	615-233-5432
3	Satellite	112	Allecia R. Smith	BE	85.00	20.7	615-678-6879

SOURCE: Course Technology/Cengage Learning

5. Identify and discuss the serious data redundancy problems exhibited by the file structure shown in Figure P1.5.
6. Looking at the EMP_NAME and EMP_PHONE contents in Figure P1.5, what change(s) would you recommend?
7. Identify the various data sources in the file you examined in Problem 5.
8. Given your answer to Problem 7, what new files should you create to help eliminate the data redundancies found in the file shown in Figure P1.5?

FIGURE P1.9 The file structure for Problems 9–10

BUILDING_CODE	ROOM_CODE	TEACHER_LNAME	TEACHER_FNAME	TEACHER_INITIAL	DAYS_TIME
KOM	204E	vWilliston	Horace	G	MMF 8:00-8:50
KOM	123	Cordoza	Maria	L	MMF 8:00-8:50
LDB	504	Patroski	Donald	J	TTh 1:00-2:15
KOM	34	Hawkins	Anne	vV	MMF 10:00-10:50
JKP	225B	Risell	James		TTh 9:00-10:15
LDB	301	Robertson	Jeanette	P	TTh 9:00-10:15
KOM	204E	Cordoza	Maria	I	MMF 9:00-9:50
LDB	504	vWilliston	Horace	G	TTh 1:00-2:15
KOM	34	Cordoza	Maria	L	MMF 11:00-11:50
LDB	504	Patroski	Donald	J	MMF 2:00-2:50

SOURCE: Course Technology/Cengage Learning

9. Identify and discuss the serious data redundancy problems exhibited by the file structure shown in Figure P1.9. (The file is meant to be used as a teacher class assignment schedule. One of the many problems with data redundancy is the likely occurrence of data inconsistencies—two different initials have been entered for the teacher named Maria Cordoza.)
10. Given the file structure shown in Figure P1.9, what problem(s) might you encounter if building KOM were deleted?

In this chapter, you will learn:

- About data modeling and why data models are important
- About the basic data-modeling building blocks
- What business rules are and how they influence database design
- How the major data models evolved
- About emerging alternative data models and the need they fulfill
- How data models can be classified by their level of abstraction

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 and the relations among them.

First, you will learn some basic data-modeling concepts and how current data models 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 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.

Preview

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 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.

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 constitute the most basic information units employed by a system. Applications are created to manage data and to help transform data into information, but data are 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 men and the elephant: the blind man 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.

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. For example:

- 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.

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 are used, and in what time frames they are 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), two entities (EMPLOYEE and TRAINING), and a relationship between EMPLOYEE and TRAINING.

2.4.1 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 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:

- They help to standardize the company's view of data.
- They can be a communications tool between users and designers.
- They allow the designer to understand the nature, role, and scope of the data.
- They allow the designer to understand business processes.
- They allow 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. However, such a business rule can be enforced by application software.

2.4.2 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 that:

- 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.

As a general rule, to properly identify the relationship type, you should 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.

2.4.3 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.

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 Managed records, not relationships
Second	1970s	Hierarchical and network	IMS, ADABAS, IDS-II	Early database systems Navigational access
Third	Mid-1970s	Relational	DB2 Oracle MS SQL Server MySQL	Conceptual simplicity Entity relationship (ER) modeling and support for relational data modeling
Fourth	Mid-1980s	Object-oriented Object/relational (O/R)	Versant Objectivity/DB DB2 UDB Oracle 11g	Object/relational supports object data types Star Schema support for data warehousing Web databases become common
Fifth	Mid-1990s	XML Hybrid DBMS	dbXML Tamino DB2 UDB Oracle 11g MS SQL Server	Unstructured data support O/R model supports XML documents Hybrid DBMS adds object front end to relational databases Support large databases (terabyte size)
Emerging Models: NoSQL	Late 2000s to present	Key-value store Column store	SimpleDB (Amazon) BigTable (Google) Cassandra (Apache)	Distributed, highly scalable High performance, fault tolerant Very large storage (petabytes) Suited for sparse data Proprietary API



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 that model.

2.5.1 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 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.

2.5.2 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.

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 matrix 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.

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 are 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.

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-2012	502
10011	Dunne	Leona	K	713	894-1238	T1	250.00	16-Jun-2012	501
10012	Smith	Kathy	W	615	894-2285	S2	150.00	29-Jan-2013	502
10013	Olowski	Paul	F	615	894-2180	S1	300.00	14-Oct-2012	502
10014	Orlando	Myron		615	222-1672	T1	100.00	28-Dec-2013	501
10015	O'Brian	Amy	B	713	442-3381	T2	850.00	22-Sep-2012	503
10016	Brown	James	G	615	297-1228	S1	120.00	25-Mar-2013	502
10017	Williams	George		615	290-2556	S1	250.00	17-Jul-2012	503
10018	Farriss	Anne	G	713	382-7185	T2	100.00	03-Dec-2012	501
10019	Smith	Olette	K	615	297-3809	S2	500.00	14-Mar-2013	503

SOURCE: Course Technology/Cengage Learning

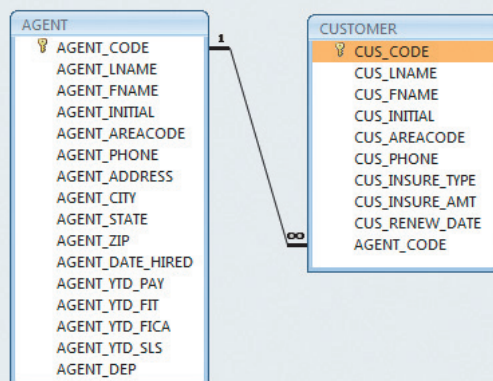


ONLINE CONTENT

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.

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 are stored in one table and the sales representative data are 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.

FIGURE 2.2 A relational diagram



SOURCE: Course Technology/Cengage Learning

Although the tables are independent of one another, you 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.

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 are 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 below.

- *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 are 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.

2.5.3 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.

NOTE

Because this chapter's objective is to introduce data-modeling concepts, a simplified ERD is discussed in this section. 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.

NOTE

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).

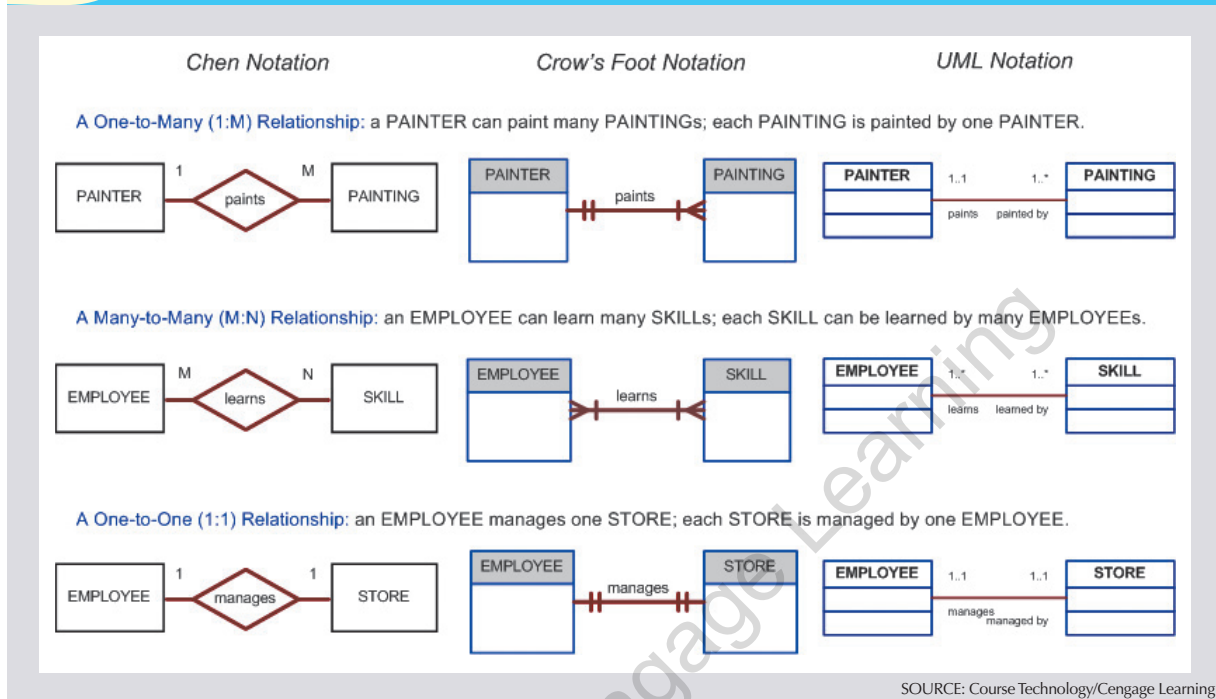
The left side of the ER diagram shows the Chen notation, based on Peter Chen's landmark paper. In this notation, the connectivities are written next to each entity box. Relationships are represented by a diamond connected to the related entities through a relationship line. The relationship name is written 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..*). Also, the UML notation uses names in both sides of the relationship. For example, to read the relationship between PAINTER and PAINTING, note that:

- 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.

FIGURE 2.3 The ER model notations



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's class diagram notation. Microsoft Visio Professional software was used to generate the Crow's Foot designs you will see in subsequent chapters.



ONLINE CONTENT

Aside from the Chen, Crow's Foot, and UML notations, there are other ER model notations. For a summary of the symbols used by several additional ER model notations, see **Appendix E, Comparison of ER Model Notations**, at www.cengagebrain.com.

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.

2.5.4 THE OBJECT-ORIENTED (OO) 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 *their 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)**.



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.

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:

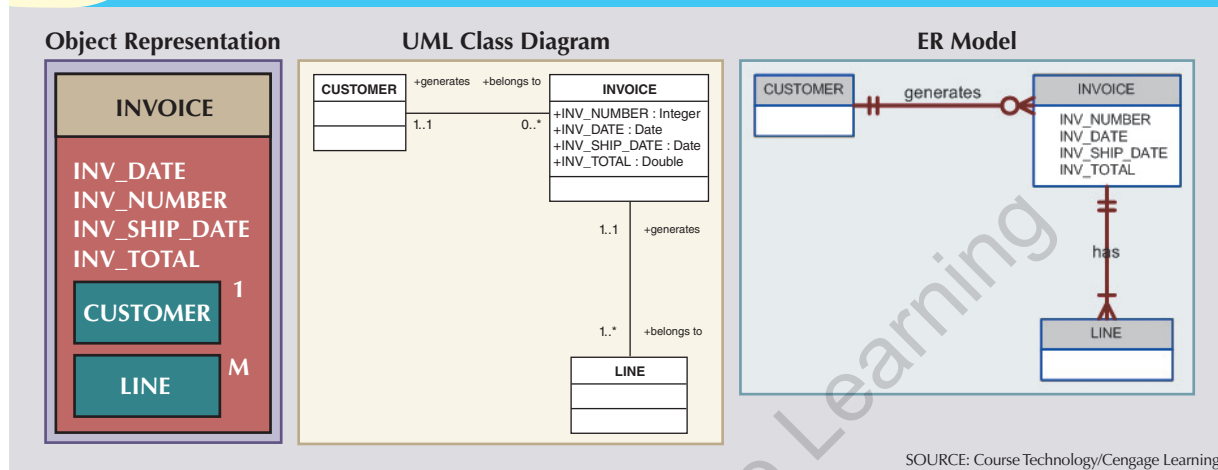
- An object is an abstraction of a real-world entity. In general terms, an object may be considered equivalent to an ER model's entity. More precisely, an object represents only one occurrence of an entity. (The object's semantic content is defined through several of the items in this list.)
- Attributes 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 their 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).

To illustrate the main concepts of the object-oriented data model, 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 OO, UML, and ER models



As you examine Figure 2.4, note that:

- 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.

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.

2.5.5 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 DBMS 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 use XML data soon realized that they needed to manage large amounts of unstructured data such as word-processing documents, Web pages, e-mails, and diagrams. To address this need, XML databases emerged to manage unstructured data within a native XML format. (See Chapter 14, 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.

2.5.6 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. According to many studies, the rapid pace of data growth is the top challenge for organizations,¹ 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-generated data and derive business insight from it, while simultaneously providing high performance and scalability at a reasonable cost.

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 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 sophisticated data analysis tools that may not be available in the relational environment.
- Generally speaking, 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

¹ See www.gartner.com/it/page.jsp?id=1460213, "Gartner Survey Shows Data Growth as the Largest Data Center Infrastructure Challenge," Nov. 2011.

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:

- 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.

Therefore, it is not surprising that some organizations are turning to 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% of market needs. For the remaining 2%, NoSQL databases are an option.

NoSQL Databases

Every time you search for a product on Amazon, send messages to friends in Facebook, watch a video in 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:

- Not based on the relational model, hence the name NoSQL.
- Supports distributed database architectures.
- Provides high scalability, high availability, and fault tolerance.
- Supports very large amounts of sparse data.
- Geared toward performance rather than transaction consistency.

Let's examine these characteristics in more detail.

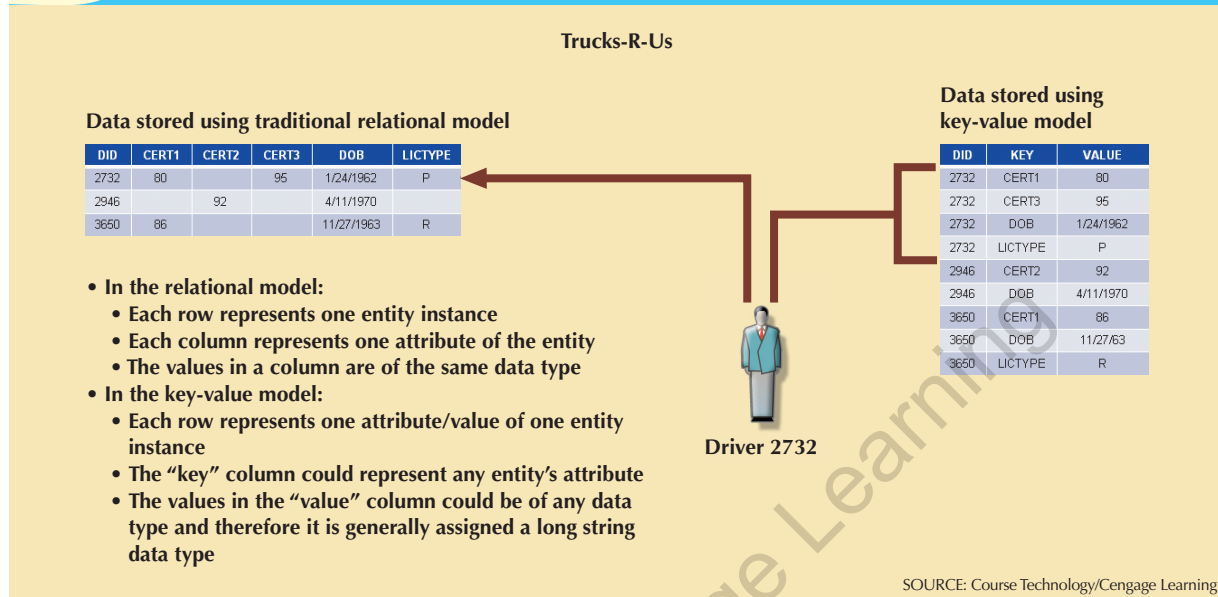
NoSQL databases are not based on the relational model. In fact, there is no standard NoSQL data model. To the contrary, many different data models are grouped under the NoSQL umbrella, from document databases to graph stores, column stores, and key-value stores. It is still too early to know which, if any, of these data models will survive and grow to become a dominant force in the database arena. However, the early success of products such as Amazon's SimpleDB, Google's BigTable, and Apache's Cassandra points to the *key-value stores* and *column stores* as the early leaders. The word *stores* indicates that these data models permanently store data in secondary storage, just like any other database. This added emphasis comes from the fact that these data models originated from programming languages (such as LISP), in which in-memory arrays of values are used to hold data.

The **key-value** data model is based on a structure composed of two data elements: a key and a value, in which every key has a corresponding value or set of values. The key-value data model is also referred to as the attribute-value or associative data model. To better understand the key-value model, look at the simple example in Figure 2.5.

Figure 2.5 shows the example of a small truck-driving company called Trucks-R-Us. Each of the three drivers has one or more certifications and other general information. Using this example, we can draw the following important points:

- In the relational model, every row represents a single entity occurrence and every column represents an attribute of the entity occurrence. Each column has a defined data type.

FIGURE 2.5 A simple key-value representation



- In the key-value data model, each row represents one attribute of one entity instance. The “key” column points to an attribute and the “value” column contains the actual value for the attribute.
- The data type of the “value” column is generally a long string to accommodate the variety of actual data types of the values placed in the column.
- To add a new entity attribute in the relational model, you need to modify the table definition. To add a new attribute in the key-value store, you add a row to the key-value store, which is why it is said to be “schema-less.”
- NoSQL databases do not store or enforce relationships among entities. The programmer is required to manage the relationships in the program code. Furthermore, all data and integrity validations must be done in the program code (although some implementations have been expanded to support metadata).
- NoSQL databases use their own native application programming interface (API) with simple data access commands, such as *put*, *read*, and *delete*. Because there is no declarative SQL-like syntax to retrieve data, the program code must take care of retrieving related data in the correct way.
- Indexing and searches can be difficult. Because the “value” column in the key-value data model could contain many different data types, it is often difficult to create indexes on the data. At the same time, searches can become very complex.

As a matter of fact, you could use the key-value structure as a general data modeling technique when attributes are numerous but actual data values are scarce. The key-value data model is not exclusive of NoSQL databases; actually, key-value data structures could reside inside a relational database. However, because of the problems with maintaining relationships and integrity within the data, and the increased complexity of even simple queries, key-value structures would be a poor design for most structured business data.

Several NoSQL database implementations, such as Google’s BigTable and Apache’s Cassandra, have extended the key-value data model to group multiple key-value sets into column families or column stores. In addition, such implementations support features such as versioning using a date/time stamp. For example, BigTable stores data in the syntax of [row, column, time, value], where row, column, and value are string data types and time is a date/time data type. The key used to access the data is composed of (row, column, time), where time can be left blank to indicate the most recent stored value.

NoSQL supports distributed database architecture. One of the big advantages of NoSQL databases is that they generally use a distributed architecture. In fact, several of them (Cassandra, BigTable) are designed to use low-cost commodity servers to form a complex network of distributed database nodes. Remember that several NoSQL databases originated in the research labs of some of the most successful Web companies, and most started on very small budgets!

NoSQL supports very large amounts of sparse data. NoSQL databases can handle very high volumes of data. In particular, they are suited for **sparse data**—that is, for cases in which the number of attributes is very large but the number of actual data instances is low. Using the preceding example, drivers can take any certification exam, but they are not required to take all. In this case, if there are three drivers and three possible certificates for each driver, there will be nine possible data points. In practice, however, there are only four data instances. Now extrapolate this example for the case of a clinic with 15,000 patients and more than 500 possible tests, remembering that each patient can take a few tests but is not required to take all.

NoSQL provides high scalability, high availability, and fault tolerance. True to its Web origins, NoSQL databases are designed to support Web operations, such as the ability to add capacity in the form of nodes to the distributed database when the demand is high, and to do it transparently and without downtime. Fault tolerance means that if one of the nodes in the distributed database fails, it will keep operating as normal.

Most NoSQL databases are geared toward performance rather than transaction consistency. One of the biggest problems of very large distributed databases is enforcing data consistency. Distributed databases automatically make copies of data elements at multiple nodes to ensure high availability and fault tolerance. If the node with the requested data goes down, the request can be served from any other node with a copy of the data. However, what happens if the network goes down during a data update? In a relational database, transaction updates are guaranteed to be consistent or the transaction is rolled back. NoSQL databases sacrifice consistency to attain high levels of performance. (See Chapter 12, Distributed Database Management Systems, to learn more about this topic.) Some NoSQL databases provide a feature called **eventual consistency**, which means that updates to the database will propagate through the system and eventually all data copies will be consistent. With eventual consistency, data are not guaranteed to be consistent across all copies of the data immediately after an update.

NoSQL is one of the hottest items in database technologies today. But, as you learned in Chapter 1, it is only one of many emerging trends in data management. Whatever database technology you use, you need to be able to select the best tool for the job by understanding the pros and cons of each technology. The following section briefly summarizes the evolution of data models and provides some advantages and disadvantages of each.

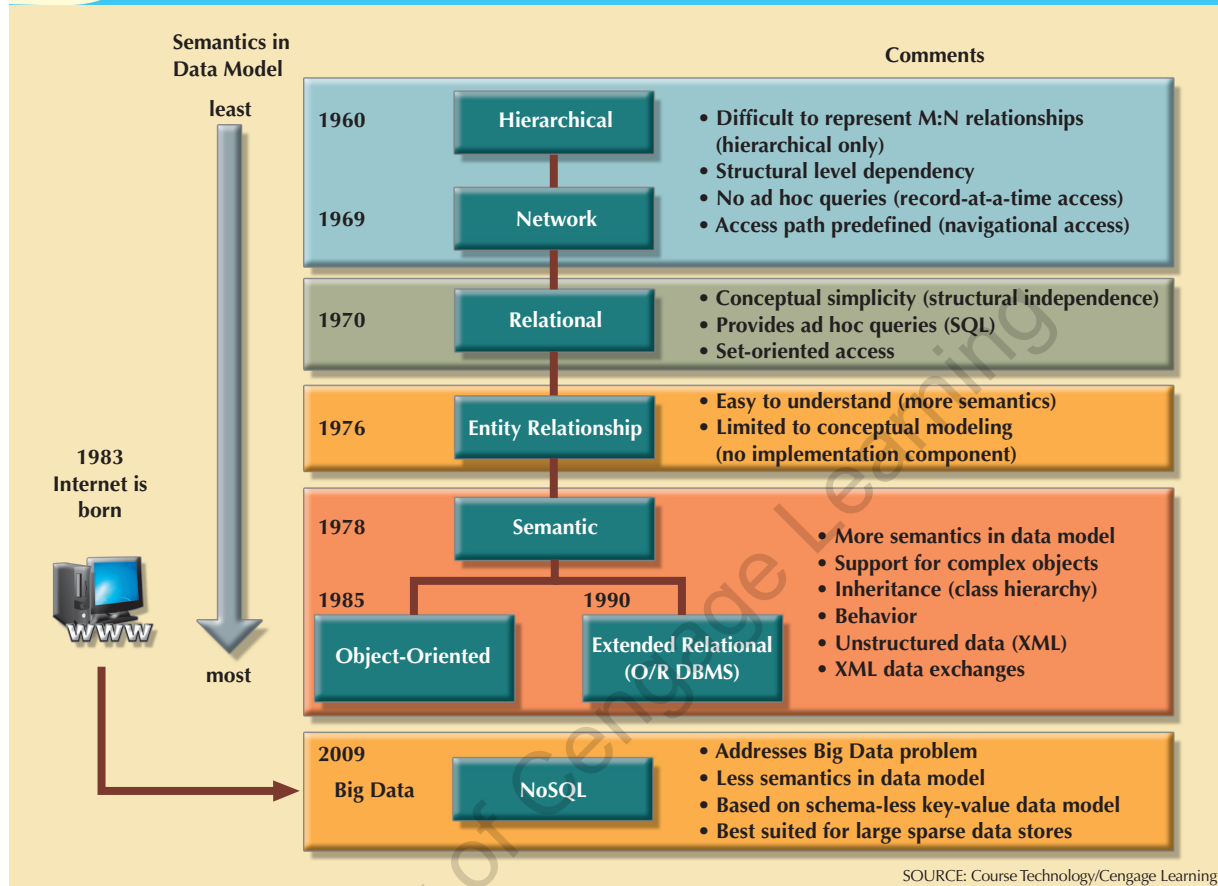
2.5.7 DATA MODELS: A SUMMARY

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 2.6.

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 are 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.

FIGURE 2.6 The evolution of data models



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 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 also 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 entity relationship 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.

TABLE 2.2

Advantages and Disadvantages of Various Database Models

DATA MODEL	DATA INDEPENDENCE	STRUCTURAL INDEPENDENCE	ADVANTAGES	DISADVANTAGES
Hierarchical	Yes	No	<ol style="list-style-type: none"> 1. It promotes data sharing. 2. Parent/child relationship promotes conceptual simplicity. 3. Database security is provided and enforced by DBMS. 4. Parent/child relationship promotes data integrity. 5. It is efficient with 1:M relationships. 	<ol style="list-style-type: none"> 1. Complex implementation requires knowledge of physical data storage characteristics. 2. Navigational system yields complex application development, management, and use; requires knowledge of hierarchical path. 3. Changes in structure require changes in all application programs. 4. There are implementation limitations (no multiparent or M:N relationships). 5. There is no data definition or data manipulation language in the DBMS. 6. There is a lack of standards.
Network	Yes	No	<ol style="list-style-type: none"> 1. Conceptual simplicity is at least equal to that of the hierarchical model. 2. It handles more relationship types, such as M:N and multiparent. 3. Data access is more flexible than in hierarchical and file system models. 4. Data owner/member relationship promotes data integrity. 5. There is conformance to standards. 6. It includes data definition language (DDL) and data manipulation language (DML) in DBMS. 	<ol style="list-style-type: none"> 1. System complexity limits efficiency—still a navigational system. 2. Navigational system yields complex implementation, application development, and management. 3. Structural changes require changes in all application programs.
Relational	Yes	Yes	<ol style="list-style-type: none"> 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. 2. Tabular view substantially improves conceptual simplicity, thereby promoting easier database design, implementation, management, and use. 3. Ad hoc query capability is based on SQL. 4. Powerful RDBMS isolates the end user from physical-level details and improves implementation and management simplicity. 	<ol style="list-style-type: none"> 1. The RDBMS requires substantial hardware and system software overhead. 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. 3. It may promote islands of information problems as individuals and departments can easily develop their own applications.
Entity relationship	Yes	Yes	<ol style="list-style-type: none"> 1. Visual modeling yields exceptional conceptual simplicity. 2. Visual representation makes it an effective communication tool. 3. It is integrated with the dominant relational model. 	<ol style="list-style-type: none"> 1. There is limited constraint representation. 2. There is limited relationship representation. 3. There is no data manipulation language. 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	<ol style="list-style-type: none"> 1. Semantic content is added. 2. Visual representation includes semantic content. 3. Inheritance promotes data integrity. 	<ol style="list-style-type: none"> 1. Slow development of standards caused vendors to supply their own enhancements, thus eliminating a widely accepted standard. 2. It is a complex navigational system. 3. There is a steep learning curve. 4. High system overhead slows transactions.
NoSQL	Yes	Yes	<ol style="list-style-type: none"> 1. High scalability, availability, and fault tolerance are provided. 2. It uses low-cost commodity hardware. 3. It supports Big Data. 4. Key-value model improves storage efficiency. 	<ol style="list-style-type: none"> 1. Complex programming is required. 2. There is no relationship support—only by application code. 3. There is no transaction integrity support. 4. In terms of data consistency, it provides an eventually consistent model.

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.

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
<i>Note:</i> 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 .							

2.6 DEGREES OF DATA ABSTRACTION

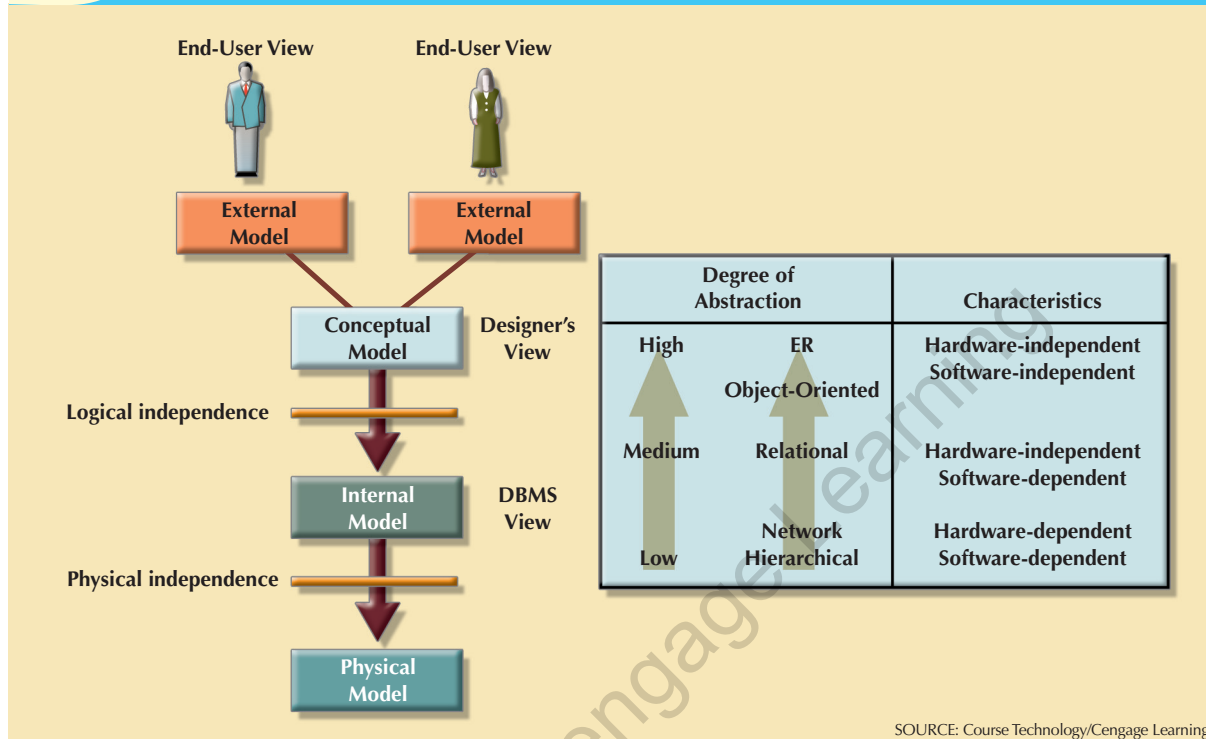
If you ask 10 database designers what a data model is, you will end up with 10 different answers—depending on the degree of data abstraction. To illustrate the meaning of data abstraction, consider the example of automotive design. A car designer begins by drawing the *concept* of the car to be produced. 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 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 2.7. 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.

2.6.1 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.

FIGURE 2.7 Data abstraction levels



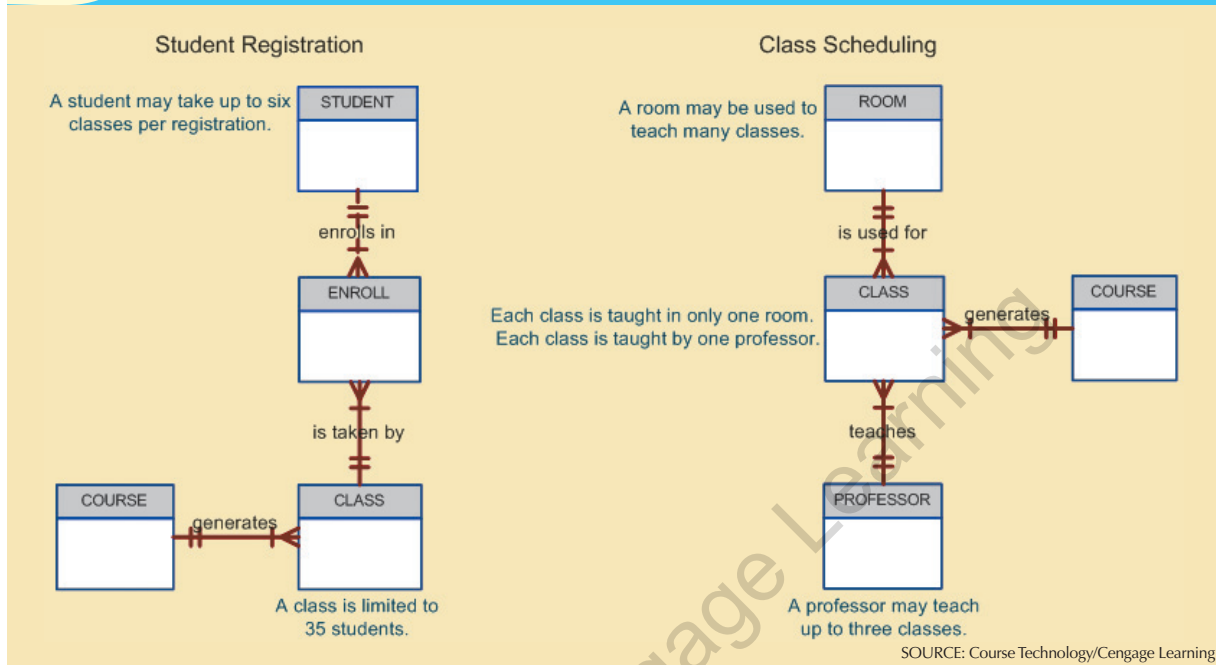
Because data are being modeled, ER diagrams will be used to represent the external views. A specific representation of an external view is known as an **external schema**. To illustrate the external model's view, examine the data environment of Tiny College.

Figure 2.8 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.

Note the entity relationships represented in Figure 2.8. For example:

- 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.

FIGURE 2.8 External models for Tiny College



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.

2.6.2 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 2.9. 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).

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 2.9.

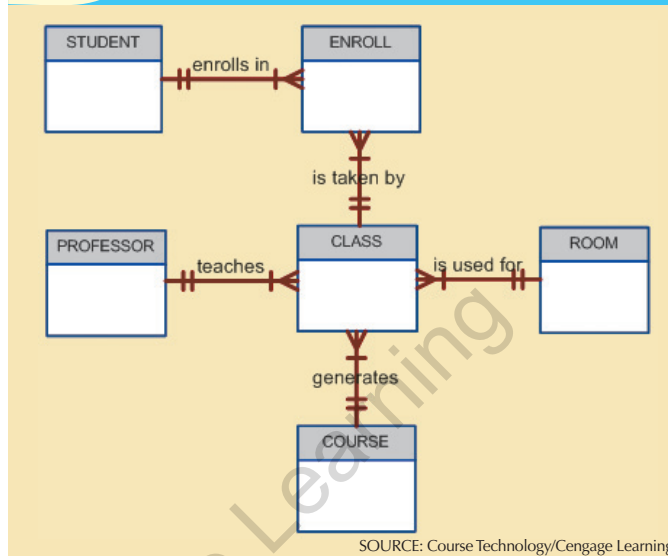
Second, the conceptual model is independent of both software and hardware. **Software independence** means that the model does not depend on the DBMS software used to implement the model. **Hardware independence** means that the

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.

2.6.3 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.

FIGURE 2.9 Conceptual model for Tiny College



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 2.9, 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 2.10.

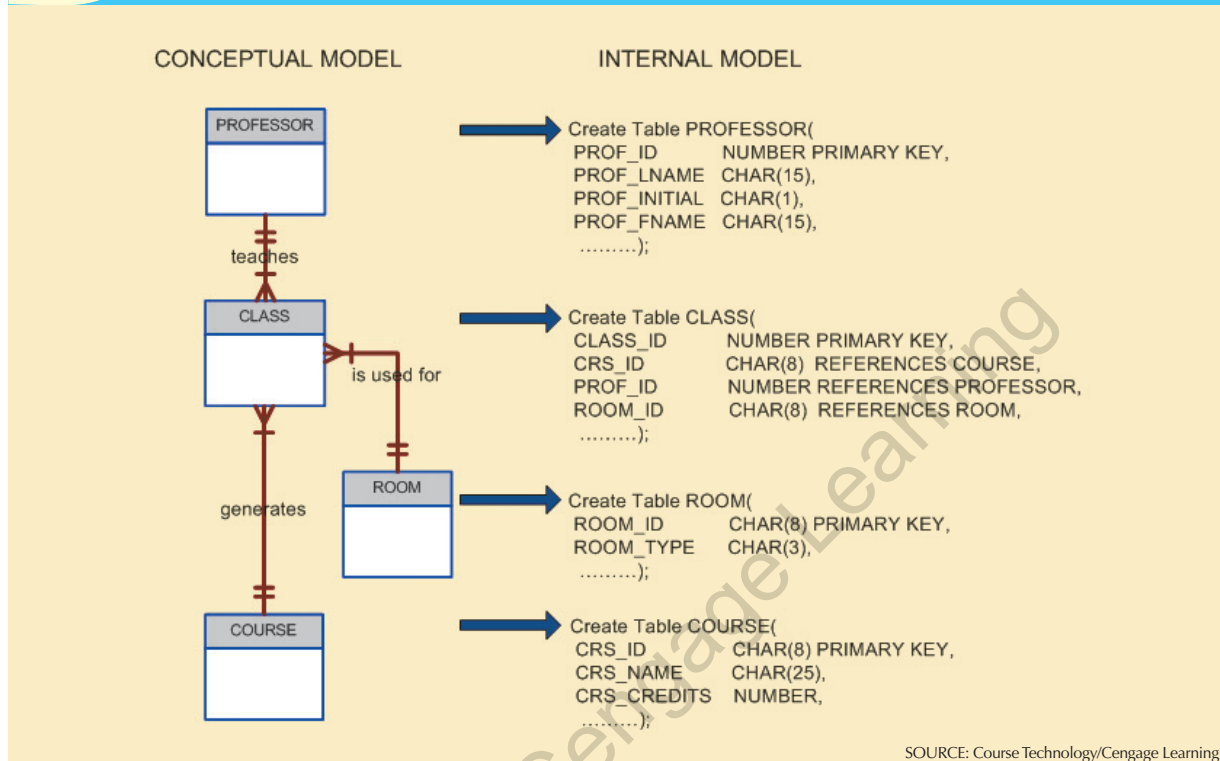
The development of a detailed internal model is especially important to database designers who work with hierarchical or network models because those models require precise specification of data storage location and data access paths. In contrast, the relational model requires less detail in its internal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

2.6.4 THE PHYSICAL MODEL

The **physical model** operates at the lowest level of abstraction, describing the way data are saved on storage media such as disks or tapes. 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.

FIGURE 2.10 Internal model for Tiny College



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 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

S U M M A R Y

- 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.
- 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 vs. 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.

K E Y T E R M S

American National Standards Institute (ANSI)	Crow's Foot notation	external model
attribute	data definition language (DDL)	external schema
Big Data	data manipulation language (DML)	hardware independence
business rule	data model	hierarchical model
Chen notation	entity	inheritance
class	entity instance	internal model
class diagram	entity occurrence	internal schema
class diagram notation	entity relationship diagram (ERD)	key-value
class hierarchy	entity relationship (ER) model (ERM)	logical design
conceptual model	entity set	logical independence
conceptual schema	eventual consistency	many-to-many (M:N or *.*)
connectivity	extended relational data model (ERDM)	relationship
constraint	Extensible Markup Language (XML)	method
		network model

NoSQL	one-to-one (1:1 or 1..1) relationship	schema
object	physical independence	segment
object-oriented data model (OODM)	physical model	semantic data model
object-oriented database management system (OODBMS)	relation	software independence
object/relational database management system (O/R DBMS)	relational database management system (RDBMS)	sparse data
one-to-many (1:M or 1..*) relationship	relational diagram	subschema
	relational model	table
	relationship	tuple
		Unified Modeling Language (UML)



ONLINE CONTENT

Flashcards and crossword puzzles for key term practice are available at www.cengagebrain.com.

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. What languages emerged to standardize the basic network data model, and why was such standardization important to users and designers?
5. Describe the basic features of the relational data model and discuss their importance to the end user and the designer.
6. Explain how the entity relationship (ER) model helped produce a more structured relational database design environment.
7. 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.
8. Why is an object said to have greater semantic content than an entity?
9. What is the difference between an object and a class in the object-oriented data model (OODM)?
10. How would you model Question 7 with an OODM? (Use Figure 2.4 as your guide.)
11. What is an ERDM, and what role does it play in the modern (production) database environment?
12. In terms of data and structural independence, compare file system data management with the five data models discussed in this chapter.
13. What is a relationship, and what three types of relationships exist?
14. Give an example of each of the three types of relationships.
15. What is a table, and what role does it play in the relational model?
16. What is a relational diagram? Give an example.
17. What is connectivity? (Use a Crow’s Foot ERD to illustrate connectivity.)
18. Describe the Big Data phenomenon.
19. What is sparse data? Give an example.
20. Define and describe the basic characteristics of a NoSQL database.
21. 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.
22. What is logical independence?
23. What is physical independence?

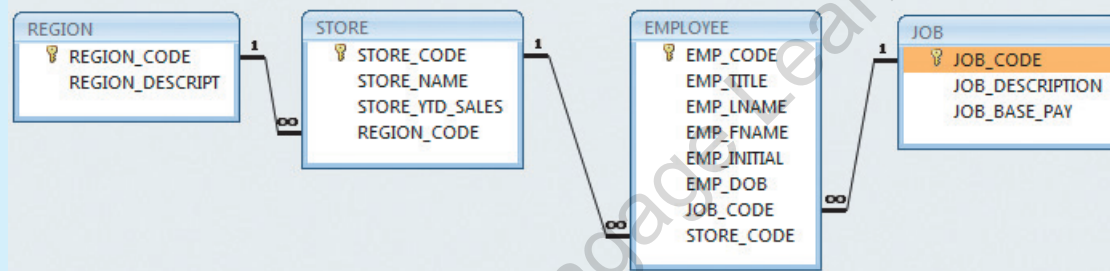
P R O B L E M S

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

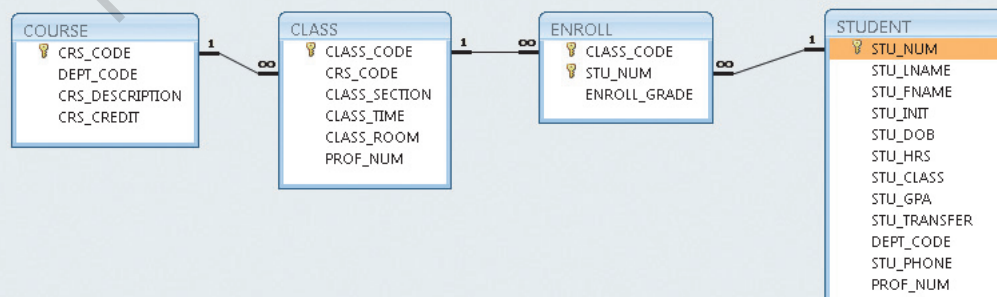


SOURCE: Course Technology/Cengage Learning

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

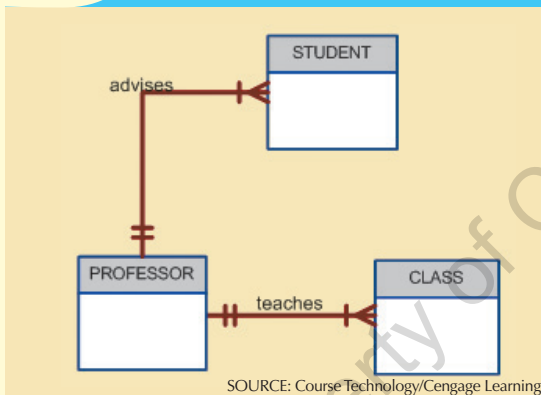


SOURCE: Course Technology/Cengage Learning

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.

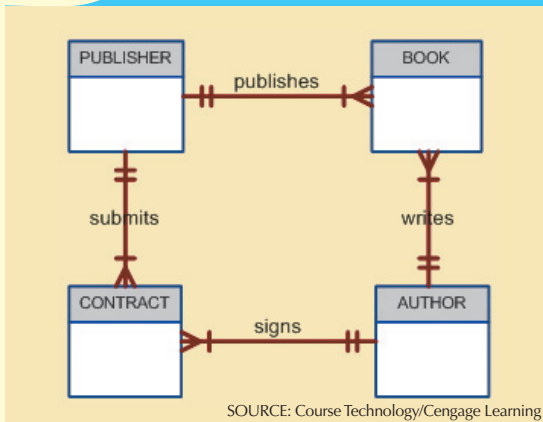
FIGURE P2.13 The Crow's Foot ERD for Problem 13



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.)
- 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.
 - During some period of time, a customer can rent many videotapes from the BigVid store. Each of BigVid's videotapes can be rented to many customers during that period of time.
 - An airliner can be assigned to fly many flights, but each flight is flown by only one airliner.
 - 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.
 - An employee may have earned many degrees, and each degree may have been earned by many employees.