

# Contents

## Data Management: Databases and Organizations

### *Open Edition*

**Richard T. Watson**

Department of MIS  
Terry College of Business  
The University of Georgia

Release date 2021-12-06



The online version of this book is licensed under the Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License CC BY-NC-SA 4.0.

## Preface

---

This is not your traditional database textbook. It differs in three fundamental ways.

*First*, it is deeper than most database books in its coverage of data modeling and SQL. The market seeks graduates who have these fundamental skills. Time and again, students who have completed my data management class have told me how these skills have been invaluable in their first and subsequent jobs. The intention is to place great emphasis on the core skills of data management. The consequence is that there is a better match between the skills students develop and those the market needs. This means that students find this text highly relevant.

*Second*, the treatments of data modeling and SQL are intertwined because my database teaching experience indicates that students more readily understand the intent of data modeling when they grasp the long-term goal—querying a well-designed relational database. The double helix, upward, intertwined, spiraling of data modeling and SQL is a unique pedagogical feature. Classroom testing indicates it is a superior method of teaching compared to handling data modeling and SQL separately. Students quickly understand the reason for data modeling and appreciate why it is a valuable skill. Also, rapid exposure to SQL means students gain hands-on experience more quickly.

*Third*, the book is broader than most database books. Databases are one component of an expansive organizational memory. Information systems professionals need to develop a wide perspective of data management if they are to comprehend fully the organizational role of information systems.

In essence, the book is deeper where it matters—data modeling and SQL—and broader to give students a managerial outlook and an understanding of the latest technological advancements. Information is a key resource for modern organizations. It is a critical input to managerial tasks. Because managers need high-quality information to manage change in a turbulent, global environment, many organizations have established systems for storing and retrieving data, the raw material of information. These storage and

retrieval systems are an organization's memory. The organization relies on them, just as individuals rely on their personal memory, to be able to continue as a going concern.

The central concern of information systems management is to design, build, and maintain information delivery systems. Information systems management needs to discover its organization's information requirements, which includes the needs of its customers, so that it can design systems to serve these demands. It must merge a system's design and information technology to build applications that provide the organization and its customers with data in a timely manner, appropriate formats, and at convenient locations. Furthermore, it must manage applications so they evolve to meet changing needs, continue to operate under adverse conditions, and are protected from unauthorized access.

An information delivery system has two components: data and processes. This book concentrates on data, which is customarily thought of as a database. I deliberately set out to extend this horizon, however, by including all forms of organizational data stores because I believe students need to understand the role of data management that is aligned with current practice. In my view, data management is the design and maintenance of computer-based organizational memory. Thus, you will find chapters on XML and organizational intelligence technologies.

The decision to start the book with a managerial perspective arises from the belief that successful information systems practice is based on matching managerial needs, social system constraints, and technical opportunities in an ecologically sound way. I want readers to appreciate the big picture before they become immersed in the intricacies of data modeling and SQL.

The second section of the book provides in-depth coverage of data modeling and SQL. Data modeling is the foundation of database quality. A solid grounding in data modeling principles and extensive practice are necessary for successful database design. In addition, this section exposes students to the full power of SQL.

I intend this book to be a long-term investment. There are useful reference sections for data modeling and SQL. The data modeling section details the standard structures and their relational mappings. The SQL section contains an extensive list of queries that serves as a basis for developing other SQL queries. The purpose of these sections is to facilitate pattern matching. For example, a student with an SQL query that is similar to a previous problem can rapidly search the SQL reference section to find the closest match. The student can then use the model answer as a guide to formulating the SQL query for the problem at hand. These reference sections are another unique teaching feature that will serve students well during the course and in their careers.

The third section has chapters on R, a statistics and graphics package, which provides the foundation necessary for the new chapters on data visualization, text mining, cluster computing, and dashboards. These chapters provide today's students with the skills they need to work in topical areas such as social media analytics and big data. These chapters are included in this third section, Advanced Data Management, which also covers spatial and temporal data, XML, and organizational intelligence.

The fourth and final section examines the management of organizational data stores. It covers data structures and storage, data processing architectures, SQL and Java, data integrity, and data administration.

A student completing this text will

- have a broad, managerial perspective of an organization's need for a memory;
- be able to design and create a relational database;
- be able to formulate complex SQL queries;
- be able to use R to create data visualizations, mine text data, write an R script for big data problems, and create dashboards;
- understand the purpose of XML and be able to design an XML schema, prepare an XML document, and write an XML stylesheet;
- have a sound understanding of database architectures and their managerial implications;

- be familiar with the full range of technologies available for organizational memory;
- be able to write a Java program to create a table from a CSV file and process a transaction;
- understand the fundamentals of data administration;

My goal is to give the reader a data management text that is innovative, relevant, and lively. I trust that you will enjoy learning about managing data in today's organization.

## **Supplements**

Accompanying this book are an instructor's manual and an extensive Web site that provides - slides in PowerPoint format; - all relational tables in the book in electronic format; - code for examples in the book; - answers to many of the exercises; - additional exercises.

## **Acknowledgments**

I thank my wife, Clare, and son, Ned, for help with prior editions of the book. I appreciate the efforts of Christopher Gauthier in converting the prior edition to Bookdown.

Richard T. Watson

Athens, Georgia

# Contents

## Section 1 The Managerial Perspective

*People only see what they are prepared to see.*

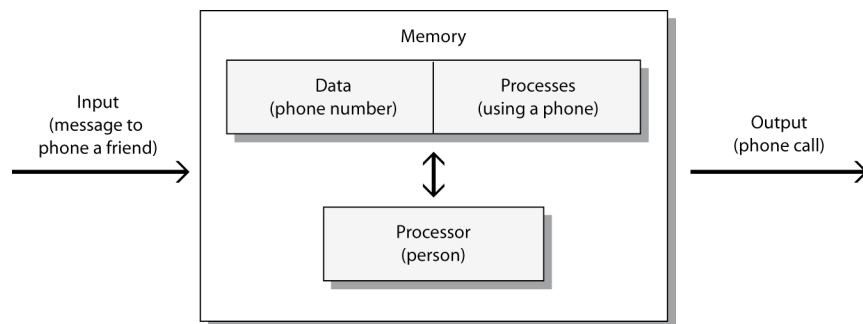
Ralph Waldo Emerson, *Journals*, 1863

Organizations are accumulating vast volumes of data because of the ease with which data can be captured and distributed in digital format (e.g., smartphone cameras and tweets). The world's data are estimated to be doubling every 1-2 years, and large companies now routinely manage petabytes ( $10^{15}$  bytes) of data every day. Data management is a critical function for many organizations.

The first section of this book prepares you to see the role of data and information in an organization. The managerial perspective on data management concentrates on why enterprises design and maintain data management systems, or organizational memories. Chapter 1 examines this topic by detailing the components of organizational memory and then discussing some of its common problems. The intention is to make you aware of the scope of data management and its many facets. Chapter 2 discusses the relationship between information and organizational goals. Again, a very broad outlook is adopted in order to provide a sweeping perspective on the relationship of information to organizational success and change.

At this point, we want to give you some **maps** for understanding the territory you will explore. Since the terrain is possibly very new, these maps initially may be hard to read, and so you may need to consult them several times before you understand the landscape you are about to enter.

The first map is based on the Newell-Simon model<sup>1</sup> of the human information processing system, which shows that humans receive input, process it, and produce output. The processing is done by a processor, which is linked to a memory that stores both data and processes. The processor component retrieves both data and processes from memory.



To understand this model, consider a person arriving on a flight who has agreed to meet a fellow traveler in the terminal. She receives a text message to meet her friend at B27. The message is input to her human information processing system. She retrieves the process for interpreting the message (i.e., decoding that B27 means terminal B and gate 27) and finds the terminal and gate. The person then walks to the terminal and gate, the output. Sometimes these processes are so well ingrained in our memory that we never think about retrieving them. We just do them automatically.

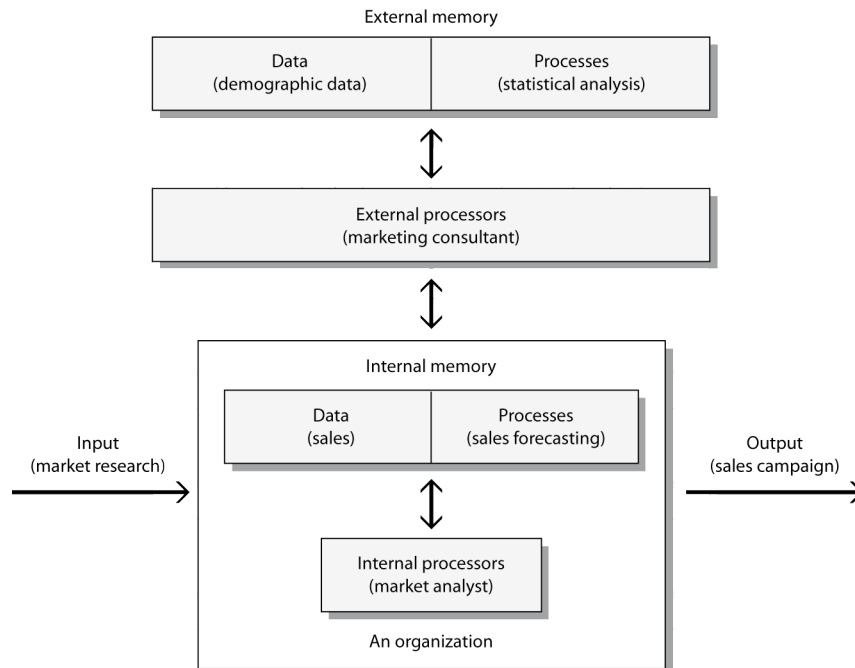
Human information processing systems are easily overloaded. Our memory is limited, and our ability to process data is restricted; thus we use a variety of external tools to extend and augment our capacities. A contacts app is an instance of external data memory. A recipe, a description of the process for preparing food, is an example of external process memory. Smartphones are now the external processor of choice that we use to augment our limited processing capacity.

The original model of human information processing can be extended to include external memory, for storing data and processes, and external processors, for executing processes.

### ***Missing this image***

<sup>1</sup>Newell, A., & Simon, H. A. (1972). Human problem solving. Englewood Cliffs, NJ: Prentice-Hall.

This model of augmented human information processing translates directly to an organizational setting. Organizations collect inputs from the environment: market research, customer complaints, and competitor actions. They process these data and produce outputs: sales campaigns, new products, price changes, and so on. The following figure gives an example of how an organization might process data. As a result of some market research (input), a marketing analyst (an internal processor) retrieves sales data (data) and does a sales forecast (process). The analyst also requests a marketing consultant (an external processor) to analyze (process) some demographic reports (data) before deciding to launch a new sales campaign (output).



An organization's memory comes in a variety of forms, as you will see in Chapter 1. This memory also can be divided into data and processes. The data part may contain information about customers. The process portion may store details of how to handle a customer order. Organizations use a variety of processors to handle data, including people and computers. Organizations also rely on external sources to extend their information processing capacity. For example, a business may use a specialist credit agency to check a customer's creditworthiness, or an engineering firm may use a cloud computing service for structural analysis of a bridge. Viewed this way, the augmented human information processing model becomes a pattern for an organizational information processing system.

This book focuses on the data side of organizational memory. While it is primarily concerned with data stored within the organization, there is also coverage of data in external memory. The process side of organizational memory is typically covered in a systems analysis and design or a business process management course.

# 1 Managing Data

*All the value of this company is in its people. If you burned down all our plants, and we just kept our people and our information files, we should soon be as strong as ever.*

Thomas Watson, Jr., former chairman of IBM<sup>2</sup>

## Learning objectives

Students completing this chapter will

- understand the key concepts of data management;
- recognize that there are many components of an organization's memory;
- understand the problems with existing data management systems;
- realize that successful data management requires an integrated understanding of organizational behavior and information technology.

## Introduction

Imagine what would happen to a bank that forgot who owed it money or a magazine that lost the names and addresses of its subscribers. Both would soon be in serious difficulty, if not out of business. Organizations have data management systems to record the myriad of details necessary for transacting business and making informed decisions. Since the birth of agriculture, societies and organizations have recorded data. The system may be as simple as carving a notch in a stick to keep a tally, or as intricate as modern database technology. A memory system can be as personal as a to-do list or as public as a library.

The management of organizational data, generally known as data management, requires skills in designing, using, and managing the memory systems of modern organizations. It requires multiple perspectives. Data managers need to see the organization as a social system and to understand data management technology. The integration of these views, the socio-technical perspective, is a prerequisite for successful data management.

Individuals also need to manage data. You undoubtedly are more familiar with individual memory management systems. They provide a convenient way of introducing some of the key concepts of data management.

## Individual data management

As humans, we are well aware of our limited capacity to remember many things. The brain, our internal memory, can get overloaded with too much detail, and its memory decays with time. We store a few things internally: our cell phone number, where we last parked our car, and faces of people we have met recently. We use external memory to keep track of those many things we would like to remember. External memory comes in a variety of forms.

On our smartphones, we have calendars to remind us of meetings and project deadlines. We have a contact app to record the addresses and phone numbers of those we contact frequently. We use to-do lists to remind us of the things we must do today or this week. The interesting thing about these aides-mémoire is that each has a unique way of storing data and supporting its rapid retrieval.

Calendars come in many shapes and forms, but they are all based on the same organizing principle. A set amount of space is allocated for each day of the year, and the spaces are organized in date and time

---

<sup>2</sup>Quinn, J. B. (1994). Appraising intellectual assets. \*The McKinsey Quarterly\*(2), 90-96.

order, which supports rapid retrieval. Some calendars have added features to speed up access. For example, electronic calendars usually have a button to select today's data.

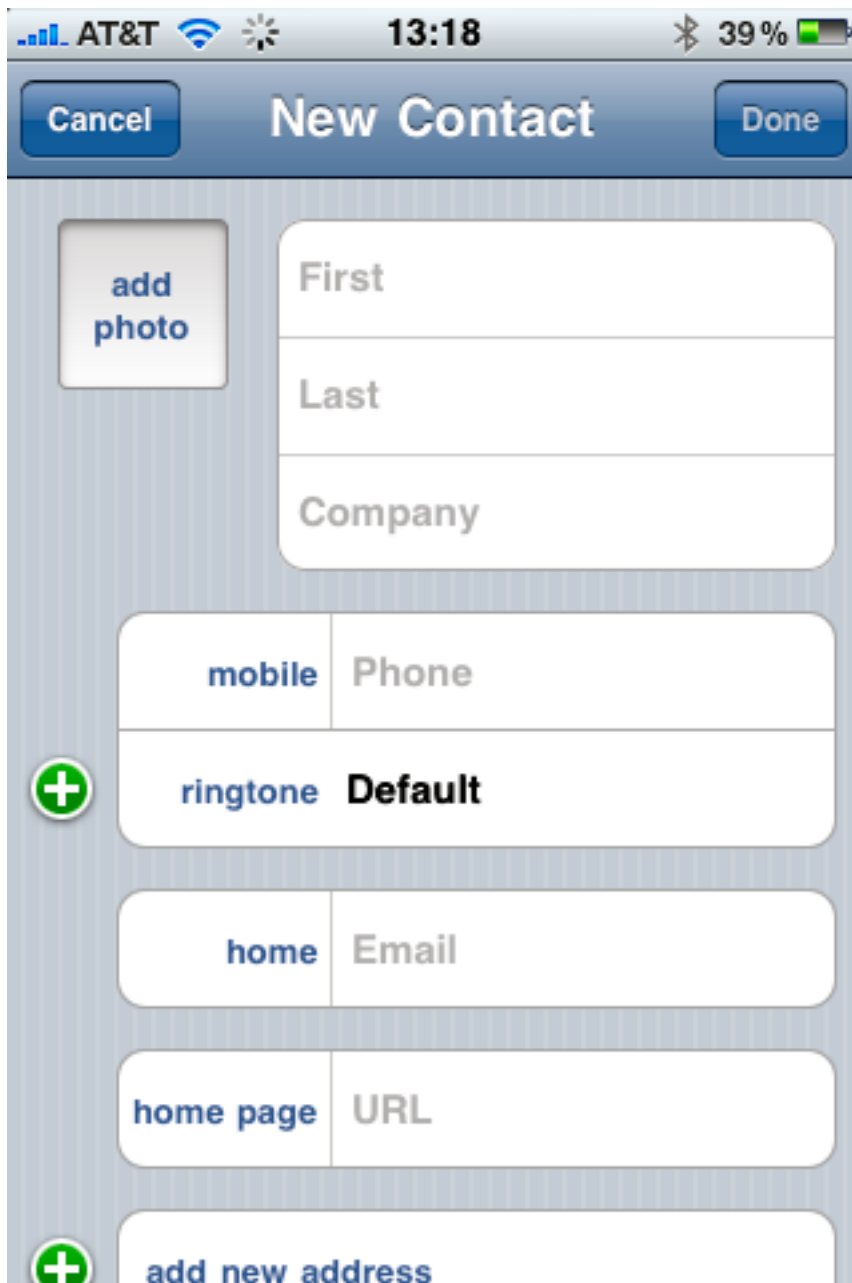
*A calendar*



Address books also have a standard format. They typically contain predefined spaces for storing address details (e.g., name, company, phone, and email). Access is often supported by a set of buttons for each letter of the alphabet and a search engine.

*An address book*





Cancel      New Contact      Done

add photo

First

Last

Company

mobile Phone

+ ringtone **Default**

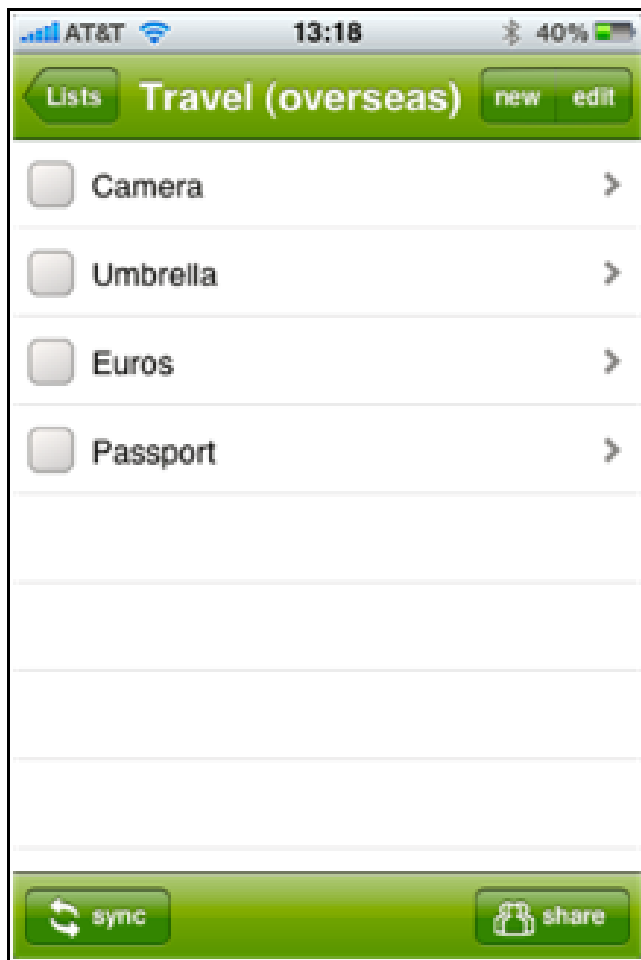
home Email

home page URL

+ add new address

The structure of to-do lists tends to be fairly standard. They are often set up in list format with a small left-hand margin. The idea is to enter each item to be done on the right side of the screen. The left side is used to check or mark those tasks that have been completed. The beauty of the check method is that you can quickly scan the left side to identify incomplete tasks.

*A to-do or reminder list*



Many people use some form of the individual memory systems just described. They are typically included in the suite of standard applications for a smart phone.

These three examples of individual memory systems illustrate some features common to all data management systems:

- There is a storage medium. Data are stored electronically in each case.
- There is a structure for storing data. For instance, the address book has labeled spaces for entering pertinent data.
- The interface organized for rapid data entry and retrieval. A calendar is stored in date and time sequence so that the data space for any appointment for a particular day can be found quickly.
- The selection of a data management system frequently requires a trade-off decision. In these examples, the trade-off is screen dimensions versus the amount of data that can be seen without scrolling. For example, you will notice the address book screen is truncated and will need to be scrolled to see full address details.

#### *Skill builder*

Smart phones have dramatically changed individual data management. We now have calendars, address books, to-do lists, and many more apps literally in our hands. What individual data are still difficult to manage? What might be the characteristics of an app for these data?

There are differences between internal and external memories. Our internal memory is small, fast, and convenient (our brain is always with us—well, most of the time). External memory is often slower to reference and not always as convenient. The two systems are interconnected. We rely on our internal memory to access external memory. Our internal memory and our brain’s processing skills manage the use of external memories. For example, we depend on our internal memory to recall how to use our smartphone and its apps. Again, we see some trade-offs. Ideally, we would like to store everything in our fast and convenient internal memory, but its limited capacity means that we are forced to use external memory for many items.

## Organizational data management

Organizations, like people, need to remember many things. If you look around any office, you will see examples of the apparatus of organizational memory: people, bookshelves, planning boards, and computers. The same principles found in individual memory systems apply to an organization’s data management systems.

There is a storage medium. In the case of computers, the storage medium varies. Small files might be stored on a USB drive and large, archival files on a magnetic disk. The chapter *Data Structure and Storage* discusses electronic storage media in more detail.

A table is a common structure for storing data. For example, if we want to store details of customers, we can set up a table with each row containing individual details of a customer and each column containing data on a particular feature (e.g., customer code).

Storage devices are organized for rapid data entry and retrieval. Time is the manager’s enemy: too many things to be done in too little time. Customers expect rapid responses to their questions and quick processing of their transactions. Rapid data access is a key goal of nearly all data management systems, but it always comes at a price. Fast access memories cost more, so there is nearly always a trade-off between access speed and cost.

As you will see, selecting *how* and *where* to store organizational data frequently involves a trade-off. Data managers need to know and understand what the compromises entail. They must know the key questions to ask when evaluating choices.

When we move from individual to organizational memory, some other factors come into play. To understand these factors, we need to review the different types of information systems. The automation of routine business transactions was the earliest application of information technology to business. A transaction processing system (TPS) handles common business tasks such as accounting, inventory, purchasing, and sales. The realization that the data collected by these systems could be sorted, summarized, and rearranged gave birth to the notion of a management information system (MIS). Furthermore, it was recognized that when internal data captured by a TPS is combined with appropriate external data, the raw material is available for a decision support system (DSS). Online analytical processing (OLAP), data mining, business intelligence (BI), and machine learning are techniques analyzing data captured by business transactions and gathered from other sources (these systems are covered in detail in Chapter 14). The purpose of each of these systems is described in the following table and their interrelationship can be understood by examining the information systems cycle.

### *Types of information systems*

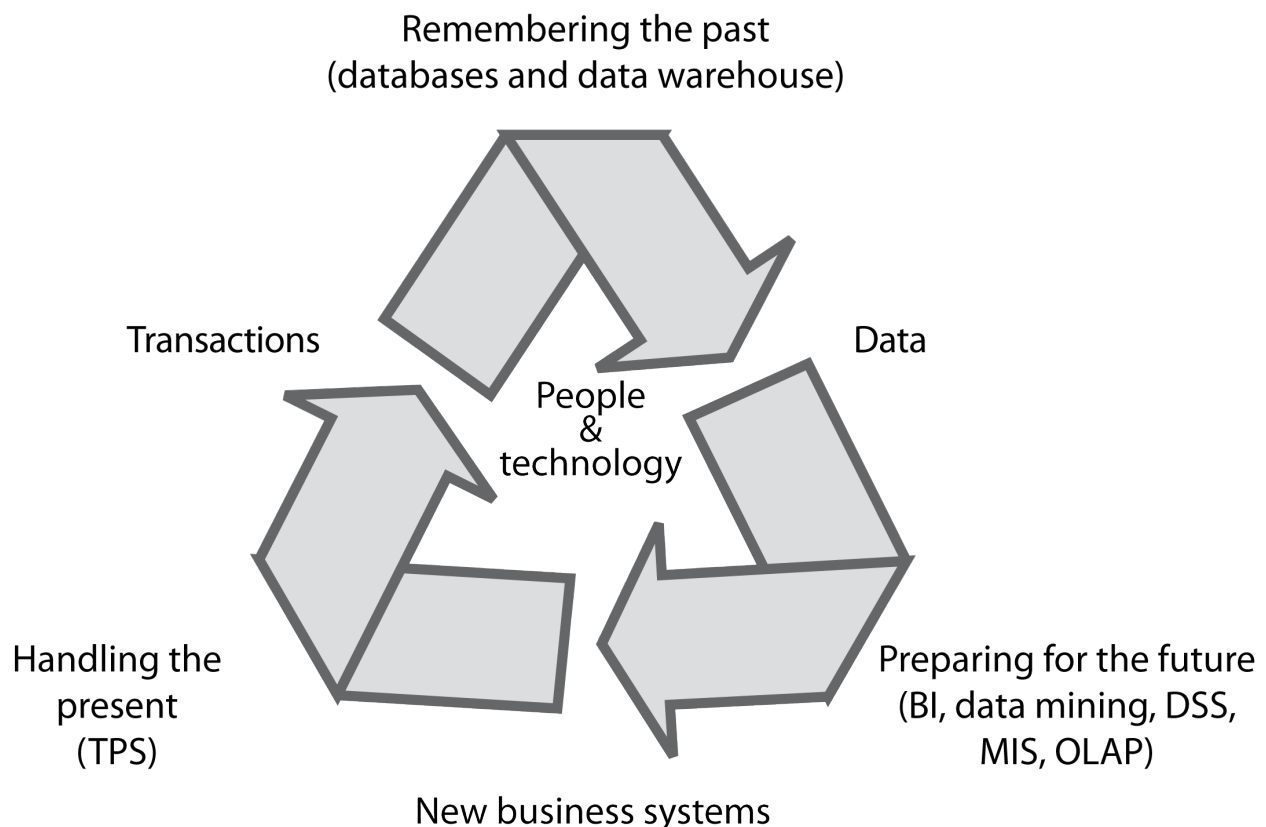
	Type of information system	System’s purpose
TPS	Transaction processing system	Collect and store data from routine transactions
MIS	Management information system	Convert data from a TPS into information for planning, controlling, and managing an organization

	Type of information system	System's purpose
DSS	Decision support system	Support managerial decision making by providing models for processing and analyzing data
BI	Business intelligence	Gather, store, and analyze data to improve decision making
OLAP	Online analytical processing	Provide a multidimensional view of data
	Data mining	Use of statistical analysis and artificial intelligence techniques to identify hidden relationships in data
	Machine learning	Using software to make decisions or recommendations traditionally made by humans.

### The information systems cycle

The various systems and technologies found in an organization are linked in a cycle. The routine ongoing business of the organization is processed by TPSs, the systems that handle the present. Data collected by TPSs are stored in databases, a record of the past, the history of the organization and its interaction with those with whom it conducts business. These data are converted into information by analysts using a variety of software (e.g., a DSS). These technologies are used by the organization to prepare for the future (e.g., sales in Finland have expanded, so we will build a new service center in Helsinki). The business systems created to prepare for the future determine the transactions the company will process and the data that will be collected, and the process continues. The entire cycle is driven by people using technology (e.g., a customer booking a hotel room via a Web browser).

*The information systems cycle*



Decision making, or preparing for the future, is the central activity of modern organizations. Today's

organizations are busy turning out goods, services, and decisions. Knowledge and information workers, over half of the U.S. labor force, produce the bulk of GDP. Many of these people are decision makers. Their success, and their organization's as well, depends on the quality of their decisions.

Industrial society is a producer of goods, and the hallmark of success is product quality. Japanese manufacturers convincingly demonstrated that focusing on product quality is the key to market leadership and profitability. The methods and the philosophy of quality gurus, such as W. Edwards Deming, have been internationally recognized and adopted by many providers of goods and services. We are now in the information age as is evidenced by the key consumer products of the times, such as smart phones, tablets, and wearables. These are all information appliances, and they are supported by a host of information services. For example, consider how Apple connects together its various devices and services through cloud-based systems. For example, a person can buy an electronic book from Apple's store to read with the iBooks app on an iPhone or iPad.

In the information society, which is based on innovation, knowledge, and services, the key determinant of success has shifted from product quality to decision quality. In the turbulent environment of global business, successful organizations are those able to quickly make high-quality decisions about what customers will buy, how much they will pay, and how to deliver a high-quality experience with a minimum of fuss. Companies are very dependent on information systems to create value for their customers.

## Desirable attributes of data

Once we realize the critical importance of data to organizations, we can recognize some desirable attributes of data.

### *Desirable attributes of data*

Shareable	Readily accessed by more than one person at a time
Transportable	Easily moved to a decision maker
Secure	Protected from destruction and unauthorized use
Accurate	Reliable, precise records
Timely	Current and up-to-date
Relevant	Appropriate to the decision

**Shareable** Organizations contain many decision makers. There are occasions when more than one person will require access to the same data at the same time. For example, in a large bank it would not be uncommon for two customer representatives simultaneously to want data on the latest rate for a three-year certificate of deposit. As data become more volatile, shareability becomes more of a problem. Consider a restaurant. The permanent menu is printed, today's special might be displayed on a blackboard, and the waiter tells you what is no longer available.

**Transportable** Data should be transportable from their storage location to the decision maker. Technologies that transport data have a long history. Homing pigeons were used to relay messages by the Egyptians and Persians 3,000 years ago. The telephone revolutionized business and social life because it rapidly transmitted voice data. Computers have changed the nature of many aspects of business because they enable the transport of text, visuals, voice and video.

Today, transportability is more than just getting data to a decision maker's desk. It means getting product availability data to a salesperson in a client's office or advising a delivery driver, en route, of address details for an urgent parcel pickup. The general notion is that decision makers should have access to relevant data whenever and wherever required, although many organizations are still some way from reaching this target.

**Secure** In an information society, organizations value data as a resource. As you have already learned, data support day-to-day business transactions and decision making. Because the forgetful organization will soon be out of business, organizations are very vigilant in protecting their data. There are a number of actions that organizations take to protect data against loss, sabotage, and theft. A common approach is to duplicate data and store the copy, or copies, at other locations. This technique is popular for data stored in computer systems. Access to data is often restricted through the use of physical barriers (e.g., a vault) or electronic barriers (e.g., a password). Another approach, which is popular with firms that employ knowledge workers, is a noncompete contract. For example, some software companies legally restrain computer programmers from working for a competitor for two years after they leave, hoping to prevent the transfer of valuable data, in the form of the programmer's knowledge of software, to competitors.

**Accurate** You probably recall friends who excel in exams because of their good memories. Similarly, organizations with an accurate memory will do better than their less precise competitors. Organizations need to remember many details precisely. For example, an airline needs accurate data to predict the demand for each of its flights. The quality of decision making will drop dramatically if managers use a data management system riddled with errors.

Polluted data threatens a firm's profitability. One study suggests that missing, wrong, and otherwise bad data cost U.S. firms billions of dollars annually. The consequences of bad data include improper billing, cost overruns, delivery delays, and product recalls. Because data accuracy is so critical, organizations need to be watchful when capturing data—the point at which data accuracy is most vulnerable.

**Timely** The value of a collection of data is often determined by its age. You can fantasize about how rich you would be if you knew tomorrow's stock prices. Although decision makers are most interested in current data, the required currency of data can vary with the task. Operational managers often want real-time data. They want to tap the pulse of the production line so that they can react quickly to machine breakdowns or quality slippages. In contrast, strategic planners might be content with data that are months old because they are more concerned with detecting long-term trends.

**Relevant** Organizations must maintain data that are relevant to transaction processing and decision making. In processing a credit card application, the most relevant data might be the customer's credit history, current employment status, and income level. Hair color would be irrelevant. When assessing the success of a new product line, a marketing manager probably wants an aggregate report of sales by marketing region. A voluminous report detailing every sale would be irrelevant. Data are relevant when they pertain directly to the decision and are aggregated appropriately.

Relevance is a key concern in designing a data management system. Clients have to decide what should be stored because it is pertinent now or could have future relevance. Of course, identifying data that might be relevant in the future is difficult, and there is a tendency to accumulate too much. Relevance is also an important consideration when extracting and processing data from a data management system. Provided the germane data are available, query languages can be used to aggregate data appropriately.

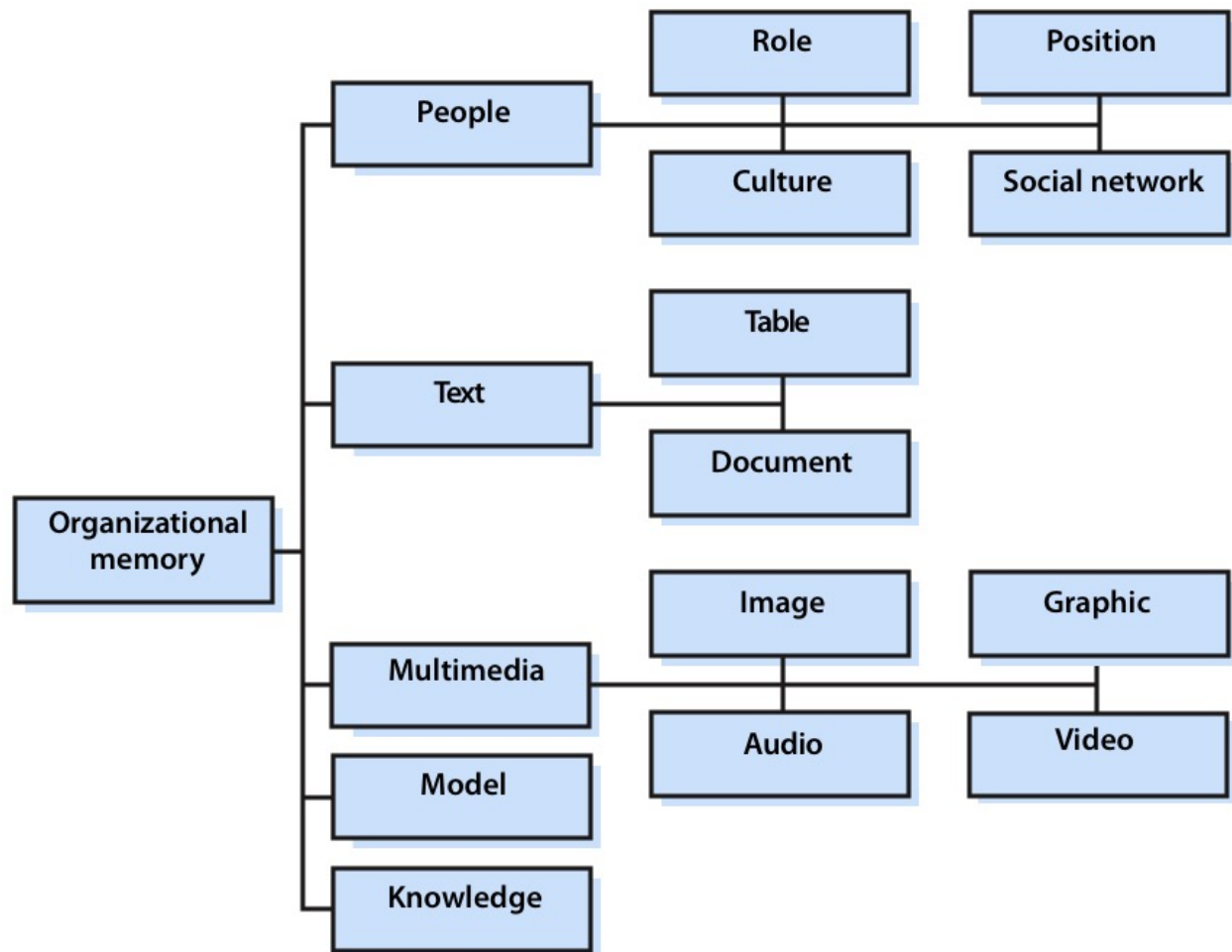
In the final years of the twentieth century, organizations started to share much of their data, both high and low volatility, via the Web. This move increased shareability, timeliness, and availability, and it has lowered the cost of distributing data.

In summary, a data management system for maintaining an organization's memory supports transaction processing, remembering the past, and decision making. Its contents must be shareable, secure, and accurate. Ideally, the clients of a data management system must be able to get timely and relevant data when and where required. A major challenge for data management professionals is to create data management systems that meet these criteria. Unfortunately, many existing systems fail in this regard, though we can understand some of the reasons why by reviewing the components of existing organizational memory systems.

## Components of organizational memory

An organization's memory resides on a variety of media in a variety of ways. It is in people, standard operating procedures, roles, organizational culture, physical storage equipment, and electronic devices. It is scattered around the organization like pieces of a jigsaw puzzle designed by a berserk artist. The pieces don't fit together, they sometimes overlap, there are gaps, and there are no edge pieces to define the boundaries. Organizations struggle to design structures and use data management technology to link some of the pieces. To understand the complexity of this wicked puzzle, we need to examine some of the pieces. Data managers have a particular need to understand the different forms of organizational memory because their activities often influence a number of the components.

*Components of organizational memory*



**People** People are the linchpin of an organization's memory. They recall prior decisions and business actions. They create, maintain, evolve, and use data management systems. They are the major component of an organization's memory because they know how to use many of the other components. People extract data from the various elements of organizational memory to provide as complete a picture of a situation as possible.

Each person in an organization has a role and a position in the hierarchy. Role and position are both devices for remembering how the organization functions and how to process data. By labeling people (e.g., Chief Information Officer) and placing their names on an organizational chart, the organization creates another form of organizational memory.

Organizational culture is the shared beliefs, values, attitudes, and norms that influence the behavior and expectations of each person in an organization. As a long-lived and stable memory system, culture determines acceptable behavior and influences decision making.

People develop skills for doing their particular job—learning what to do, how to do it, and who can help them get things done. For example, they might discover someone in employee benefits who can handle personnel problems or a contact in a software company who can answer questions promptly. These social networks, which often take years to develop, are used to make things happen and to learn about the business environment. Despite their high value, they are rarely documented, at least not beyond an address book, and they are typically lost when a person leaves an organization.

**Conversations** are an important method for knowledge workers to create, modify, and share organizational memory and to build relationships and social networks. Discussions with customers are a key device for learning how to improve an organization’s products and services and learning about competitors. The *conversational company* can detect change faster and react more rapidly. The telephone, instant message, e-mail, coffee machine, cocktail hour, and cafeteria are all devices for promoting conversation and creating networks. Some firms deliberately create structures for supporting dialog to make the people component of organizational memory more effective.

Standard operating procedures exist for many organizational tasks. Processing a credit application, selecting a marketing trainee, and preparing a departmental budget are typical procedures that are clearly defined by many organizations. They are described on Web pages, computer programs, and job specifications. They are the way an organization remembers how to perform routine activities.

Successful people learn how to use organizational memory. They learn what data are stored where, how to retrieve them, and how to put them together. In promoting a new product, a salesperson might send the prospect a package containing some brochures and an email of a product review in a trade journal, and supply the phone number and e-mail address of the firm’s technical expert for that product. People’s recall of how to use organizational memory is the core component of organizational memory. Academics call this **metamemory**; people in business call it *learning the ropes*. New employees spend a great deal of time building their metamemory so that they can use organizational memory effectively. Without this knowledge, organizational memory has little value.

**Tables** A table is a common form of storing organizational data. The following table shows a price list in tabular form. Often, the first row defines the meaning of data in subsequent rows.

*A price list*

Product	Price
Pocket knife–Nile	4.5
Compass	10
Geopositioning system	500
Map measure	4.9

A table is a general form that describes a variety of other structures used to store data. Computer-based files are tables or can be transformed into tables; the same is true for general ledgers, worksheets, and spreadsheets. Accounting systems make frequent use of tables. As you will discover in the next section, the table is the central structure of the relational database model.

Data stored in tables typically have certain characteristics:

- Data in one column are of the same type. For example, each cell of the column headed “Price” contains a number. (Of course, the exception is the first row of each column, which contains the title of the column.)



- Data are limited by the width of the available space.

Rapid searching is one of the prime advantages of a table. For example, if the price list is sorted by product name, you can quickly find the price of any product.

Tables are a common form of storing organizational data because their structure is readily understood. People learn to read and build tables in the early years of their schooling. Also, a great deal of the data that organizations want to remember can be stored in tabular form.

**Documents** A **document**—of which reports, manuals, brochures, and memos are examples—is a common medium for storing organizational data. Although documents may be subdivided into chapters, sections, paragraphs, and sentences, they lack the regularity and discipline of a table. Each row of a table has the same number of columns, but each paragraph of a document does not have the same number of sentences.

Most documents are now stored electronically. Because of the widespread use of word processing, text files are a common means of storing documents. Typically, such files are read sequentially like a book. Although there is support for limited searching of the text, such as finding the next occurrence of a specified text string, text files are usually processed linearly.

Hypertext, the familiar linking technology of the Web, supports nonlinear document processing. A hypertext document has built-in linkages between sections of text that permit the reader to jump quickly from one part to another. As a result, readers can find data they require more rapidly.

Although hypertext is certainly more reader-friendly than a flat, sequential text file, it takes time and expertise to establish the links between the various parts of the text and to other documents. Someone familiar with the topic has to decide what should be linked and then establish these links. While it takes the author more time to prepare a document this way, the payoff is the speed at which readers of the document can find what they want.

**Multimedia** Many Web sites display multimedia objects, such as sound and video clips. Automotive company Web sites have video clips of cars, music outlets provide sound clips of new releases, and clothing companies have online catalogs displaying photos of their latest products. Maintaining a Web site, because of the many multimedia objects that some sites contain, has become a significant data management problem for some organizations. Consider the different types of data that a news outfit such as the British Broadcasting Corporation (BBC) has to store to provide a timely, informative, and engaging Web site.

**Images** Images are visual data: photographs and sketches. Image banks are maintained for several reasons. *First*, images are widely used for identification and security. Police departments keep fingerprints and mug shots. *Second*, images are used as evidence. Highly valuable items such as paintings and jewelry often are photographed for insurance records. *Third*, images are used for advertising and promotional campaigns, and organizations need to maintain records of material used in these ventures. Image archiving and retrieval are essential for mail-order companies, which often produce several photo-laden catalogs every year. *Fourth*, some organizations specialize in selling images and maintain extensive libraries of clip art and photographs.

**Graphics** Maps and engineering drawings are examples of electronically stored graphics. An organization might maintain a map of sales territories and customers. Manufacturers have extensive libraries of engineering drawings that define the products they produce. Graphics often contain a high level of detail. An engineering drawing will define the dimensions of all parts and may refer to other drawings for finer detail about any components.

A graphic differs from an image in that it contains explicitly embedded data. Consider the difference between an engineering plan for a widget and a photograph of the same item. An engineering plan shows dimensional data and may describe the composition of the various components. The embedded data are used to manufacture the widget. A photograph of a widget does not have embedded data and contains insufficient

data to manufacture the product. An industrial spy will receive far more for an engineering plan than for a photograph of a widget.

A geographic information systems (GIS) is a specialized graphical storage system for geographic data. The underlying structure of a GIS is a map on which data are displayed. A power company can use a GIS to store and display data about its electricity grid and the location of transformers. Using a pointing device such as a mouse, an engineer can click on a transformer's location to display a window of data about the transformer (e.g., type, capacity, installation date, and repair history). GISs have found widespread use in governments and organizations that have geographically dispersed resources.

**Audio** News organizations, such as National Public Radio (NPR), provide audio versions of their new stories for replay. Some firms conduct a great deal of their business by phone. In many cases, it is important to maintain a record of the conversation between the customer and the firm's representative. The Royal Hong Kong Jockey Club, which covers horse racing gambling in Hong Kong, records all conversations between its operators and customers. Phone calls are stored on a highly specialized voice recorder, which records the time of the call and other data necessary for rapid retrieval. In the case of a customer dispute, an operator can play back the original conversation.

**Video** A video clip can give a potential customer additional detail that cannot be readily conveyed by text or a still image. Consequently, some auto companies use video and virtual reality to promote their cars. On a visit to Toyota's Web site, you can view video clips of the latest models or rotate an image to view a car from multiple angles.

**Models** Organizations build mathematical models to describe their business. These models, usually placed in the broader category of DSS, are then used to analyze existing problems and forecast future business conditions. A mathematical model can often produce substantial benefits to the organization.

**Knowledge** Organizations build systems to capture the knowledge of their experienced decision makers and problem solvers. This expertise is typically represented as a set of rules, semantic nets, and frames in a knowledge base, another form of organizational memory.

**Decisions** Decision making is the central activity of modern organizations. Very few organizations, however, have a formal system for recording decisions. Most keep the minutes of meetings, but these are often very brief and record only a meeting's outcome. Because they do not record details such as the objectives, criteria, assumptions, and alternatives that were considered prior to making a decision, there is no formal audit trail for decision making. As a result, most organizations rely on humans to remember the circumstances and details of prior decisions.

**Specialized memories** Because of the particular nature of their business, some organizations maintain memories rarely found elsewhere. Perfume companies, for instance, maintain a library of scents, and paint manufacturers and dye makers catalog colors.

### **Components of organizational memory**

Organizations are not limited to their own memory stores. There are firms whose business is to store data for resale to other organizations. Such businesses have existed for many years and are growing as the importance of data in a postindustrial society expands. U.S. lawyers can use document management services to access the laws and court decisions of all 50 American states and the U.S. federal government. Similar legal data services exist in many other nations. There is a range of other services that provide news, financial, business, scientific, and medical data.

## Problems with data management systems

Successful management of data is a critical skill for nearly every organization. Yet few have gained complete mastery, and there are a variety of problems that typically afflict data management in most firms.

### *Problems with organizational data management systems*

Redundancy	Same data are stored in different systems
Lack of data control	Data are poorly managed
Poor interface	Data are difficult to access
Delays	There are frequently delays following requests for reports
Lack of reality	Data management systems do not reflect the complexity of the real world
Lack of data integration	Data are dispersed across different systems

### **Redundancy**

In many cases, data management systems have grown haphazardly. As a result, it is often the situation that the same data are stored in several different memories. The classic example is a customer's address, which might be stored in the sales reporting system, accounts receivable system, and the salesperson's address book. The danger is that when the customer changes address, the alteration is not recorded in all systems. Data redundancy causes additional work because the same item must be entered several times. Redundancy causes confusion when what is supposedly the same item has different values.

### **Lack of data control**

Allied with the redundancy problem is poor data control. Although data are an important organizational resource, they frequently do not receive the same degree of management attention as other important organizational resources, such as people and money. Organizations have a personnel department to manage human resources and a treasury to handle cash. The IS department looks after data captured by the computer systems it operates, but there are many other data stores scattered around the organization. Data are stored everywhere in the organization (e.g., on personal computers and departmental servers), but there is a general lack of data management. This lack is particularly surprising, since many pundits claim that data are a key competitive resource.

### **Poor interface**

Too frequently, the potential clients of data management systems have been deterred by an unfriendly interface. The computer interface for accessing a data store is sometimes difficult to remember for the occasional inquirer. People become frustrated and give up because their queries are rejected and error messages are unintelligible.

### **Delays**

Globalization and technology have accelerated the pace of business in recent years. Managers must make more decisions more rapidly. They cannot afford to wait for programmers to write special-purpose programs to retrieve data and format reports. They expect their questions to be answered rapidly, often within an hour and sometimes more quickly. Managers, or their support personnel, need query languages that provide rapid access to the data they need, in a format that they want.

## Lack of reality

Organizational data stores must reflect the reality and complexity of the real world. Consider a typical bank customer who might have a personal checking account, mortgage account, credit card account, and some certificates of deposit. When a customer requests an overdraft extension, the bank officer needs full details of the customer's relationship with the bank to make an informed decision. If customer data are scattered across unrelated data stores, then these data are not easily found, and in some cases important data might be overlooked. The request for full customer details is reasonable and realistic, and the bank officer should expect to be able to enter a single query to obtain it. Unfortunately, this is not always the case, because data management systems do not always reflect reality.

In this example, the reality is that the personal checking, mortgage, and credit card accounts, and certificates of deposit all belong to one customer. If the bank's data management system does not record this relationship, then it does not mimic reality. This makes it impossible to retrieve a single customer's data with a single query.

A data management system must meet the decision making needs of managers, who must be able to request both routine and ad hoc reports. To do so effectively, a data management system must reflect the complexity of the real world. If it does not store required organizational data or record a real-world relationship between data elements, then some managerial queries might not be answerable quickly.

## Lack of data integration

There is a general lack of data integration in most organizations. Not only are data dispersed in different forms of organizational memory (e.g., files and image stores), but even within one storage format there is often a lack of integration. For example, many organizations maintain file systems that are not integrated. Appropriate files in the accounting system may not be linked to the production system.

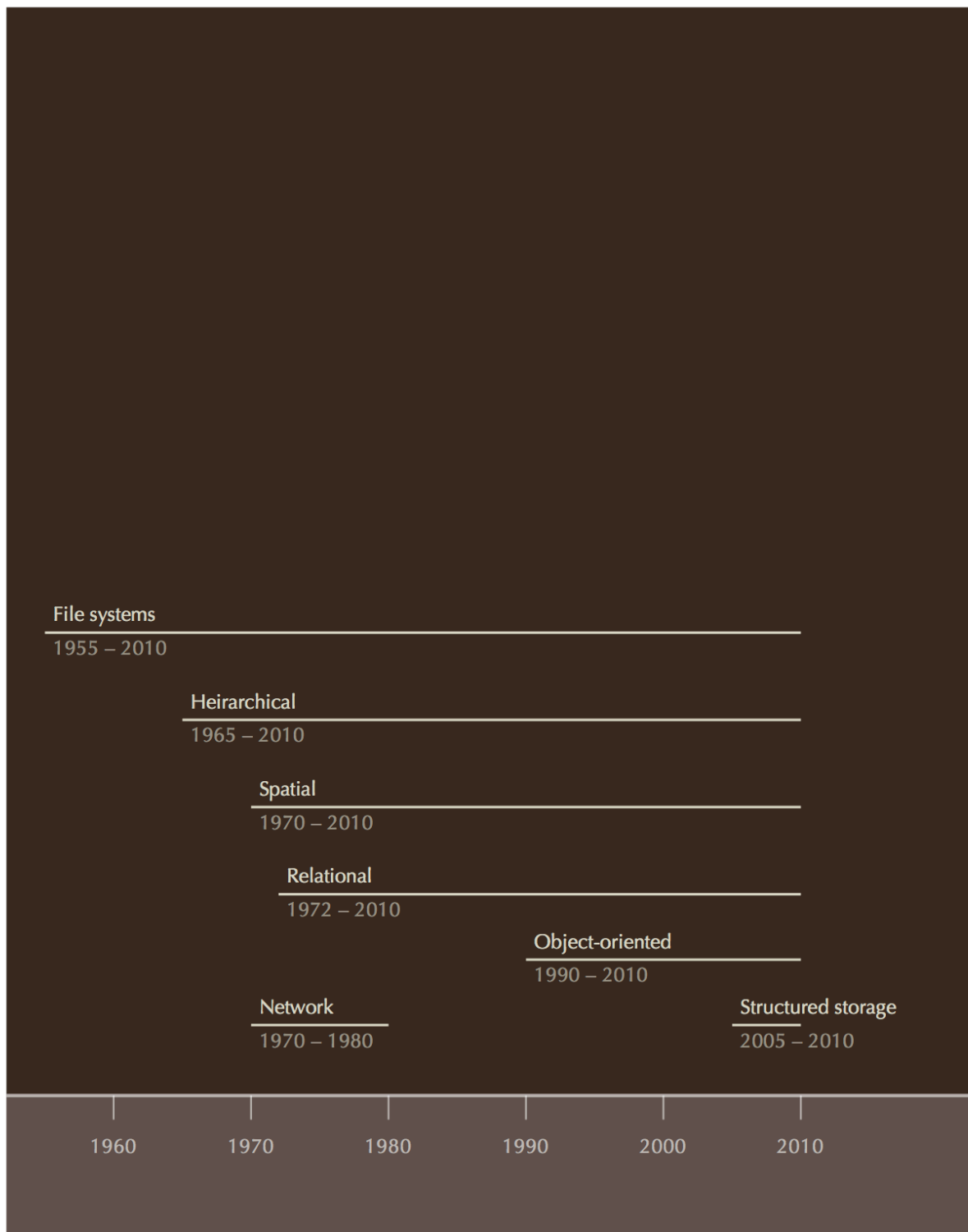
This lack of integration will be a continuing problem for most organizations for two important reasons. **First**, earlier computer systems might not have been integrated because of the limitations of available technology. Organizations created simple file systems to support a particular function. Many of these systems are still in use. **Second**, integration is a long-term goal. As new systems are developed and old ones rewritten, organizations can evolve integrated systems. It would be too costly and disruptive to try to solve the data integration problem in one step.

Many data management problems can be solved with present technology. Data modeling and relational database technology, topics covered in Section 2, help overcome many of the current problems.

## A brief history of data management systems

Data management is not a new organizational concern. It is an old problem that has become more significant, important, and critical because of the emergence of data as a critical resource for effective performance in the modern economy. Organizations have always needed to manage their data so that they could remember a wide variety of facts necessary to conduct their affairs. The recent history of computer-based data management systems is depicted in the following figure.

*Data management systems timeline*



File systems were the earliest form of data management. Limited by the sequential nature of magnetic tape technology, it was very difficult to integrate data from different files. The advent of magnetic disk technology in the mid-1950s stimulated development of integrated file systems, and the hierarchical database

management system (DBMS) emerged in the 1960s, followed some years later by the network DBMS. The spatial database, or geographic information system (GIS), appeared around 1970. Until the mid-1990s, the hierarchical DBMS, mainly in the form of IBM's DL/I product, was the predominant technology for managing data. It has now been replaced by the relational DBMS, a concept first discussed by Edgar Frank Codd in an academic paper in 1970 but not commercially available until the mid-1970s. In the late 1980s, the notion of an object-oriented DBMS, primed by the ideas of object-oriented programming, emerged as a solution to situations not handled well by the relational DBMS. Also around this time, the idea of modeling a database as a graph was introduced. Towards the end of the 20th century, XML was developed for exchanging data between computers, and it can also be used as a data store as you will learn in section 3. More recently, distributed file system, such as Hadoop, have emerged as alternative models for data management. Other recent data management systems include graph and NoSQL databases. While these are beyond the scope of an introductory data management text, if you decided to pursue a career in data management you should learn about their advantages and the applications to which they are well-suited. For example, graph databases are a good fit for the analysis of social networks.

This book concentrates on the relational model, currently the most widely used data management system. As mentioned, Section 2 is devoted to the development of the necessary skills for designing and using a relational database.

## Data, information, and knowledge

Often the terms *data* and *information* are used interchangeably, but they are distinctly different. Data are raw, unsummarized, and unanalyzed facts. Information is data that have been processed into a meaningful form.

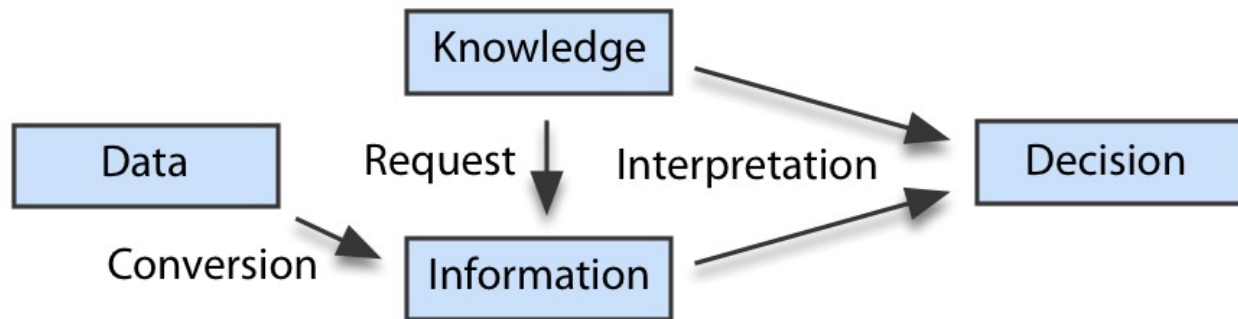
A list of a supermarket's daily receipts is data, but it is not information, because it is too detailed to be very useful. A summary of the data that gives daily departmental totals is information, because the store manager can use the report to monitor store performance. The same report might be data for a regional manager, because it is too detailed for meaningful decision making at the regional level. Information for a regional manager might be a weekly report of sales by department for each supermarket.

Data are always data, but one person's information can be another person's data. Information that is meaningful to one person can be too detailed for another. A manager's notion of information can change quickly, however. When a problem is identified, a manager might request finer levels of detail to diagnose the problem's cause. Thus, what was previously data suddenly becomes information because it helps solve the problem. When the problem is solved, the information reverts to data. There is a need for information systems that let managers customize the processing of data so that they always get information. As their needs change, they need to be able to adjust the detail of the reports they receive.

Knowledge is the capacity to use information. The education and experience that managers accumulate provide them with the expertise to make sense of the information they receive. Knowledge means that managers can interpret information and use it in decision making. In addition, knowledge is the capacity to recognize what information would be useful for making decisions. For example, a sales manager might know that requesting a report of profitability by product line is useful when she has to decide whether to employ a new product manager. Thus, when a new information system is delivered, managers need to be taught what information the system can deliver and what that information means.

The relationship between data, information, and knowledge is depicted in the following figure. A knowledgeable person requests information to support decision making. To fulfill the request, data are converted into information. Personal knowledge is then applied to interpret the requested information and reach a conclusion. Of course, the cycle can be repeated several times if more information is needed before a decision can be made. Notice how knowledge is essential for grasping what information to request and interpreting that information in terms of the required decision.

*The relationship between data, information, and knowledge*



## The challenge

A major challenge facing organizations is to make effective use of the data currently stored in their diverse data management systems. This challenge exists because these various systems are not integrated and many potential clients not only lack the training to access the systems but often are unaware what data exist. Before data managers can begin to address this problem, however, they must understand how organizational memories are used. In particular, they need to understand the relationship between information and managerial decision making. Data management is not a new problem. It has existed since the early days of civilization and will be an enduring problem for organizations and societies.

## Summary

Organizations must maintain a memory to process transactions and make decisions. Organizational data should be shareable, transportable, secure, and accurate, and provide timely, relevant information. The essential components are people (the most important), text, multimedia data, models, and knowledge. A wide variety of technologies can be used to manage data. External memories enlarge the range of data available to an organization. Data management systems often have some major shortcomings: redundancy, poor data control, poor interfaces, long lead times for query resolution, an inability to supply answers for questions posed by managers, and poor data integration. Data are raw facts; information is data processed into a meaningful form. Knowledge is the capacity to use information.

---

### Key terms and concepts

---

Data	Internal memory
Database management system (DBMS)	Knowledge
Data management	Machine learning
Data mining	Management information system (MIS)
Data security	Metamemory
Decision making	Online analytical processing (OLAP)
Decision quality	Organizational culture
Decision support system (DSS)	Organizational memory
External memory	Standard operating procedures
Geographic information system (GIS)	Tables
Information	Transaction processing system (TPS)

---

## References and additional readings

Davenport, T. H. (1998). Putting the enterprise into the enterprise system. *Harvard Business Review*, 76(4), 121-131.

## Exercises

1. What are the major differences between internal and external memory?
2. What is the difference between the things you remember and the things you record on your computer?
3. What features are common to most individual memory systems?
4. What do you think organizations did before computers were invented?
5. Discuss the memory systems you use. How do they improve your performance? What are the shortcomings of your existing systems? How can you improve them?
6. Describe the most “organized” person you know. Why is that person so organized? Why haven’t you adopted some of the same methods? Why do you think people differ in the extent to which they are organized?
7. Think about the last time you enrolled in a class. What data do you think were recorded for this transaction?
8. What roles do people play in organizational memory?
9. What do you think is the most important attribute of organizational memory? Justify your answer.
10. What is the difference between transaction processing and decision making?
11. When are data relevant?
12. Give some examples of specialized memories.
13. How can you measure the quality of a decision?
14. What is organizational culture? Can you name some organizations that have a distinctive culture?
15. What is hypertext? How does it differ from linear text? Why might hypertext be useful in an organizational memory system?
16. What is imaging? What are the characteristics of applications well suited for imaging?
17. What is an external memory? Why do organizations use external memories instead of building internal memories?
18. What is the common name used to refer to systems that help organizations remember knowledge?
19. What is a DSS? What is its role in organizational memory?
20. What are the major shortcomings of many data management systems? Which do you think is the most significant shortcoming?
21. What is the relationship between data, information, and knowledge?
22. Estimate how much data Netflix requires to store its many movies.
23. Using the Web, find some stories about firms using data management systems. You might enter keywords such as “database” and “business analytics” and access the sites of publications such as Computerworld. Identify the purpose of each system. How does the system improve organizational performance? What are the attributes of the technology that make it useful? Describe any trade-offs the organization might have made. Identify other organizations in which the same technology might be applied.
24. Make a list of the organizational memory systems identified in this chapter. Interview several people working in organizations. Ask them to indicate which organizational memory systems they use. Ask which system is most important and why. Write up your findings and your conclusion.



## 2 Information

*Effective information management must begin by thinking about how people use information—not with how people use machines.*

Davenport, T. H. (1994). Saving IT's soul: human-centered information management. *Harvard Business Review*, 72(2), 119-131.

### Learning objectives

Students completing this chapter will

- understand the importance of information to society and organizations;
- be able to describe the various roles of information in organizational change;
- be able to distinguish between soft and hard information;
- know how managers use information;
- be able to describe the characteristics of common information delivery systems;
- distinguish the different types of knowledge.

### Introduction

There are three characteristics of the early decades of the twenty-first century: global warming, high-velocity global change, and the emerging power of information organizations.<sup>3</sup> The need to create a sustainable civilization, the globalization of business, and the rise of China as a major economic power are major forces contributing to mass change. Organizations are undergoing large-scale restructuring as they attempt to reposition themselves to survive the threats and exploit the opportunities presented by these changes.

In the last few years, some very powerful and highly profitable information-based organizations have emerged. Apple and MS vie to be the world's largest company in terms of market value. Leveraging the iPhone and IOS, Apple has shown the power of information services to change an industry. Google has become a well-known global brand as it fulfills its mission "to organize the world's information and make it universally accessible and useful." Amazon is disrupting traditional retail. Facebook has digitized our social world, and Microsoft has dominated the office environment for decades. We gain further insights into the value of information by considering its role in civilization.

### A historical perspective

Humans have been collecting data to manage their affairs for several thousand years. In 3000 BCE, Mesopotamians recorded inventory details in cuneiform, an early script. Today's society transmits Exabytes of data every day as billions of people and millions of organizations manage their affairs. The dominant issue facing each type of economy has changed over time, and the collection and use of data has changed to reflect these concerns, as shown in the following table.

#### *Societal focus*

---

<sup>3</sup>In mid 2017, the five most valuable public companies were Apple, Google, Amazon, Microsoft, and Facebook. Furthermore, they are concentrated on the west coast of the U.S. in either Silicon Valley or Seattle.

Economy	Subsistence	Agricultural	Industrial	Service	Sustainable
Question	How to survive?	How to farm?	How to manage resources?	How to create customers?	How to reduce environmental impact?
Dominant issue	Survival				
		Production			
				Customer service	
					Sustainability
Key information systems	Gesturing Speech	Writing Calendar Money Measuring	Accounting ERP Project management	BPM CRM Analytics	Simulation Optimization Design

Agrarian society was concerned with productive farming, and an important issue was when to plant crops. Ancient Egypt, for example, based its calendar on the flooding of the Nile. The focus shifted during the industrial era to management of resources (e.g., raw materials, labor, logistics). Accounting, ERP, and project management became key information systems for managing resources. In the current service economy, the focus has shifted to customer creation. There is an oversupply of many consumer products (e.g., cars) and companies compete to identify services and product features that will attract customers. They are concerned with determining what types of customers to recruit and finding out what they want. As a result, we have seen the rise of business analytics and customer relationship management (CRM) to address this dominant issue. As well as creating customers, firms need to serve those already recruited. High quality service often requires frequent and reliable execution of multiple processes during manifold encounters with customers by a firm's many customer facing employees (e.g., a fast food store, an airline, a hospital). Consequently, business process management (BPM) has grown in importance since the mid 1990s when there was a surge of interest in business process reengineering.

We are in transition to a new era, sustainability, where attention shifts to assessing environmental impact because, after several centuries of industrialization, atmospheric CO<sub>2</sub> levels have become alarmingly high. We are also reaching the limits of the planet's resources as its population now exceeds seven billion people. As a result, a new class of application is emerging, such as environmental management systems and UPS's telematics project.<sup>4</sup> These new systems will also include, for example, support for understanding environmental impact through simulation of energy consuming and production systems, optimization of energy systems, and design of low impact production and customer service systems. Notice that dominant issues don't disappear but rather aggregate in layers, so tomorrow's business will be concerned with survival, production, customer service, and sustainability. As a result, a firm's need for data never diminishes, and each new layer creates another set of data needs. The flood will not subside and for most firms the data flow will need to grow significantly to meet the new challenge of sustainability.

A constant across all of these economies is organizational memory, or in its larger form, social memory. Writing and paper manufacturing developed about the same time as agricultural societies. Limited writing systems appeared about 30,000 years ago. Full writing systems, which have evolved in the last 5,000 years, made possible the technological transfer that enabled humanity to move from hunting and gathering to farming. Writing enables the recording of knowledge, and information can accumulate from one generation to the next. Before writing, knowledge was confined by the limits of human memory.

There is a need for a technology that can store knowledge for extended periods and support transport of

<sup>4</sup>Watson, R. T., Boudreau, M.-C., Li, S., & Levis, J. (2010). Telematics at UPS: En route to Energy Informatics. MISQ Executive, 9(1), 1-11.

written information. Storage medium has advanced from clay tablets (4000 BCE), papyrus (3500 BCE), and parchment (2000 BCE) to paper (100 CE.). Written knowledge gained great impetus from Johannes Gutenberg, whose achievement was a printing system involving movable metal type, ink, paper, and press. In less than 50 years, printing technology diffused throughout most of Europe. In the last century, a range of new storage media appeared (e.g., photographic, magnetic, and optical).

Organizational memories emerged with the development of large organizations such as governments, armies, churches, and trading companies. The growth of organizations during the industrial revolution saw a massive increase in the number and size of organizational memories. This escalation continued throughout the twentieth century.

The Internet has demonstrated that we now live in a borderless world. There is a free flow of information, investment, and industry across borders. Customers ignore national boundaries to buy products and services. In the borderless information age, the old ways of creating wealth have been displaced by intelligence, marketing, global reach, and education. Excelling in the management of data, information, and knowledge has become a prerequisite to corporate and national wealth.

*Wealth creation*

	The old	The new
Military power		Intelligence
Natural resources		Marketing
Population		Global reach
Industry		Education

This brief history shows the increasing importance of information. Civilization was facilitated by the discovery of means for recording and disseminating information. In our current society, organizations are the predominant keepers and transmitters of information.

### A brief history of information systems

Information systems has three significant eras. In the first era, information work was transported to the computer. For instance, a punched card deck was physically transported to a computer center, the information was processed, and the output physically returned to the worker as a printout.

*Information systems eras*

E ra	Focus	Period	Technology	Networks
1	Take i nformation work to the computer	1950s — mid-1970s	Batch	Few data networks
2	Take i nformation work to the employee	<b>2.1 Mid- 1970s</b>	H ost/termin al Cli ent/server	Spread of private networks
3	Take i nformation work to the customer and other st akeholders	mid-1990s <b>2.2 Mid- 1990s</b>  present	Brow ser/server	Public networks (Internet)

In the second era, private networks were used to take information work to the employee. Initially, these were time-sharing and host/terminal systems. IS departments were concerned primarily with creating systems for

use by an organization's employees when interacting with customers (e.g., a hotel reservation system used by call center employees) or for the employees of another business to transact with the organization (e.g., clerks in a hospital ordering supplies).

Era 3 starts with the appearance of the Web browser in the mid-1990s. The browser, which can be used on the public and global Internet, permits organizations to take information and information work to customers and other stakeholders. Now, the customer undertakes work previously done by the organization (e.g., making an airline reservation).

The scale and complexity of era 3 is at least an order of magnitude greater than that of era 2. Nearly every company has far more customers than employees. For example, UPS, with an annual investment of more than \$1 billion in information technology and over 400,000 employees, is one of the world's largest employers. However, there are 11 million customers, over 25 times the number of employees, who are today electronically connected to UPS.

Era 3 introduced direct electronic links between a firm and its stakeholders, such as investors and citizens. In the earlier eras, intermediaries often communicated with stakeholders on behalf of the firm (e.g., a press release to the news media). These messages could be filtered and edited, or sometimes possibly ignored, by intermediaries. Now, organizations can communicate directly with their stakeholders via the Web, e-mail, social media. Internet technologies offer firms a chance to rethink their goals vis-à-vis each stakeholder class and to use Internet technology to pursue these goals.

This brief history leads to the conclusion that the value IS creates is determined by whom an organization can reach, how it can reach them, and where and when it can reach them.

- **Whom.** Whom an organization can contact determines whom it can influence, inform, or transfer work to. For example, if a hotel can be contacted electronically by its customers, it can promote online reservations (transfer work to customers), and reduce its costs.
- **How.** How an organization reaches a stakeholder determines the potential success of the interaction. The higher the bandwidth of the connection, the richer the message (e.g., using video instead of text), the greater the amount of information that can be conveyed, and the more information work that can be transferred.
- **Where.** Value is created when customers get information directly related to their current location (e.g., a navigation system) and what local services they want to consume (e.g., the nearest Italian restaurant).
- **When.** When a firm delivers a service to a client can greatly determine its value. Stockbrokers, for instance, who can inform clients immediately of critical corporate news or stock market movements are likely to get more business.

## Information characteristics

Three useful concepts for describing information are hardness, richness, and class. Information hardness is a subjective measure of the accuracy and reliability of an item of information. Information richness describes the concept that information can be rich or lean depending on the information delivery medium. Information class groups types of information by their key features.

### Information hardness

In 1812, the Austrian mineralogist Friedrich Mohs proposed a scale of hardness, in order of increasing relative hardness, based on 10 common minerals. Each mineral can scratch those with the same or a lower number, but cannot scratch higher-numbered minerals.

A similar approach can be used to describe information. Market information, such as the current price of gold, is the hardest because it is measured extremely accurately. There is no ambiguity, and its measurement is highly reliable. In contrast, the softest information, which comes from unidentified sources, is rife with uncertainty.

*An information hardness scale*

Mineral	S cale	Data
Talc	1	Unidentified source—rumors, gossip, and hearsay
Gypsum	2	Identified non-expert source—opinions, feelings, and ideas
Calcite	3	Identified expert source—predictions, speculations, forecasts, and estimates
Fluorite	4	Unsworn testimony—explanations, justifications, assessments, and interpretations
Apatite	5	Sworn testimony—explanations, justifications, assessments, and interpretations
Or thoclase	6	Budgets, formal plans
Quartz	7	News reports, non-financial data, industry statistics, and surveys
Topaz	8	Unaudited financial statements, and government statistics
Corundum	9	Audited financial statements
Diamond	10	Stock exchange and commodity market data

Audited financial statements are in the corundum zone. They are measured according to standard rules (known as “generally accepted accounting principles”) that are promulgated by national accounting societies. External auditors monitor the application of these standards, although there is generally some leeway in their application and sometimes multiple standards for the same item. The use of different accounting principles can lead to different profit and loss statements. As a result, the information in audited financial statements has some degree of uncertainty.

There are degrees of hardness within accounting systems. The hardest data are counts, such as units sold or customers served. These are primary measures of organizational performance. Secondary measures, such as dollar sales and market share, are derived from counts. Managers vary in their preference for primary and secondary measures. Operational managers opt for counts for measuring productivity because they are uncontaminated by price changes and currency fluctuations. Senior managers, because their focus is on financial performance, select secondary measures.

The scratch test provides a convenient and reliable method of assessing the hardness of a mineral. Unfortunately, there is no scratch test for information. Managers must rely on their judgment to assess information hardness.

Although managers want hard information, there are many cases when it is not available. They compensate by seeking information from several different sources. Although this approach introduces redundancy, this is precisely what the manager seeks. Relatively consistent information from different sources is reassuring.

## Information richness

Information can be described as rich or lean. It is richest when delivered face-to-face. Conversation permits immediate feedback for verification of meaning. You can always stop the other speaker and ask, “What do you mean?” Face-to-face information delivery is rich because you see the speaker’s body language, hear the tone of voice, and natural language is used. A numeric document is the leanest form of information. There is no opportunity for questions, no additional information from body movements and vocal tone. The information richness of some communication media is shown in the following table.

Richest				Leanest
Face-to-face	Telephone	Personal documents	Impersonal written documents	Numeric documents

Managers seek rich information when they are trying to resolve equivocality or ambiguity. It means that managers cannot make sense of a situation because they arrive at multiple, conflicting interpretations of the information. An example of an equivocal situation is a class assignment where some of the instructions are missing and others are contradictory (of course, this example is an extremely rare event).

Equivocal situations cannot be resolved by collecting more information, because managers are uncertain about what questions to ask and often a clear answer is not possible. Managers reduce equivocality by sharing their interpretations of the available information and reaching a collective understanding of what the information means. By exchanging opinions and recalling their experiences, they try to make sense of an ambiguous situation.

Many of the situations that managers face each day involve a high degree of equivocality. Formal organizational memories, such as databases, are not much help, because the information they provide is lean. Many managers rely far more on talking with colleagues and using informal organizational memories such as social networks.

Data management is almost exclusively concerned with administering the formal information systems that deliver lean information. Although this is their proper role, data managers must realize that they can deliver only a portion of the data required by decision makers.

## Information classes

Information can be grouped into four classes: content, form, behavior, and action. Until recently, most organizational information fell into the first category.

### *Information classes*

Class	Description
Content	Quantity, location, and types of items
Form	Shape and composition of an object
Behavior	Simulation of a physical object
Action	Creation of action (e.g., industrial robots)

Content information records details about quantity, location, and types of items. It tends to be historical in nature and is traditionally the class of information collected and stored by organizations. The content information of a car would describe its model number, color, price, options, horsepower, and so forth. Hundreds of bytes of data may be required to record the full content information of a car. Typically, content data are captured by a TPS.

Form information describes the shape and composition of an object. For example, the form information of a car would define the dimensions and composition of every component in the car. Millions of bytes of data are needed to store the form of a car. CAD/CAM systems are used to create and store form information.

Behavior information is used to predict the behavior of a physical object using simulation techniques, which typically require form information as input. Massive numbers of calculations per second are required to

<sup>5</sup>Daft, R. L., & Lengel, R. H. (1986). Organizational information requirements, media richness, and structural design. *Management Science*, 32(5), 554-571.

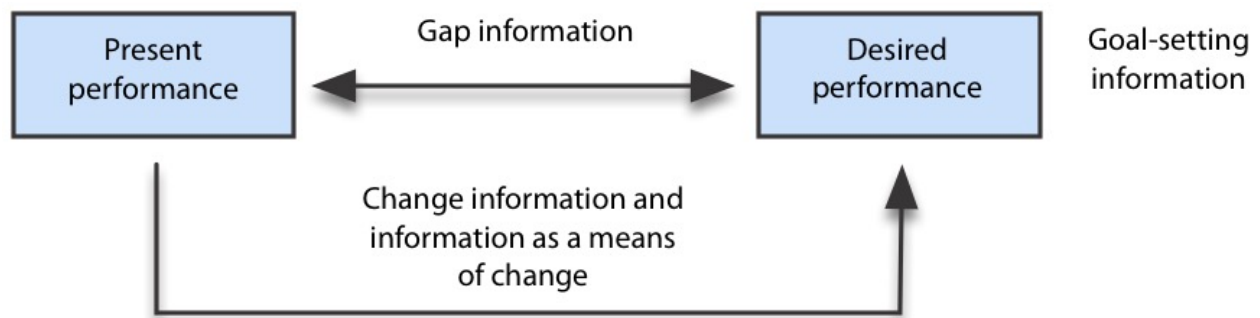
simulate behavior. For example, simulating the flight of a new aircraft design may require trillions of computations. Behavior information is often presented visually because the vast volume of data generated cannot be easily processed by humans in other formats.

Action information enables the instantaneous creation of sophisticated action. Industrial robots take action information and manufacture parts, weld car bodies, or transport items. Antilock brakes are an example of action information in use.

## Information and organizational change

Organizations are goal-directed. They undergo continual change as they use their resources, people, technology, and financial assets to reach some future desired outcome. Goals are often clearly stated, such as to make a profit of \$100 million, win a football championship, or decrease the government deficit by 25 percent in five years. Goals are often not easily achieved, however, and organizations continually seek information that supports goal attainment. The information they seek falls into three categories: goal setting, gap, and change.

*Organizational information categories*



The emergence of an information society also means that information provides dual possibilities for change. Information is used to plan change, and information is a medium for change.

## Goal-setting information

Organizations set goals or levels of desired performance. Managers need information to establish goals that are challenging but realistic. A common approach is to take the previous goal and stretch it. For example, a company with a 15 percent return on investment (ROI) might set the new goal at 17 percent ROI. This technique is known as “anchoring and adjusting.” Prior performance is used as a basis for setting the new performance standards. The problem with anchoring and adjusting is that it promotes incremental improvement rather than radical change because internal information is used to set performance standards. Some organizations have turned to external information and are using benchmarking as a source of information for goal setting.

## Planning

Planning is an important task for senior managers. To set the direction for the company, they need information about consumers’ potential demands and social, economic, technical, and political conditions. They use this information to determine the opportunities and threats facing the organization, thus permitting them to take advantage of opportunities and avoid threats.

Most of the information for long-term planning comes from sources external to the company. There are think tanks that analyze trends and publish reports on future conditions. Journal articles and books can

be important sources of information about future events. There also will be a demand for some internal information to identify trends in costs and revenues. Major planning decisions, such as building a new plant, will be based on an analysis of internal data (use of existing capacity) and external data (projected customer demand).

## **Benchmarking**

Benchmarking establishes goals based on best industry practices. It is founded on the Japanese concept of *dantotsu*, striving to be the best of the best. Benchmarking is externally directed. Information is sought on those companies, regardless of industry, that demonstrate outstanding performance in the areas to be benchmarked. Their methods are studied, documented, and used to set goals and redesign existing practices for superior performance.

Other forms of external information, such as demographic trends, economic forecasts, and competitors' actions can be used in goal setting. External information is valuable because it can force an organization to go beyond incremental goal setting.

Organizations need information to identify feasible, motivating, and challenging goals. Once these goals have been established, they need information on the extent to which these goals have been attained.

## **Gap information**

Because goals are meant to be challenging, there is often a gap between actual and desired performance. Organizations use a number of mechanisms to detect a gap and gain some idea of its size. Problem identification and scorekeeping are two principal methods of providing gap information.

### **Problem identification**

Business conditions are continually changing because of competitors' actions, trends in consumer demand, and government actions. Often these changes are reflected in a gap between expectations and present performance. This gap is known as a problem.

To identify problems, managers use exception reports, which are generated only when conditions vary from the established goal or standard. Once a potential problem has been identified, managers collect additional information to confirm that a problem really exists. Once management has been alerted, the information delivery system needs to shift into high gear. Managers will request rapid delivery of ad hoc reports from a variety of sources. The ideal organizational memory system can adapt smoothly to deliver appropriate information quickly.

### **Scorekeeping**

Keeping track of the score provides gap information. Managers ask many questions: How many items did we make yesterday? What were the sales last week? Has our market share increased in the last year? They establish measurement systems to track variables that indicate whether organizational performance is on target. Keeping score is important; managers need to measure in order to manage. Also, measurement lets people know what is important. Once a manager starts to measure something, subordinates surmise that this variable must be important and pay more attention to the factors that influence it.

There are many aspects of the score that a manager can measure. The overwhelming variety of potentially available information is illustrated by the sales output information that a sales manager could track. Sales input information (e.g., number of service calls) can also be measured, and there is qualitative information to be considered. Because of time constraints, most managers are forced to limit their attention to 10 or



fewer key variables singled out by the critical success factors (CSF) method.<sup>6</sup> Scorekeeping information is usually fairly stable.

#### *Sales output tracking information*

Category	Example
Orders	Number of current customers Average order size Batting average (orders to calls)
Sales volume	Dollar sales volume Unit sales volume By customer type By product category Translated to market share
Margins	Quota achieved Gross margin Net profit By customer type By product
Customers	Number of new accounts Number of lost accounts Percentage of accounts sold Number of accounts overdue Dollar value of receivables Collections of receivables

## Change information

Accurate change information is very valuable because it enables managers to predict the outcome of various actions to close a gap with some certainty. Unfortunately, change information is usually not very precise, and there are many variables that can influence the effect of any planned change. Nevertheless, organizations spend a great deal of time collecting information to support problem solving and planning.

## Problem solution

Once a concern has been identified, a manager seeks to find its cause. A decrease in sales could be the result of competitors introducing a new product, an economic downturn, an ineffective advertising campaign, or many other reasons. Data can be collected to test each of these possible causes. Additional data are usually required to support analysis of each alternative. For example, if the sales decrease has been caused by an economic recession, the manager might use a decision support system (DSS) to analyze the effect of a price decrease or a range of promotional activities.

## Information as a means of change

The emergence of an information society means that information can be used as a means of changing an organization's performance. Corporations can create information-based products and services, adding information to products, and using information to enhance existing performance or gain a competitive advantage. Further insights into the use of information as a change agent are gained by examining marketing, customer service, and empowerment.

<sup>6</sup>Rockart, J.F. (1982). The changing role of the information systems executive: a critical success factors perspective. *Sloan Management Review*, 24(1), 3-13.

## Marketing

Marketing is a key strategy for changing organizational performance by increasing sales. Information has become an important component in marketing. Airlines and retailers have established frequent flyer and buyer programs to encourage customer loyalty and gain more information about customers. These systems are heavily dependent on database technology because of the massive volume of data that must be stored. Indeed, without database technology, some marketing strategies could never be implemented.

Database technology offers the opportunity to change the very nature of communications with customers. Broadcast media have been the traditional approach to communication. Advertisements are aired on television, placed in magazines, or displayed on a Web site. Database technology can be used to address customers directly. No longer just a mailing list, today's database is a memory of customer relationships, a record of every message and response between the firm and a customer. Some companies keep track of customers' preferences and customize advertisements to their needs. Leading online retailers now send customers only those emails for products for which they estimate there is a high probability of a purchase. For example, a customer with a history of buying jazz music is sent an email promoting jazz recordings instead of one featuring classical music.

The Web has significantly enhanced the value of database technology. Many firms now use a combination of a Web site and a DBMS to market products and service customers.

## Customer Service

Many American business leaders rank customer service as their most important goal for organizational success. Many of the developed economies are service driven. In the United States, services account for nearly 80 percent of the GNP and most of the new jobs. Many companies now compete for customers by offering superior customer service. Information is frequently a key to this improved service.

### *Skill builder*

For many businesses, information is the key to high-quality customer service. Thus, some firms use information, and thus customer service, as a key differentiating factor, while others might compete on price. Compare the electronics component of Web sites Amazon and Walmart. How do they use price and information to compete? What are the implications for data management if a firm uses information to compete?

## Empowerment

Empowerment means giving employees greater freedom to make decisions. More precisely, it is sharing with frontline employees

- information about the organization's performance;
- rewards based on the organization's performance;
- knowledge and information that enable employees to understand and contribute to organizational performance;
- power to make decisions that influence organizational direction and performance.

Notice that information features prominently in the process. A critical component is giving employees access to the information they need to perform their tasks with a high degree of independence and discretion. Information is empowerment. By linking employees to organizational memory, data managers play a pivotal role in empowering people. Empowerment can contribute to organizational performance by increasing the quality of products and services. Together, empowerment and information are mechanisms of planned change.

## Information and managerial work

Because managers frequently use data management systems in their normal activities as a source of information about change and as a means of implementing change, it is crucial for systems designers to understand how managers work. Failure to take account of managerial behavior can result in a system that is technically sound but rarely used because it does not fit the social system.

Studies over several decades reveal a very consistent pattern: Managerial work is very fragmented. Managers spend an average of 10 minutes on any task, and their day is organized into brief periods of concentrated attention to a variety of tasks. They work unrelentingly and are frequently interrupted by unexpected disturbances. Managers are action oriented and rely on intuition and judgment far more than contemplative analysis of information.

Managers strongly prefer oral communication. They spend a great deal of time conversing directly or by telephone. Managers use interpersonal communication to establish networks, which they later use as a source of information and a way to make things happen. The continual flow of verbal information helps them make sense of events and lets them feel the organizational pulse.

Managers rarely use formal reporting systems. They do not spend their time analyzing computer reports or querying databases but resort to formal reports to confirm impressions, should interpersonal communications suggest there is a problem. Even when managers are provided with a purpose-built, ultra-friendly executive information system, their behavior changes very little. They may access a few screens during the day, but oral communication is still their preferred method of data gathering and dissemination.

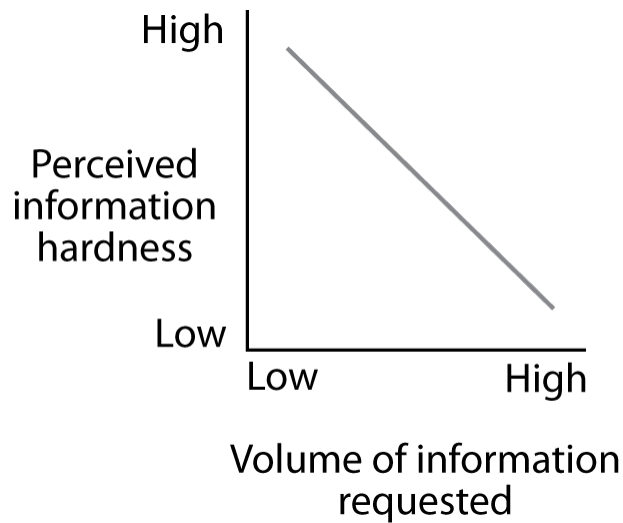
## Managers' information requirements

Managers have certain requirements of the information they receive. These expectations should shape a data management system's content and how data are processed.

Managers expect to receive information that is useful for their current task under existing business conditions. Unfortunately, managerial tasks can change rapidly. The interlinked, global, economic environment is highly turbulent. Since managers' expectations are not stable, the information delivery system must be sufficiently flexible to meet changing requirements.

Managers' demands for information vary with their perception of its hardness; they require only one source that scores 10 on the information hardness scale. The Nikkei Index at the close of today's market is the same whether you read it in the *Asian Wall Street Journal* or *The Western Australian* or hear it on CNN. As perceived hardness decreases, managers demand more information, hoping to resolve uncertainty and gain a more accurate assessment of the situation. When the reliability of a source is questionable, managers seek confirmation from other sources. If a number of different sources provide consistent information, a manager gains confidence in the information's accuracy.

*Relationship of perceived information hardness to volume of information requested*



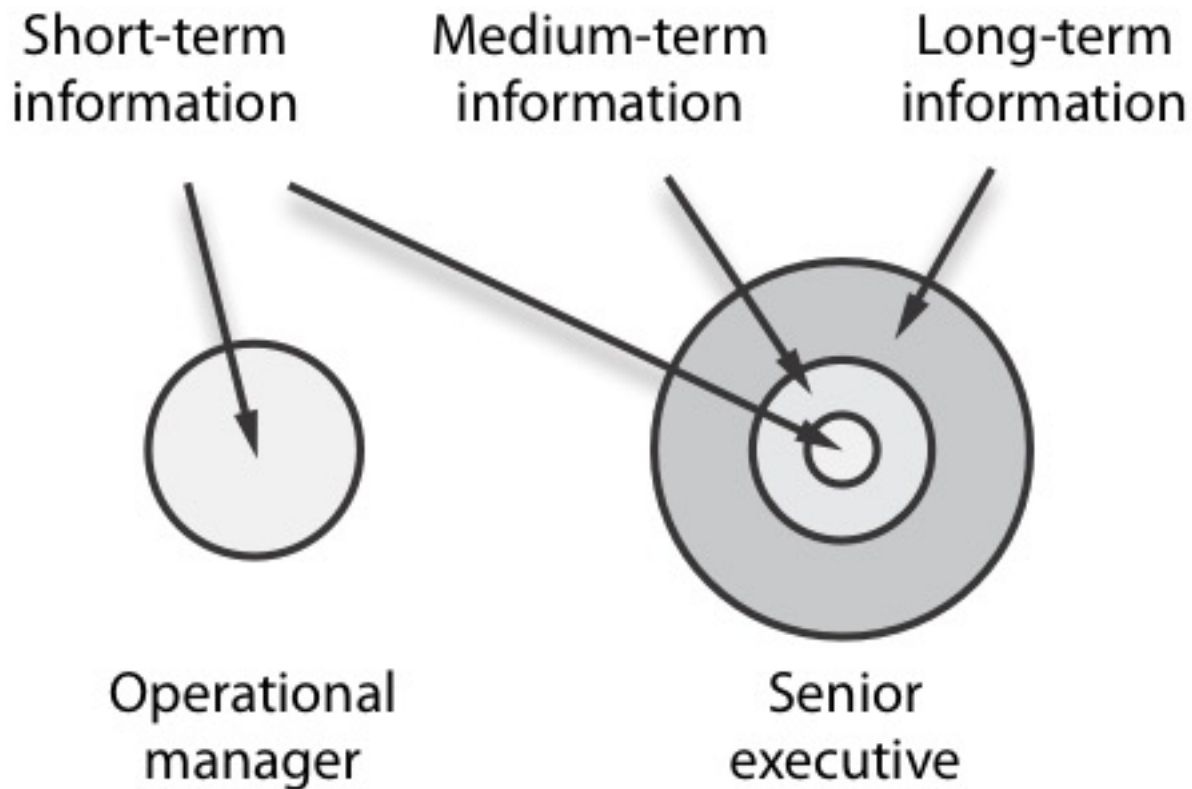
It is not unusual, therefore, to have a manager seek information from a database, a conversation with a subordinate, and a contact in another organization. If a manager gets essentially the same information from each source, she or he has the confirmation sought. This means each data management system should be designed to minimize redundancy, but different components of organizational memory can supply overlapping information.

### **Managers' needs for information vary accordingly with responsibilities**

Operational managers need detailed, short-term information to deal with well-defined problems in areas such as sales, service, and production. This information comes almost exclusively from internal sources that report recent performance in the manager's area. A sales manager may get weekly reports of sales by each direct report.

As managers move up the organizational hierarchy, their information needs both expand and contract. They become responsible for a wider range of activities and are charged with planning the future of the organization, which requires information from external sources on long-term economic, demographic, political, and social trends. Despite this long-term focus, top-level managers also monitor short-term, operational performance. In this instance, they need less detail and more summary and exception reports on a small number of key indicators. To avoid information overload as new layers of information needs are added, the level of detail on the old layers naturally must decline, as illustrated in the following figure.

*Management level and information need*



### Information satisficing

Because managers face making many decisions in a short period, most do not have the time or resources to collect and interpret all the information they need to make the best decision. Consequently, they are often forced to satisfice. That is, they accept the first satisfactory decision they discover. They also satisfice in their information search, collecting only enough information to make a satisfactory decision.

Ultimately, information satisficing leads to lower-quality decision making. If, however, information systems can accelerate delivery and processing of the right information, then managers should be able to move beyond selecting the first satisfactory decision to selecting the best of several satisfactory decisions.

### Information delivery systems

Most organizations have a variety of delivery systems to provide information to managers. Developed over many years, these systems are integrated, usually via a Web site, to give managers better access to information. There are two aspects to information delivery. First, there is a need for software that accesses an organizational memory, extracts the required data, and formats it for presentation. We can use the categories of organizational memories introduced in Chapter 1 to describe the software side of information delivery systems. The second aspect of delivery is the hardware that gets information from a computer to the manager.

#### *Information delivery systems software*

Organizational memory	Delivery systems
People	Conversation
	E-mail

Organizational memory	Delivery systems
	Meeting
	Report
	Groupware
Files	Management information system (MIS)
Documents	Web browser
	E-mail attachment
Images	Image processing system (IPS)
Graphics	Computer aided design (CAD)
	Geographic information system (GIS)
Voice	Voice mail
	Voice recording system
Mathematical model	Decision support system (DSS)
Decisions	Conversation
	E-mail
	Meeting
	Report
	Groupware

Software is used to move data to and from organizational memory. There is usually tight coupling between software and the format of an organizational memory. For example, a relational database management system can access tables but not decision-support models. This tight coupling is particularly frustrating for managers who often want integrated information from several organizational memories. For example, a sales manager might expect a monthly report to include details of recent sales (from a relational database) to be combined with customer comments (from email messages to a customer service system). A quick glance at the preceding table shows that there are many different information delivery systems. We will discuss each of these briefly to illustrate the lack of integration of organizational memories.

### Verbal exchange

Conversations, meetings, and oral reporting are commonly used methods of information delivery. Indeed, managers show a strong preference for verbal exchange as a method for gathering information. This is not surprising because we are accustomed to oral information. This is the way we learned for thousands of years as a preliterate culture. Only recently have we learned to make decisions using spreadsheets and computer reports.

### Voice mail

Voice mail is useful for people who do not want to be interrupted. It supports asynchronous communication; that is, the two parties to the conversation are not simultaneously connected. Voice-mail systems also can store many prerecorded messages that can be selectively replayed using a phone's keypad. Organizations use this feature to support standard customer queries.

### Electronic mail

Email is an important system of information delivery. It too supports asynchronous messaging, and it is less costly than voice mail for communication. Many documents are exchanged as attachments to email.

## Formal report

Formal reports have a long history in organizations. Before electronic communication, they were the main form of information delivery. They still have a role in organizations because they are an effective method of integrating information of varying hardness and from a variety of organizational memories. For example, a report can contain text, tables, graphics, and images.

Formal reports are often supplemented by a verbal presentation of the key points in the report. Such a presentation enhances the information delivered, because the audience has an opportunity to ask questions and get an immediate response.

## Meetings

Because managers spend 30-80 percent of their time in meetings, these are a key source of information.

## Groupware

Since meetings occupy so much managerial time and in many cases are poorly planned and managed, organizations are looking for improvements. Groupware is a general term applied to a range of software systems designed to improve some aspect of group work. It is excellent for tapping soft information and the informal side of organizational memory.

**Management information system** Management information systems are a common method of delivering information from data management systems. A preplanned query is often used to extract the data. Managers who have developed some skills in using a query language might create custom reports as needed.

Preplanned reports often contain too much or too detailed information, because they are designed in anticipation of a manager's needs. Customized reports do not have these shortcomings, but they are often more expensive and time consuming because they need to be prepared by an analyst.

**Web** Word processing, desktop publishing, or HTML editors are used for preparing documents that are disseminated by placing them on a Web server for convenient access and sharing.

**Image processing system** An image processing system (IPS) captures data using a scanner to digitize an image. Images in the form of letters, reports, illustrations, and graphics have always been an important type of organizational memory. An IPS permits these forms of information to be captured electronically and disseminated.

**Computer-aided design** Computer-aided design (CAD) is used extensively to create graphics. For example, engineers use CAD in product design, and architects use it for building design. These plans are a part of organizational memory for designers and manufacturers.

**Geographic information system** Many cities use a geographic information system (GIS) to store graphical data about roads, utilities, and services. This information is another form of organizational memory.

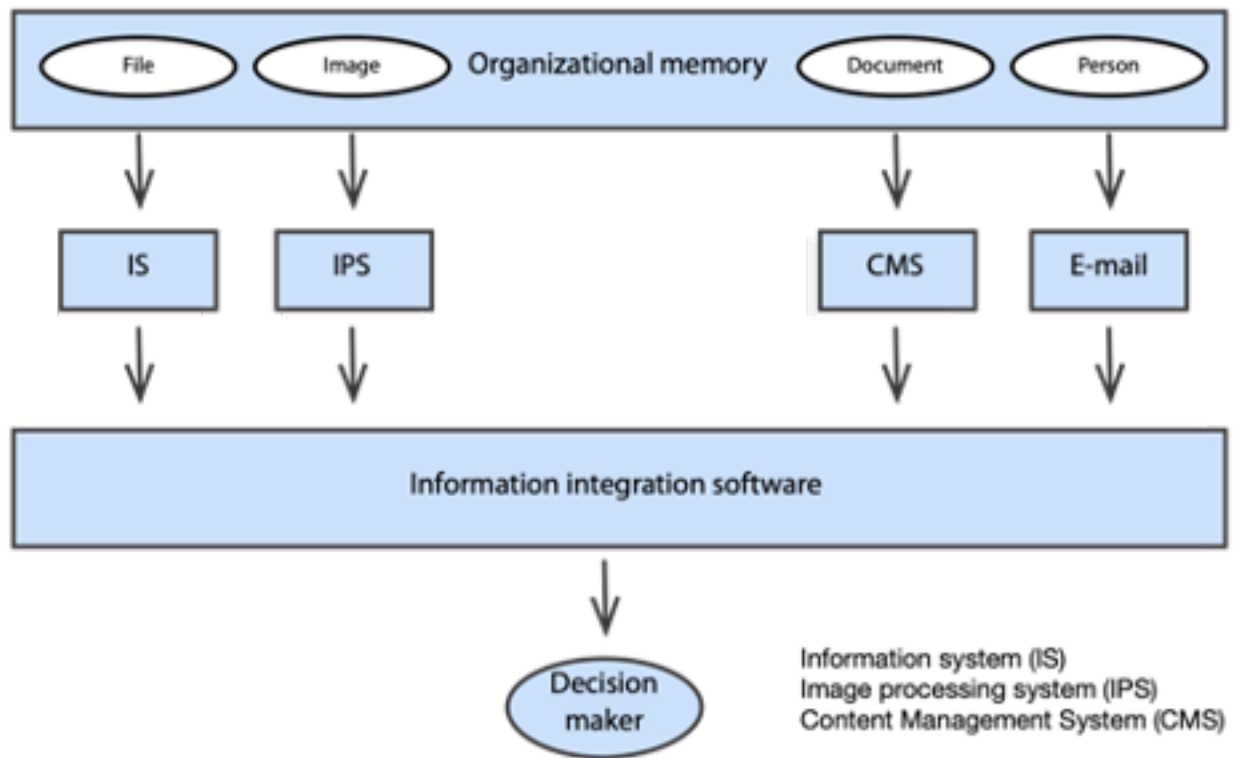
**Decision support system** A decision support system (DSS) is frequently a computer-based mathematical model of a problem. DSS software, available in a range of packages, permits the model and data to be retrieved and executed. Model parameters can be varied to investigate alternatives.

## Information integration

A fundamental problem for most organizations is that their memory is fragmented across a wide variety of formats and technologies. Too frequently, there is a one-to-one correspondence between an organizational memory for a particular functional area and the software delivery system. For example, sales information is delivered by the sales system and production information by the production system. This is not a very desirable situation because managers want all the information they need, regardless of its source or format, amalgamated into a single report.

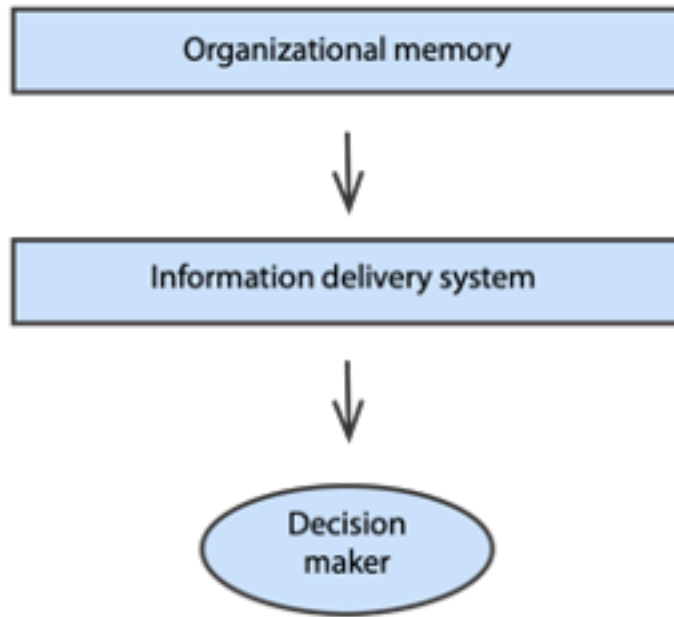
A common solution is to develop software, such as a Web application, that integrates information from a variety of delivery systems. An important task of this apl is to integrate and present information from multiple organizational memories. Recently, some organizations have created vast integrated data stores— data warehouses— that are organized repositories of organizational data.

*Information integration - the present situation*



*The ideal organizational memory and information delivery system*





## Knowledge

The performance of many organizations is determined more by their cognitive capabilities and knowledge than their physical assets. A nation's wealth is increasingly a result of the knowledge and skills of its citizens, rather than its natural resources and industrial plants. Currently, about 85 percent of all jobs in America and 80 percent of those in Europe are knowledge-based.

An organization's knowledge, in order of increasing importance, is

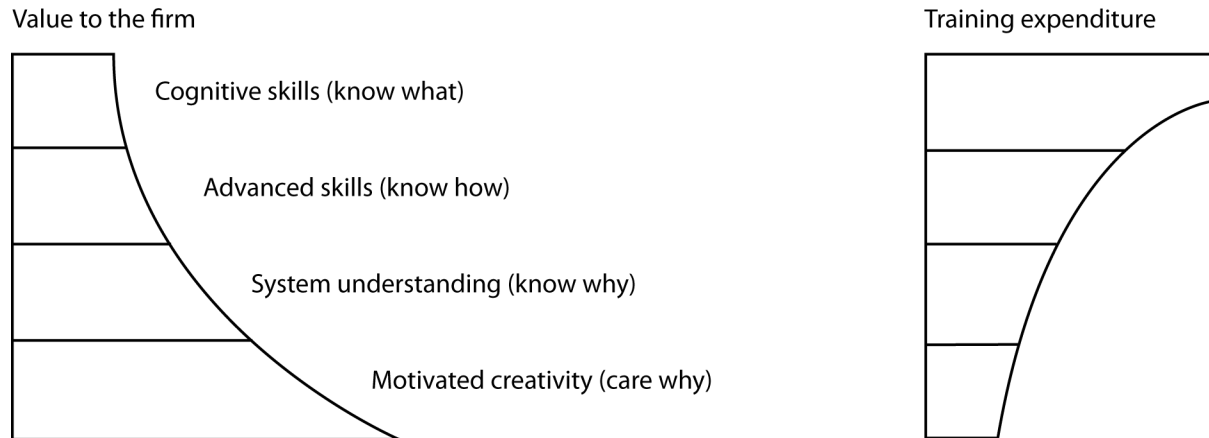
- cognitive knowledge (know what);
- advanced skills (know how);
- system understanding and trained intuition (know why);
- self-motivated creativity (care why).

This text illustrates the different types of knowledge. You will develop cognitive knowledge in Section 4, when you learn about data architectures and implementations. For example, knowing what storage devices can be used for archival data is cognitive knowledge. Section 2, data modeling and SQL, develops advanced skills because, upon completion of that section, you will know how to model data and write SQL queries. The first two chapters expand your understanding of the influence of organizational memory on organizational performance. You need to know why you should learn data management skills. Managers know when and why to apply technology, whereas technicians know what to apply and how to apply it. Finally, you are probably an IS major, and your coursework is inculcating the values and norms of the IS profession so that you care why problems are solved using information technology.

Organizations tend to spend more on developing cognitive skills than they do on fostering creativity. This is, unfortunately, the wrong priority. Returns are likely to be much greater when higher-level knowledge skills are developed. Well-managed organizations place more attention on creating know why and care why skills because they recognize that knowledge is a key competitive weapon. Furthermore, these firms have learned that knowledge grows, often very rapidly, when shared. Knowledge is like a communication network whose potential benefit grows exponentially as the nodes within the network grow arithmetically. When knowledge

is shared within the organization, or with customers and suppliers, it multiplies as each person receiving knowledge imparts it to someone else in the organization or a business partner.

*Skills values vs. training expenditures (Quinn et al., 1996)*



There are two types of knowledge: explicit and tacit. Explicit knowledge is codified and transferable. This textbook is an example of explicit knowledge. Knowledge about how to design databases has been formalized and communicated with the intention of transferring it to you, the reader. Tacit knowledge is personal knowledge, experience, and judgment that is difficult to codify. It is more difficult to transfer tacit knowledge because it resides in people's minds. Usually, the transfer of tacit knowledge requires the sharing of experiences. In learning to model data, the subject of the next section, you will quickly learn how to represent an entity, because this knowledge is made explicit. However, you will find it much harder to model data, because this skill comes with practice. Ideally, you should develop several models under the guidance of an experienced modeler, such as your instructor, who can pass on his or her tacit knowledge.

## Summary

Information has become a key foundation for organizational growth. The information society is founded on computer and communications technology. The accumulation of knowledge requires a capacity to encode and share information. Hard information is very exact. Soft information is extremely imprecise. Rich information exchange occurs in face-to-face conversation. Numeric reports are an example of lean information. Organizations use information to set goals, determine the gap between goals and achievements, determine actions to reach goals, and create new products and services to enhance organizational performance. Managers depend more on informal communication systems than on formal reporting systems. They expect to receive information that meets their current, ever changing needs. Operational managers need short-term information. Senior executives require mainly long-term information but still have a need for both short- and medium-term information. When managers face time constraints, they collect only enough information to make a satisfactory decision. Organizational memory should be integrated to provide one interface to an organization's information stores.

### Key terms and concepts

Advanced skills (know how)	Information organization
Benchmarking	Information requirements
Change information	Information richness
Cognitive knowledge (know what)	Information satisficing
Empowerment	Information society
Explicit knowledge	Knowledge
Gap information	Managerial work
Global change	Organizational change

Key terms and concepts	
Goal-setting information	Phases of civilization
Information as a means of change	Self-motivated creativity (care why)
Information delivery systems	Social memory
Information hardness	System understanding (know why)
Information integration	Tacit knowledge

## References and additional readings

Quinn, J. B., P. Anderson, and S. Finkelstein. 1996. Leveraging intellect. *Academy of Management Executive* 10 (3):7-27.

## Exercises

1. From an information perspective, what is likely to be different about the customer service era compared to the forthcoming sustainability era?
2. How does an information job differ from an industrial job?
3. Why was the development of paper and writing systems important?
4. What is the difference between soft and hard information?
5. What is the difference between rich and lean information exchange?
6. What are three major types of information connected with organizational change?
7. What is benchmarking? When might a business use benchmarking?
8. What is gap information?
9. Give some examples of how information is used as a means of change.
10. What sorts of information do senior managers want?
11. Describe the differences between the way managers handle hard and soft information.
12. What is information satisficing?
13. Describe an incident where you used information satisficing.
14. Give some examples of common information delivery systems.
15. What is a GIS? Who might use a GIS?
16. Why is information integration a problem?
17. How “hard” is an exam grade?
18. Could you develop a test for the hardness of a piece of information?
19. Is very soft information worth storing in formal organizational memory? If not, where might you draw the line?
20. If you had just failed your database exam, would you use rich or lean media to tell a parent or spouse about the result?

21. Interview a businessperson to determine his or her firm's critical success factors (CSFs). Remember, a CSF is something the firm must do right to be successful. Generally a firm has about seven CSFs. For the firm's top three CSFs, identify the information that will measure whether the CSF is being achieved.
22. If you were managing a fast-food store, what information would you want to track store performance? Classify this information as short-, medium-, or long-term.
23. Interview a manager. Identify the information that person uses to manage the company. Classify this information as short-, medium-, or long-term information. Comment on your findings.
24. Why is organizational memory like a data warehouse? What needs to be done to make good use of this data warehouse?
25. What information are you collecting to help determine your career or find a job? What problems are you having collecting this information? Is the information mainly hard or soft?
26. What type of knowledge should you gain in a university class?
27. What type of knowledge is likely to make you most valuable?

## Section 2 Data Modeling and SQL

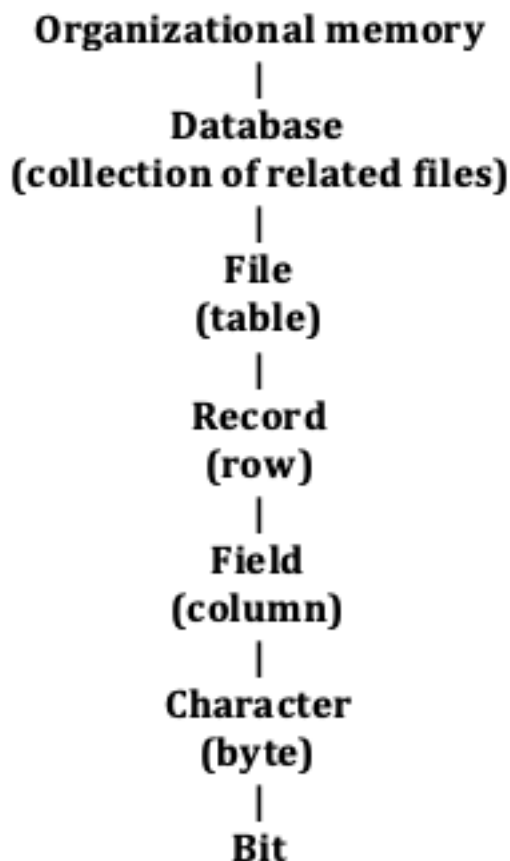
*It is a capital mistake to theorize before one has data.*

Sir Arthur Conan Doyle, “A Scandal in Bohemia,” *The Adventures of Sherlock Holmes*, 1891

The application backlog, a large number of requests for new information systems, has been a recurring problem in many organizations for decades. The demand for new information systems and the need to maintain existing systems have usually outstripped available information systems skills. The application backlog, unfortunately, is not a new problem. In the 1970s, Codd laid out a plan for improving programmer productivity and accelerating systems development by improving the management of data. Codd’s **relational model**, designed to solve many of the shortcomings of earlier systems, is currently the most popular database model.

This section develops two key skills—data modeling and query formation—that are required to take advantage of the relational model. We concentrate on the design and use of relational databases. This very abrupt change in focus is part of the plan to give you a dual understanding of data management. Section 1 is the managerial perspective, whereas this section covers technical skills development. Competent data managers are able to accommodate both views and apply whichever (or some blend of the two) is appropriate.

In Chapter 1, many forms of organizational memory were identified, and in this section we focus on files and their components. Thus, only the files branch of organizational memory is detailed in the following figure.



A collection of related files is a **database**. Describing the collection of files as related means that it has a common purpose (e.g., data about students). Sometimes files are also called tables, and there are synonyms for some other terms (the alternative names are shown in parentheses). Files contain **records** (or rows). Each record contains the data for one instance of the data the file stores. For example, if the file stores data about students, each record will contain data about a single student. Records have **fields** (or columns) that

store the fine detail of each instance (e.g., student's first name, last name, and date of birth). Fields are composed of **characters** (a, b, c,..., 1, 2, 3,..., %, \$, #,..., A, B, etc.). A **byte**, a unit of storage sufficient to store a single letter (in English) or digit, consists of a string of eight contiguous **bits** or binary digits.

The data management hierarchy stimulates three database design questions:

- What collection of files should the database contain?
- How are these files related?
- What fields should each record in the file contain?

The first objective of this section is to describe data modeling, a technique for answering the three questions. Data modeling helps you to understand the structure and meaning of data, which is necessary before a database can be created. Once a database has been designed, built, and loaded with data, the aim is to deploy it to satisfy management's requests for information. Thus, the second objective is to teach you to query a relational database. The learning of modeling and querying will be intertwined, making it easier to grasp the intent of database design and to understand why data modeling is so critical to making a database an effective tool for managerial decision making.

Chapter 3 covers modeling a single entity and querying a single-table database. This is the simplest database that can be created. As you will soon discover, a **data model** is a graphical description of the components of a database. One of these components is an entity, some feature of the real world about which data must be stored. This section also introduces the notions of a **data definition language** (DDL), which is used to describe a database, and a **data manipulation language** (DML), which is used to maintain and query a database. Subsequent chapters in this section cover advanced data modeling concepts and querying capabilities.

### 3 The Single Entity

*I want to be alone.*

Attributed to Greta Garbo

#### Learning Objectives

Students completing this chapter will be able to

- model a single entity;
- define a single database;
- write queries for a single-table database.

#### The relational model

The relational model introduced by Codd in 1970 is the most popular technology for managing large collections of data. In this chapter, the major concepts of the relational model are introduced. Extensive coverage of the relational model is left until Chapter 8, by which time you will have sufficient practical experience to appreciate fully its usefulness, value, and elegance.

A **relation**, similar to the mathematical concept of a set, is a two-dimensional table arranged in rows and columns. This is a very familiar idea. You have been using tables for many years. A **relational database** is a collection of relations, where **relation** is a mathematical term for a table. One row of a table stores

details of one observation, instance, or case of an item about which facts are retained—for example, one row for details of a particular student. All the rows in a table store data about the same type of item. Thus, a database might have one table for student data and another table for class data. Similarly, each column in the table contains the same type of data. For example, the first column might record a student’s identification number. A key database design question is to decide what to store in each table. What should the rows and columns contain?

In a relational database, each row must be uniquely identified. There must be a **primary key**, such as student identifier, so that a particular row can be designated. The use of unique identifiers is very common. Telephone numbers and e-mail addresses are examples of unique identifiers. Selection of the primary key, or unique identifier, is another key issue of database design.

### Global legal entity identifier (LEI)

There is no global standard for identifying legal entities across markets and jurisdictions. The need for such a standard was amplified by Lehman Brothers collapse in 2008. Lehman had 209 registered subsidiaries, legal entities, in 21 countries, and it was party to more than 900,000 derivatives contracts upon its collapse. Key stakeholders, such as financial regulators and Lehman’s creditors, were unable to assess their exposure. Furthermore, others were unable to assess the possible ripple on them of the effects of the collapse because of the transitive nature of many investments (i.e., A owes B, B owes C, and C owes D).

The adoption of a global legal entity identifier (LEI), should improve financial system regulation and corporate risk management. Regulators will find it easier to monitor and analyze threats to financial stability and risk managers will be more able evaluate their companies’ risks.

Source: <http://www.ny.frb.org/research/epr/2014/1403flem.pdf>

The tables in a relational database are **connected** or **related** by means of the data in the tables. You will learn, in the next chapter, that this connection is through a pair of values—a primary key and a foreign key. Consider a table of airlines serving a city. When examining this table, you may not recognize the code of an airline, so you then go to another table to find the name of the airline. For example, if you inspect the following table, you find that AM is an international airline serving Atlanta.

*International airlines serving Atlanta*

Airline
AM
JL
KX
LM
MA
OS
RG
SN
SR
LH
LY

If you don’t know which airline has the abbreviation AM, then you need to look at the following table of airline codes to discover that AeroMexico, with code AM, serves Atlanta. The two tables are related by airline code. Later, you will discover which is the primary key and which is the foreign key.

*A partial list of airline codes*

	Code	Airline
AA	American Airlines	
AC	Air Canada	
AD	Lone Star Airlines	
AE	Mandarin Airlines	
AF	Air France	
AG	Interprovincial Airlines	
AI	Air India	
AM	AeroMexico	
AQ	Aloha Airlines	

When designing the relational model, Codd provided commands for processing multiple records at a time. His intention was to increase the productivity of programmers by moving beyond the record-at-a-time processing that is found in most programming languages. Consequently, the relational model supports set processing (multiple records-at-a-time), which is most frequently implemented as **Structured Query Language (SQL)**.<sup>7</sup>

The relational model separates the logical design of a database from its physical storage. This notion of **data independence** simplifies data modeling and database programming. In this section, we focus on logical database design, and now that you have had a brief introduction to the relational model, you are ready to learn data modeling.

## Getting started

As with most construction projects, building a relational database must be preceded by a design phase. Data modeling, our design technique, is a method for creating a plan or blueprint of a database. The data model must accurately mirror real-world relationships if it is to support processing business transactions and managerial decision making.

Rather than getting bogged down with a *theory first, application later* approach to database design and use, we will start with application. We will get back to theory when you have some experience in data modeling and database querying. After all, you did not learn to talk by first studying sentence formation; you just started by learning and using simple words. We start with the simplest data model, a single entity, and the simplest database, a single table, as follows.

Firm code	Firm's name	Price	Quantity	Dividend	PE ratio
FC	Freedonia Copper	27.5	10,529	1.84	16
PT	Patagonian Tea	55.25	12,635	2.50	10
AR	Abyssinian Ruby	31.82	22,010	1.32	13
SLG	Sri Lankan Gold	50.37	32,868	2.68	16
ILZ	Indian Lead & Zinc	37.75	6,390	3.00	12
BE	Burmese Elephant	0.07	154,713	0.01	3
BS	Bolivian Sheep	12.75	231,678	1.78	11
NG	Nigerian Geese	35.00	12,323	1.68	10
CS	Canadian Sugar	52.78	4,716	2.50	15
ROF	Royal Ostrich Farms	33.75	1,234,923	3.00	6

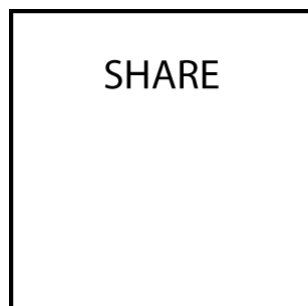
<sup>7</sup>Officially pronounced as “S-Q-L,” but often also pronounced as “sequel”.



## Modeling a single-entity database

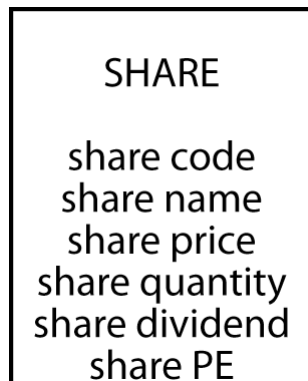
The simplest database contains information about one entity, which is some real-world thing. Some entities are physical—CUSTOMER, ORDER, and STUDENT; others are conceptual—WORK ASSIGNMENT and AUTHORSHIP. We represent an entity by a rectangle: the following figure shows a representation of the entity SHARE. The name of the entity is shown in singular form in uppercase in the top part of the rectangle.

*The entity SHARE*



An entity has characteristics or attributes. An **attribute** is a discrete element of data; it is not usually broken down into smaller components. Attributes identify data we want to store. Some attributes of the entity SHARE are *identification code*, *firm name*, *price*, *quantity owned*, *dividend*, and *price-to-earnings ratio*.<sup>8</sup> Attributes are shown below the entity's name. Notice that we refer to *share price*, rather than *price*, to avoid confusion if there should be another entity with an attribute called *price*. Attribute names must be carefully selected so that they are self-explanatory and unique. For example, *share dividend* is easily recognized as belonging to the entity SHARE.

*The entity SHARE and its attributes*



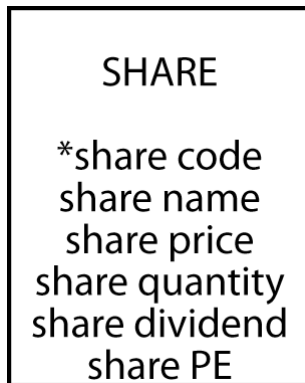
An **instance** is a particular occurrence of an entity (e.g., facts about Freedonia Copper). To avoid confusion, each instance of an entity needs to be uniquely identified. Consider the case of customer billing. In most cases, a request to bill Smith \$100 cannot be accurately processed because a firm might have more than one Smith in its customer file. If a firm has carefully controlled procedures for ensuring that each customer has a unique means of identification, then a request to bill customer number 1789 \$100 can be accurately processed. An attribute or collection of attributes that uniquely identifies an instance of an entity is called an **identifier**. The identifier for the entity SHARE is *share code*, a unique identifier assigned by the stock exchange.

There may be several attributes, or combinations of attributes, that are feasible identifiers for an instance of an entity. Attributes that are identifiers are prefixed by an asterisk. The following figure shows an example of a representation of an entity, its attributes, and identifier.

*The entity SHARE, its attributes, and identifier*

---

<sup>8</sup>Attributes are shown in italics.



Briefly, entities are things in the environment about which we wish to store information. Attributes describe an entity. An entity must have a unique identifier.

The modeling language used in this text is designed to record the essential details of a data model. The number of modeling symbols to learn is small, and they preserve all the fundamental concepts of data modeling. Since data modeling often occurs in a variety of settings, the symbols used have been selected so that they can be quickly drawn using pencil-and-paper, whiteboard, or a general-purpose drawing program. This also means that models can be quickly revised as parts can be readily erased and redrawn.

The symbols are distinct and visual clutter is minimized because only the absolutely essential information is recorded. This also makes the language easy for clients to learn so they can read and amend models.

Models can be rapidly translated to a set of tables for a relational database. More importantly, since this text implements the fundamental notions of all data modeling languages, you can quickly convert to another data modeling dialect. Data modeling is a high-level skill, and the emphasis needs to be on learning to think like a data modeler rather than on learning a modeling language. This text's goal is to get you off to a fast start.

*Skill builder* A ship has a name, registration code, gross tonnage, and a year of construction. Ships are classified as cargo or passenger. Draw a data model for a ship.

## Creating a single-table database

The next stage is to translate the data model into a relational database. The translation rules are very direct:

- Each entity becomes a table.
- The entity name becomes the table name.
- Each attribute becomes a column.
- The identifier becomes the primary key.

The American National Standards Institute's (ANSI) recommended language for relational database definition and manipulation is SQL, which is both a data definition language (DDL) (to define a database), a data manipulation language (DML) (to query and maintain a database), and a data control language (DCL) (to control access). SQL is a common standard for describing and querying databases and is available with many commercial relational database products, including DB2, Oracle, and Microsoft SQL Server, and open source products such as MySQL and PostgreSQL.

In this book, MySQL is the relational database for teaching SQL. Because SQL is a standard, it does not matter which implementation of the relational model you use as the SQL language is common across both the

proprietary and open variants.<sup>9</sup> SQL uses the CREATE<sup>10</sup> statement to define a table. It is not a particularly friendly command, and most products have friendlier interfaces for defining tables. However, it is important to learn the standard, because this is the command that SQL understands. Also, a table definition interface will generate a CREATE statement for execution by SQL. Your interface interactions ultimately translate into a standard SQL command.

It is common practice to abbreviate attribute names, as is done in the following example.

**Defining a table** The CREATE command to establish a table called **share** is as follows:

```
CREATE TABLE share (  
  shrcode CHAR(3),  
  shrfirm VARCHAR(20) NOT NULL,  
  shrprice DECIMAL(6,2),  
  shrqty DECIMAL(8),  
  shrdiv DECIMAL(5,2),  
  shrpe DECIMAL(2),  
  PRIMARY KEY (shrcode));
```

The first line of the command names the table; subsequent lines describe each of its columns. The first component is the name of the column (e.g., **shrcode**). The second component is the data type (e.g., CHAR), and its length is shown in parentheses. **shrfirm** is a variable-length character field of length 20, which means it can store up to 20 characters, including spaces. The column **shrdiv** stores a decimal number that can be as large as 999.99 because its total length is 5 digits and there are 2 digits to the right of the decimal point. Some examples of allowable data types are shown in the following table. The third component (e.g., NOT NULL), which is optional, indicates any instance that cannot have null values. A column might have a null value when it is either unknown or not applicable. In the case of the share table, we specify that **shrfirm** must be defined for each instance in the database.

The final line of the CREATE statement defines **shrcode** as the primary key, the unique identifier for SHARE. When a primary key is defined, the relational database management system (RDBMS) will enforce the requirement that the primary key is unique and not null. Before any row is added to the table SHARE, the RDBMS will check that the value of **shrcode** is not null and that there does not already exist a duplicate value for **shrcode** in an existing row of **share**. If either of these constraints is violated, the RDBMS will not permit the new row to be inserted. This constraint, the **entity integrity rule**, ensures that every row has a unique, non-null primary key. Allowing the primary key to take a null value would mean there could be a row of **share** that is not uniquely identified. Note that an SQL statement is terminated by a semicolon, though this is not always enforced.

SQL statements can be written in any mix of valid upper and lowercase characters. To make it easier for you to learn the syntax, this book adopts the following conventions:

- SQL keywords are in uppercase.
- Table and column names are in lowercase.

There are more elaborate layout styles, but we will bypass those because it is more important at this stage to learn SQL. You should lay out your SQL statements so that they are easily read by you and others.

The following table shows some of the data types supported by most relational databases. Other implementations of the relational model may support some of these data types and additional ones. It is a good idea to review the available data types in your RDBMS before defining your first table.

---

<sup>9</sup>Now would be a good time to install the MySQL Community server <http://www.mysql.com/downloads/mysql/> on your computer, unless your instructor has set up a class server.

<sup>10</sup>SQL keywords are shown in uppercase.

### *Some allowable data types*

Category	Data type	Description
Numeric	SMALLINT	A 15-bit signed binary value
	INTEGER	A 31-bit signed binary value
	FLOAT(p)	A scientific format number of p binary digits precision
	DECIMAL(p,q)	A packed decimal number of p digits total length; q decimal spaces to the right of the decimal point may be specified
String	CHAR(n)	A fixed-length string of n characters
	VARCHAR(n)	A variable-length string of up to n characters
	TEXT	A variable-length string of up to 65,535 characters
Date/time	DATE	Date in the form yyyyymmdd
	TIME	Time in the form hhmmss
	DATETIME	A combination of date and time in the form yyyyymmdd hhmmss
	TIMESTAMP	A combination of date and time to the nearest microsecond
	TIMESTAMP WITH TIME	Same as timestamp, with the addition of an offset from universal time coordinated (UTC) of the specified time
	ZONE	
Logical	BOOLEAN	A set of truth values: TRUE, FALSE, or UNKNOWN

The CHAR and VARCHAR data types are similar but differ in the way character strings are stored and retrieved. Both can be up to 255 characters long. The length of a CHAR column is fixed to the declared length. When values are stored, they are right-padded with spaces to the specified length. When CHAR values are retrieved, trailing spaces are removed. VARCHAR columns store variable-length strings and use only as many characters as are needed to store the string. Values are not padded; instead, trailing spaces are removed. In this book, we use VARCHAR to define most character strings, unless they are short (less than five characters is a good rule-of-thumb).

### **Data modeling with MySQL Workbench**

MySQL Workbench is a professional quality, open source, cross-platform tool for data modeling and SQL querying.<sup>11</sup> In this text, you will also learn some of the features of Workbench that support data modeling and using SQL. You will find it helpful to complete the MySQL Workbench tutorial on creating a data model<sup>12</sup> prior to further reading, as we will assume you have such proficiency.

#### *The entity share created with MySQL Workbench*

---

<sup>11</sup><http://wb.mysql.com/>

<sup>12</sup><http://dev.mysql.com/doc/workbench/en/wb-getting-started-tutorial-creating-a-model.html>



You will notice some differences from the data model we have created previously. Workbench automatically generates the SQL code to create the table, so when modeling you establish the names you want for tables and columns. A gold key symbol is used to indicate the identifier, which becomes the primary key. An open diamond indicates that a column can be null, whereas a closed blue diamond indicates the column must have a value, as with **shrfirm** in this case.

When specifying columns in Workbench you must also indicate the datatype. We opt to turn off the display of a column's datatype<sup>13</sup> in a model to maintain focus on the entity.

A major advantage of using a tool such as Workbench is that it will automatically generate the CREATE statement code (Database > Forward Engineer ...) and execute the code to create the database. The Workbench tutorial will have shown you how to do this, and you should try this out for yourself by creating a database with the single **share** table.

**Inserting rows into a table** The rows of a table store instances of an entity. A particular shareholding (e.g., Freedonia Copper) is an example of an instance of the entity **share**. The SQL statement INSERT is used to add rows to a table. Although most implementations of the relational model use a spreadsheet for row insertion and you can also usually import a structured text file, the INSERT command is defined for completeness.

The following command adds one row to the table **share**:

```
INSERT INTO share
(shrcode,shrfirm,shrprice,shrqty,shrdiv,shrpe)
VALUES ('FC','Freedonia Copper',27.5,10529,1.84,16);
```

There is a one-to-one correspondence between a column name in the first set of parentheses and a value in the second set of parentheses. That is, **shrcode** has the value "FC," **shrfirm** the value "Freedonia Copper," and so on. Notice that the value of a column that stores a character string (e.g., **shrfirm**) is contained within straight quotes.

The list of field names can be omitted when values are inserted in all columns of the table in the same order as that specified in the CREATE statement, so the preceding expression could be written

```
INSERT INTO share
VALUES ('FC','Freedonia Copper',27.5,10529,1.84,16);
```

<sup>13</sup>Preferences > Diagram > Show Column Types

The data for the **share** table will be used in subsequent examples. If you have ready access to a relational database, it is a good idea to now create a table and enter the data. Then you will be able to use these data to practice querying the table.

*Data for share*

<u>Firm code</u>	Firm's name	Price	Quantity	Dividend	PE ratio
FC	Freedonia Copper	27.5	10,529	1.84	16
PT	Patagonian Tea	55.25	12,635	2.50	10
AR	Abyssinian Ruby	31.82	22,010	1.32	13
SLG	Sri Lankan Gold	50.37	32,868	2.68	16
ILZ	Indian Lead & Zinc	37.75	6,390	3.00	12
BE	Burmese Elephant	0.07	154,713	0.01	3
BS	Bolivian Sheep	12.75	231,678	1.78	11
NG	Nigerian Geese	35.00	12,323	1.68	10
CS	Canadian Sugar	52.78	4,716	2.50	15
ROF	Royal Ostrich Farms	33.75	1,234,923	3.00	6

Notice that **shrcode**, the primary key, is underlined in the preceding table. This is a convention we will use for denoting the primary key. In the relational model, an identifier becomes a primary key, a column that guarantees that each row of the table can be uniquely addressed.

MySQL Workbench offers a spreadsheet interface for entering data, as explained in the tutorial on adding data to a database.<sup>14</sup>

*Inserting rows with MySQL Workbench*

shrcode	shrfirm	shrprice	shrqty	shrdiv	shrpe
FC	Freedonia Copper	27.50	10529	1.84	16
NULL	NULL	NULL	NULL	NULL	NULL

*Skill builder* Create a relational database for the ship entity you modeled previously. Insert some rows.

## Querying a database

The objective of developing a database is to make it easier to use the stored data to solve problems. Typically, a manager raises a question (e.g., How many shares have a PE ratio greater than 12?). A question or request for information, usually called a query, is then translated into a specific data manipulation or query language. The most widely used query language for relational databases is SQL. After the query has been executed, the resulting data are displayed. In the case of a relational database, the answer to a query is always a table.

There is also a query language called relational algebra, which describes a set of operations on tables. Sometimes it is useful to think of queries in terms of these operations. Where appropriate, we will introduce the corresponding relational algebra operation.

Generally we use a four-phase format for describing queries:

1. A brief explanation of the query's purpose

<sup>14</sup><http://dev.mysql.com/doc/workbench/en/wb-getting-started-tutorial-adding-data.html>

Table 19: Displaying records 1 - 10

shrcode	shrfirm	shrprice	shrqty	shrdiv	shrpe
AR	Abyssinian Ruby	31.82	22010	1.32	13
BE	Burmese Elephant	0.07	154713	0.01	3
BS	Bolivian Sheep	12.75	231678	1.78	11
CS	Canadian Sugar	52.78	4716	2.50	15
FC	Freedonia Copper	27.50	10529	1.84	16
ILZ	Indian Lead & Zinc	37.75	6390	3.00	12
NG	Nigerian Geese	35.00	12323	1.68	10
PT	Patagonian Tea	55.25	12635	2.50	10
ROF	Royal Ostrich Farms	33.75	1234923	3.00	6
SLG	Sri Lankan Gold	50.37	32868	2.68	16

Table 20: Displaying records 1 - 10

shrfirm	shrpe
Abyssinian Ruby	13
Burmese Elephant	3
Bolivian Sheep	11
Canadian Sugar	15
Freedonia Copper	16
Indian Lead & Zinc	12
Nigerian Geese	10
Patagonian Tea	10
Royal Ostrich Farms	6
Sri Lankan Gold	16

2. The query italics, prefixed by •, and some phrasing you might expect from a manager
3. The SQL version of the query
4. The results of the query.

**Displaying an entire table** All the data in a table can be displayed using the SELECT statement. In SQL, the all part is indicated by an asterisk (\*).

List all data in the share table.

```
SELECT * FROM share;
```

**Project—choosing columns** The relational algebra operation **project** creates a new table from the columns of an existing table. Project takes a vertical slice through a table by selecting all the values in specified columns. The projection of **share** on columns **shrfirm** and **shrpe** produces a new table with 10 rows and 2 columns. The SQL syntax for the project operation simply lists the columns to be displayed.

*Report a firm's name and price-earnings ratio.*

```
SELECT shrfirm, shrpe FROM share;
```

Table 21: 5 records

shrcode	shrfirm	shrprice	shrqty	shrdiv	shrpe
BE	Burmese Elephant	0.07	154713	0.01	3
BS	Bolivian Sheep	12.75	231678	1.78	11
NG	Nigerian Geese	35.00	12323	1.68	10
PT	Patagonian Tea	55.25	12635	2.50	10
ROF	Royal Ostrich Farms	33.75	1234923	3.00	6

Table 23: 3 records

shrfirm	shrprice	shrqty	shrdiv
Burmese Elephant	0.07	154713	0.01
Bolivian Sheep	12.75	231678	1.78
Royal Ostrich Farms	33.75	1234923	3.00

**Restrict—choosing rows** The relational algebra operation **restrict** creates a new table from the rows of an existing table. The operation restricts the new table to those rows that satisfy a specified condition. Restrict takes all columns of an existing table but only those rows that meet the specified condition. The restriction of **share** to those rows where the PE ratio is less than 12 will give a new table with five rows and six columns. These rows are shaded.

*Restriction of share*

Restrict is implemented in SQL using the WHERE clause to specify the condition on which rows are restricted.

*Get all firms with a price-earnings ratio less than 12.*

```
SELECT * FROM share WHERE shrpe < 12;
```

In this example, we have a less than condition for the WHERE clause. All permissible comparison operators are listed below.

	Comparison operator	Meaning
=		Equal to
<		Less than
<=		Less than or equal to
>		Greater than
>=		Greater than or equal to
<>		Not equal to

In addition to the comparison operators, the BETWEEN construct is available.

a BETWEEN x AND y is equivalent to a >= x AND a <= y

**Combining project and restrict—choosing rows and columns** SQL permits project and restrict to be combined. A single SQL SELECT statement can specify which columns to project and which rows to restrict.

*List the name, price, quantity, and dividend of each firm where the share holding is at least 100,000.*

```
SELECT shrfirm, shrprice, shrqty, shrdiv FROM share
WHERE shrqty >= 100000;
```

**More about WHERE** The WHERE clause can contain several conditions linked by AND or OR. A clause containing AND means all specified conditions must be true for a row to be selected. In the case of OR, at least one of the conditions must be true for a row to be selected.

*Find all firms where the PE is 12 or higher and the share holding is less than 10,000.*



Table 24: 2 records

shrcode	shrfirm	shrprice	shrqty	shrdiv	shrpe
CS	Canadian Sugar	52.78	4716	2.5	15
ILZ	Indian Lead & Zinc	37.75	6390	3.0	12

Table 25: 1 records

shrcode	shrfirm	shrprice	shrqty	shrdiv	shrpe
AR	Abyssinian Ruby	31.82	22010	1.32	13

```
SELECT * FROM share
WHERE shrpe >= 12 AND shrqty < 10000;
```

**The power of the primary key** The purpose the primary key is to guarantee that any row in a table can be uniquely addressed. In this example, we use `shrcode` to return a single row because `shrcode` is unique for each instance of `share`. The sought code (AR) must be specified in quotes because `shrcode` was defined as a character string when the table was created.

*Report firms whose code is AR.*

```
SELECT * FROM share WHERE shrcode = 'AR';
```

A query based on a non-primary-key column cannot guarantee that a single row is accessed, as the following illustrates.

*Report firms with a dividend of 2.50.*

```
SELECT * FROM share WHERE shrdiv = 2.5;
```

**The IN crowd** The keyword IN is used with a list to specify a set of values. IN is always paired with a column name. All rows for which a value in the specified column has a match in the list are selected. It is a simpler way of writing a series of OR statements.

*Report data on firms with codes of FC, AR, or SLG.*

```
SELECT * FROM share WHERE shrcode IN ('FC', 'AR', 'SLG');
```

The foregoing query could have also been written as

```
SELECT * FROM share
WHERE shrcode = 'FC' or shrcode = 'AR' or shrcode = 'SLG';
```

Table 26: 2 records

shrcode	shrfirm	shrprice	shrqty	shrdiv	shrpe
CS	Canadian Sugar	52.78	4716	2.5	15
PT	Patagonian Tea	55.25	12635	2.5	10

Table 27: 3 records

shrcode	shrfirm	shrprice	shrqty	shrdiv	shrpe
AR	Abyssinian Ruby	31.82	22010	1.32	13
FC	Freedonia Copper	27.50	10529	1.84	16
SLG	Sri Lankan Gold	50.37	32868	2.68	16

Table 28: 8 records

shrcode	shrfirm	shrprice	shrqty	shrdiv	shrpe
AR	Abyssinian Ruby	31.82	22010	1.32	13
BE	Burmese Elephant	0.07	154713	0.01	3
BS	Bolivian Sheep	12.75	231678	1.78	11
FC	Freedonia Copper	27.50	10529	1.84	16
ILZ	Indian Lead & Zinc	37.75	6390	3.00	12
NG	Nigerian Geese	35.00	12323	1.68	10
ROF	Royal Ostrich Farms	33.75	1234923	3.00	6
SLG	Sri Lankan Gold	50.37	32868	2.68	16

**The NOT IN crowd** A NOT IN list is used to report instances that do not match any of the values.  
*Report all firms other than those with the code CS or PT.*

```
SELECT * FROM share WHERE shrcode NOT IN ('CS','PT');
```

is equivalent to

```
SELECT * FROM share WHERE shrcode <> 'CS' AND shrcode <> 'PT';
```

### *Skill Builder*

List those shares where the value of the holding exceeds one million.

**Ordering columns** The order of reporting columns is identical to their order in the SQL command. For instance, compare the output of the following queries.

```
SELECT shrcode, shrfirm FROM share WHERE shrpe = 10;
```

```
SELECT shrfirm, shrcode FROM share WHERE shrpe = 10;
```

Table 29: 2 records

shrcode	shrfirm
NG	Nigerian Geese
PT	Patagonian Tea

Table 30: 2 records

shrfirm	shrcode
Nigerian Geese	NG
Patagonian Tea	PT

Table 31: 8 records

shrcode	shrfirm	shrprice	shrqty	shrdiv	shrpe
FC	Freedonia Copper	27.50	10529	1.84	16
SLG	Sri Lankan Gold	50.37	32868	2.68	16
CS	Canadian Sugar	52.78	4716	2.50	15
AR	Abyssinian Ruby	31.82	22010	1.32	13
ILZ	Indian Lead & Zinc	37.75	6390	3.00	12
BS	Bolivian Sheep	12.75	231678	1.78	11
NG	Nigerian Geese	35.00	12323	1.68	10
PT	Patagonian Tea	55.25	12635	2.50	10

**Ordering rows** People can generally process an ordered (e.g., sorted alphabetically) report faster than an unordered one. In SQL, the ORDER BY clause specifies the row order in a report. The default ordering sequence is ascending (A before B, 1 before 2). Descending is specified by adding DESC after the column name.

*List all firms where PE is at least 10, and order the report in descending PE. Where PE ratios are identical, list firms in alphabetical order.*

```
SELECT * FROM share WHERE shrpe >= 10
ORDER BY shrpe DESC, shrfirm;
```

**Numeric versus character sorting** Numeric data in character fields (e.g., a product code) do not always sort the way you initially expect. The difference arises from the way data are stored:

- Numeric fields are right justified and have leading zeros.
- Character fields are left justified and have trailing spaces.

For example, the value 1066 stored as CHAR(4) would be stored as '1066' and the value 45 would be stored as '45'. If the column containing these data is sorted in ascending order, then '1066' precedes '45' because the leftmost character '1' is less than '4'. You can avoid this problem by always storing numeric values as numeric data types (e.g., integer or decimal) or preceding numeric values with zeros when they are stored as character data. Alternatively, start numbering at 1,000 so that all values are four digits, though the best solution is to store numeric data as numbers rather than characters.

**Derived data** One of the important principles of database design is to avoid redundancy. One form of redundancy is including a column in a table when these data can be derived from other columns. For example, we do not need a column for yield because it can be calculated by dividing dividend by price and multiplying by 100 to obtain the yield as a percentage. This means that the query language does the calculation when the value is required.

*Get firm name, price, quantity, and firm yield.*

Table 32: Displaying records 1 - 10

shrfirm	shrprice	shrqty	yield
Abyssinian Ruby	31.82	22010	4.148334
Burmese Elephant	0.07	154713	14.285714
Bolivian Sheep	12.75	231678	13.960784
Canadian Sugar	52.78	4716	4.736643
Freedonia Copper	27.50	10529	6.690909
Indian Lead & Zinc	37.75	6390	7.947020
Nigerian Geese	35.00	12323	4.800000
Patagonian Tea	55.25	12635	4.524887
Royal Ostrich Farms	33.75	1234923	8.888889
Sri Lankan Gold	50.37	32868	5.320627

Table 33: 1 records

investments
10

```
SELECT shrfirm, shrprice, shrqty, shrdiv/shrprice*100 AS yield FROM share;
```

You can give the results of the calculation a column name. In this case, a good choice is `yield`. Note the use of `AS` to indicate the name of the column in which the results of the calculation are displayed.

In the preceding query, the keyword `AS` is introduced to specify an alias, or temporary name. The statement specifies that the result of the calculation is to be reported under the column heading `yield`. You can rename any column or specify a name for the results of an expression using an alias.

**Aggregate functions** SQL has built-in functions to enhance its retrieval power and handle many common aggregation queries, such as computing the total value of a column. Four of these functions (`AVG`, `SUM`, `MIN`, and `MAX`) work very similarly. `COUNT` is a little different.

**COUNT** `COUNT` computes the number of rows in a table. Use `COUNT(*)` to count all rows irrespective of their content (i.e., null or not null), and use `COUNT(columnname)` to count rows without a null value for `columnname`. Count can be used with a `WHERE` clause to specify a condition.

*How many firms are there in the portfolio?*

```
SELECT COUNT(shrcode) AS investments FROM share;
```

*How many firms have a holding greater than 50,000?*

```
SELECT COUNT(shrfirm) AS bigholdings FROM share WHERE shrqty > 50000;
```

Table 34: 1 records

bigholdings
3

Table 35: 1 records

avgdiv
2.031

Table 36: 1 records

avgyield
7.530381

**AVG—averaging** AVG computes the average of the values in a column of numeric data. Null values in the column are not included in the calculation.

*Find the average dividend.*

```
SELECT AVG(shrdiv) AS avgdiv FROM share;
```

*What is the average yield for the portfolio?*

```
SELECT AVG(shrdiv/shrprice*100) AS avgyield FROM share;
```

**SUM, MIN, and MAX** SUM, MIN, and MAX differ in the statistic they calculate but are used similarly to AVG. As with AVG, null values in a column are not included in the calculation. SUM computes the sum of a column of values. MIN finds the smallest value in a column; MAX finds the largest.

**Subqueries** Sometimes we need the answer to another query before we can write the query of ultimate interest. For example, to list all shares with a PE ratio greater than the portfolio average, you first must find the average PE ratio for the portfolio. You could do the query in two stages:

```
SELECT AVG(shrpe) FROM share;
```

and

```
SELECT shrfirm, shrpe FROM share WHERE shrpe > x;
```

where x is the value returned from the first query.

Unfortunately, the two-stage method introduces the possibility of errors. You might forget the value returned by the first query or enter it incorrectly. It also takes longer to get the results of the query. We can solve these problems by using parentheses to indicate the first query is nested within the second one. As a result, the value returned by the inner or nested subquery, the one in parentheses, is used in the outer query. In the following example, the nested query returns 11.20, which is then automatically substituted in the outer query.

*Report all firms with a PE ratio greater than the average for the portfolio.*

```
SELECT shrfirm, shrpe FROM share
WHERE shrpe > (SELECT AVG(shrpe) FROM share);
```

Table 37: 5 records

shrfirm	shrpe
Abyssinian Ruby	13
Canadian Sugar	15
Freedonia Copper	16
Indian Lead & Zinc	12
Sri Lankan Gold	16

Table 38: 1 records

shrfirm
Abyssinian Ruby

The preceding query is often mistakenly written as  
 SELECT shrfirm, shrpe from share WHERE shrpe > avg(shrpe);

#### *Skill builder*

Find the name of the firm for which the value of the holding is greatest.

**Regular expression—pattern matching** Regular expression is a concise and powerful method for searching for a specified pattern in a nominated column. Regular expression processing is supported by languages such as Java, R, and PHP. In this chapter, we introduce a few typical regular expressions and will continue developing your knowledge of this feature in the next chapter.

**Search for a string** *List all firms containing ‘Ruby’ in their name.*

```
SELECT shrfirm FROM share WHERE shrfirm REGEXP 'Ruby';
```

**Search for alternative strings** In some situations you want to find columns that contain more than one string. In this case, we use the alternation symbol ‘|’ to indicate the alternatives being sought. For example, a|b finds ‘a’ or ‘b’.

*List the firms containing gold or zinc in their name, irrespective of case.*

```
SELECT shrfirm FROM share WHERE shrfirm REGEXP 'gold|zinc|Gold|Zinc';
```

**Search for a beginning string** If you are interested in finding a value in a column that starts with a particular character string, then use ^ to indicate this option.

*List the firms whose name begins with Sri.*

Table 39: 2 records

shrfirm
Indian Lead & Zinc
Sri Lankan Gold

Table 40: 1 records

shrfirm
Sri Lankan Gold

Table 41: 1 records

shrfirm
Nigerian Geese

```
SELECT shrfirm FROM share WHERE shrfirm REGEXP '^Sri';
```

**Search for an ending string** If you are interested in finding if a value in a column ends with a particular character string, then use \$ to indicate this option.

*List the firms whose name ends with Geese.*

```
SELECT shrfirm FROM share WHERE shrfirm REGEXP 'Geese$';
```

#### *Skill builder*

List the firms containing “ian” in their name.

**DISTINCT—eliminating duplicate rows** The DISTINCT clause is used to eliminate duplicate rows. It can be used with column functions or before a column name. When used with a column function, it ignores duplicate values.

*Report the different values of the PE ratio.*

```
SELECT DISTINCT shrpe FROM share;
```

*Find the number of different PE ratios.*

```
SELECT COUNT(DISTINCT shrpe) as 'Different PEs' FROM share;
```

When used before a column name, DISTINCT prevents the selection of duplicate rows. Notice a slightly different use of the keyword AS. In this case, because the alias includes a space, the entire alias is enclosed in straight quotes.

Table 42: 8 records

shrpe
13
3
11
15
16
12
10
6

Table 43: 1 records

Different PEs
8

Table 45: 1 records

person first
Sheila

**DELETE** Rows in a table can be deleted using the DELETE clause in an SQL statement. DELETE is typically used with a WHERE clause to specify the rows to be deleted. If there is no WHERE clause, all rows are deleted.

*Erase the data for Burmese Elephant. All the shares have been sold.*

```
DELETE FROM share WHERE shrfirm = 'Burmese Elephant';
```

In the preceding statement, shrfirm is used to indicate the row to be deleted.

**UPDATE** Rows can be modified using SQL's UPDATE clause, which is used with a WHERE clause to specify the rows to be updated.

*Change the share price of FC to 31.50.*

```
UPDATE share SET shrprice = 31.50 WHERE shrcode = 'FC';
```

*Increase the total number of shares for Nigerian Geese by 10% because of the recent bonus issue.*

```
UPDATE share SET shrqty = shrqty*1.1 WHERE shrfirm = 'Nigerian Geese';
```

**Quotes** There are three types of quotes that you can typically use with SQL. Double and single quotes are equivalent and can be used interchangeably. Note that single and double quotes must be straight rather than curly, and the back quote is to the left of the 1 key.

Type of quote	Representation
Single	'
Double	"
Back	`

The following SQL illustrates the use of three types of quotes to find a person with a last name of O'Hara and where the column names are person first and person last.

```
SELECT `person first` FROM person WHERE `person last` = "O'Hara";
```

**Debriefing** Now that you have learned how to model a single entity, create a table, and specify queries, you are on the way to mastering the fundamental skills of database design, implementation, and use. Remember, planning occurs before action. A data model is a plan for a database. The action side of a database is inserting rows and running queries.



## Summary

The relational database model is an effective means of representing real-world relationships. Data modeling is used to determine what data must be stored and how data are related. An entity is something in the environment. An entity has attributes, which describe it, and an identifier, which uniquely distinguishes an instance of an entity. Every entity must have a unique identifier. A relational database consists of tables with rows and columns. A data model is readily translated into a relational database. The SQL statement CREATE is used to define a table. Rows are added to a table using INSERT. In SQL, queries are written using the SELECT statement. Project (choosing columns) and restrict (choosing rows) are common table operations. The WHERE clause is used to specify row selection criteria. WHERE can be combined with IN and NOT IN, which specify values for a single column. The rows of a report are sorted using the ORDER BY clause. Arithmetic expressions can appear in SQL statements, and SQL has built-in functions for common arithmetic operations. A subquery is a query within a query. Regular expressions are used to find string patterns within character strings. Duplicate rows are eliminated with the DISTINCT clause. Rows can be erased using DELETE or modified with UPDATE.

## Key terms and concepts

Alias	Instance
AS	MAX
Attribute	MIN
AVG	NOT IN
Column	ORDER BY
COUNT	Primary key
CREATE	Project
Data modeling	Relational database
Data type	Restrict
Database	Row
DELETE	SELECT
DISTINCT	SQL
Entity	Subquery
Entity integrity rule	SUM
Identifier	Table
IN	UPDATE
INSERT	WHERE

## Exercises

1. Draw data models for the following entities. In each case, make certain that you show the attributes and identifiers. Also, create a relational database and insert some rows for each of the models.
  - a. Aircraft: An aircraft has a manufacturer, model number, call sign (e.g., N123D), payload, and a year of construction. Aircraft are classified as civilian or military.
  - b. Car: A car has a manufacturer, range name, and style code (e.g., a Honda Accord DX, where Honda is the manufacturer, Accord is the range, and DX is the style). A car also has a vehicle identification code, registration code, and color.
  - c. Restaurant: A restaurant has an address, seating capacity, phone number, and style of food (e.g., French, Russian, Chinese).

- d. Cow: A dairy cow has a name, date of birth, breed (e.g., Holstein), and a numbered plastic ear tag.

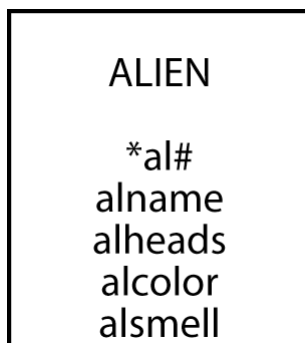
2. Do the following queries using SQL:

- a. List a share's name and its code.
- b. List full details for all shares with a price less than \$1.
- c. List the names and prices of all shares with a price of at least \$10.
- d. Create a report showing firm name, share price, share holding, and total value of shares held. (Value of shares held is price times quantity.)
- e. List the names of all shares with a yield exceeding 5 percent.
- f. Report the total dividend payment of Patagonian Tea. (The total dividend payment is dividend times quantity.)
- g. Find all shares where the price is less than 20 times the dividend.
- h. Find the share(s) with the minimum yield.
- i. Find the total value of all shares with a PE ratio  $> 10$ .
- j. Find the share(s) with the maximum total dividend payment.
- k. Find the value of the holdings in Abyssinian Ruby and Sri Lankan Gold.
- l. Find the yield of all firms except Bolivian Sheep and Canadian Sugar.
- m. Find the total value of the portfolio.
- n. List firm name and value in descending order of value.
- o. List shares with a firm name containing "Gold."
- p. Find shares with a code starting with "B."

3. Run the following queries and explain the differences in output. Write each query as a manager might state it.

- a. `SELECT shrfirm FROM share WHERE shrfirm NOT REGEXP 's';`
- b. `SELECT shrfirm FROM share WHERE shrfirm NOT REGEXP 'S';`
- c. `SELECT shrfirm FROM share WHERE shrfirm NOT REGEXP 's|S';`
- d. `SELECT shrfirm FROM share WHERE shrfirm NOT REGEXP '^S';`
- e. `SELECT shrfirm FROM share WHERE shrfirm NOT REGEXP 's$';`

4. A weekly newspaper, sold at supermarket checkouts, frequently reports stories of aliens visiting Earth and taking humans on short trips. Sometimes a captured human sees Elvis commanding the spaceship. To keep track of all these reports, the newspaper has created the following data model.



The paper has also supplied some data for the last few sightings and asked you to create the database, add details of these aliens, and answer the following queries:

- a. What's the average number of heads of an alien?
  - a. Which alien has the most heads?
  - b. Are there any aliens with a double o in their names?
  - c. How many aliens are chartreuse?
  - d. Report details of all aliens sorted by smell and color.
5. Eduardo, a bibliophile, has a collection of several hundred books. Being a little disorganized, he has his books scattered around his den. They are piled on the floor, some are in bookcases, and others sit on top of his desk. Because he has so many books, he finds it difficult to remember what he has, and sometimes he cannot find the book he wants. Eduardo has a simple personal computer file system that is fine for a single entity or file. He has decided that he would like to list each book by author(s)' name and type of book (e.g., literature, travel, reference). Draw a data model for this problem, create a single entity table, and write some SQL queries.
- a. How do you identify each instance of a book? (It might help to look at a few books.)
  - b. How should Eduardo physically organize his books to permit fast retrieval of a particular one?
  - c. Are there any shortcomings with the data model you have created?
6. What is an identifier? Why does a data model have an identifier?
7. What are entities?
8. What is the entity integrity rule?

## 4 The One-to-Many Relationship

*Cow of many—well milked and badly fed.*

Spanish proverb

### Learning Objectives

Students completing this chapter will be able to

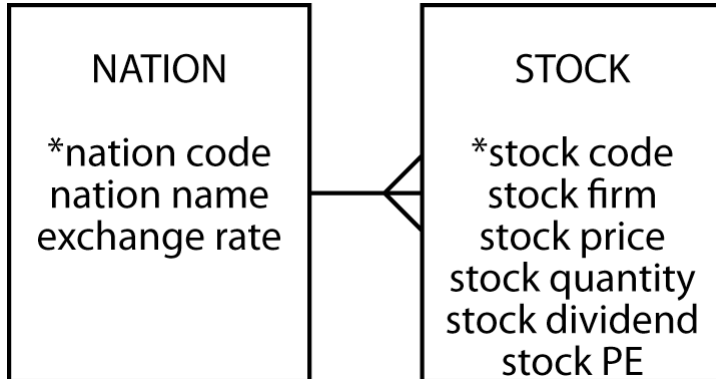
- model a one-to-many relationship between two entities;
- define a database with a one-to-many relationship;
- write queries for a database with a one-to-many relationship.

### Relationships

Entities are not isolated; they are related to other entities. When we move beyond the single entity, we need to identify the relationships between entities to accurately represent the real world. Consider the case where a person's stocks are listed in different countries. We now need to introduce an entity called NATION. We now have two entities, STOCK and NATION. Consider the relationship between them. A NATION can have many listed stocks. A stock, in this case, is listed in only one nation. There is a 1:m (one-to-many) relationship between NATION and STOCK.

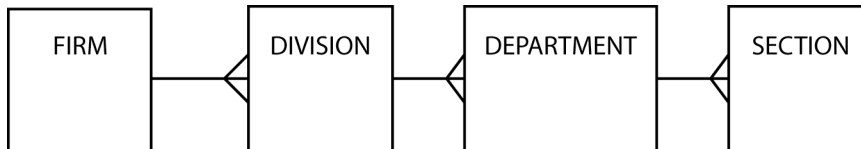
A 1:m relationship between two entities is depicted by a line connecting the two with a crow's foot at the many end of the relationship. The following figure shows the 1:m relationship between NATION and STOCK. This can be read as: “a nation can have many stocks, but a stock belongs to only one nation.” The entity NATION is identified by *nation code* and has attributes *nation name* and *exchange rate*.

*A 1:m relationship between NATION and STOCK*



The 1:m relationship occurs frequently in business situations. Sometimes it occurs in a tree or hierarchical fashion. Consider a very hierarchical firm. It has many divisions, but a division belongs to only one firm. A division has many departments, but a department belongs to only one division. A department has many sections, but a section belongs to only one department.

*A series of 1:m relationships*



### Why did we create an additional entity?

Another approach to adding data about listing nation and exchange rate is to add two attributes to STOCK: *nation name* and *exchange rate*. At first glance, this seems a very workable solution; however, this will introduce considerable redundancy, as the following table illustrates.

*The table stock with additional columns*

	stkcode	stkfirm	stkprice	stkqty	stkdiv	stkpe	natname	exchrte
FC	Freedonia	Copper	27.5	10529	1.84	16	United Kingdom	1
PT	Patagonian	Tea	55.25	12635	2.5	10	United Kingdom	1
AR	Abyssinian	Ruby	31.82	22010	1.32	13	United Kingdom	1
SLG	Sri Lankan	Gold	50.37	32868	2.68	16	United Kingdom	1
ILZ	Indian	Lead & Zinc	37.75	6390	3	12	United Kingdom	1
BE	Burmese	Elephant	0.07	154713	0.01	3	United Kingdom	1
BS	Bolivian	Sheep	12.75	231678	1.78	11	United Kingdom	1
NG	Nigerian	Geese	35	12323	1.68	10	United Kingdom	1
CS	Canadian	Sugar	52.78	4716	2.5	15	United Kingdom	1
ROF	Royal	Ostrich Farms	33.75	1234923	3	6	United Kingdom	1
MG	Minnesota	Gold	53.87	816122	1	25	USA	0.67
GP	Georgia	Peach	2.35	387333	0.2	5	USA	0.67
NE	Naremben	Emu	12.34	45619	1	8	Australia	0.46
QD	Queensland	Diamond	6.73	89251	0.5	7	Australia	0.46

	<u>stkcode</u>	stkfirm	stkprice	stkqty	stkdiv	stkpe	natname	exchrte
IR	Indooroopilly	Ruby	15.92	56147	0.5	20	Australia	0.46
BD	Bombay	Duck	25.55	167382	1	12	India	0.0228

The exact same nation name and exchange rate data pair occurs 10 times for stocks listed in the United Kingdom. Redundancy presents problems when we want to insert, delete, or update data. These problems, generally known as **update anomalies**, occur with these three basic operations.

**Insert anomalies** We cannot insert a fact about a nation's exchange rate unless we first buy a stock that is listed in that nation. Consider the case where we want to keep a record of France's exchange rate and we have no French stocks. We cannot skirt this problem by putting in a null entry for stock details because `stkcode`, the primary key, would be null, and this is not allowed. If we have a separate table for facts about a nation, then we can easily add new nations without having to buy stocks. This is particularly useful when other parts of the organization, say International Trading, also need access to exchange rates for many nations.

**Delete anomalies** If we delete data about a particular stock, we might also lose a fact about exchange rates. For example, if we delete details of Bombay Duck, we also erase the Indian exchange rate.

**Update anomalies** Exchange rates are volatile. Most companies need to update them every day. What happens when the Australian exchange rate changes? Every row in stock with nation = 'Australia' will have to be updated. In a large portfolio, many rows will be changed. There is also the danger of someone forgetting to update all the instances of the nation and exchange rate pair. As a result, there could be two exchange rates for the one nation. If exchange rate is stored in nation, however, only one change is necessary, there is no redundancy, and there is no danger of inconsistent exchange rates.

## Creating a database with a 1:m relationship

As before, each entity becomes a table in a relational database, the entity name becomes the table name, each attribute becomes a column, and each identifier becomes a primary key. The 1:m relationship is mapped by adding a column to the entity at the many end of the relationship. The additional column contains the identifier of the one end of the relationship.

Consider the relationship between the entities STOCK and NATION. The database has two tables: **stock** and **nation**. The table **stock** has an additional column, `natcode`, which contains the primary key of **nation**. If `natcode` is not stored in **stock**, then there is no way of knowing the identity of the nation where the **stock** is listed.

*A relational database with tables nation and stock*

	<u>nation</u>		
<u>natcode</u>	natname	exchrte	
AUS	Australia	0.46	
IND	India	0.0228	
UK	United Kingdom	1	
USA	United States	0.67	→

<u>stkcode</u>	stkfirm	stock		stkdiv	stkpe	<i>natcode</i>	
		stkprice	stkqty				
FC	Freedonia Copper	27.5	10,529	1.84	16	UK	
PT	Patagonian Tea	55.25	12,635	2.5	10	UK	
AR	Abyssinian Ruby	31.82	22,010	1.32	13	UK	
SLG	Sri Lankan Gold	50.37	32,868	2.68	16	UK	
ILZ	Indian Lead & Zinc	37.75	6,390	3	12	UK	
BE	Burmese Elephant	0.07	154,713	0.01	3	UK	
BS	Bolivian Sheep	12.75	231,678	1.78	11	UK	
NG	Nigerian Geese	35	12,323	1.68	10	UK	
CS	Canadian Sugar	52.78	4,716	2.5	15	UK	
ROF	Royal Ostrich Farms	33.75	1,234,923	3	6	UK	
MG	Minnesota Gold	53.87	816,122	1	25	USA	←
GP	Georgia Peach	2.35	387,333	0.2	5	USA	←
NE	Naremben Emu	12.34	45,619	1	8	AUS	
QD	Queensland Diamond	6.73	89,251	0.5	7	AUS	
IR	Indooroopilly Ruby	15.92	56,147	0.5	20	AUS	
BD	Bombay Duck	25.55	167,382	1	12	IND	

Notice that **natcode** appears in both the **stock** and **nation** tables. In **nation**, **natcode** is the primary key; it is unique for each instance of **nation**. In table **stock**, **natcode** is a **foreign key** because it is the primary key of **nation**, the one end of the 1:m relationship. The column **natcode** is a foreign key in **stock** because it is a primary key in **nation**. A matched primary key–foreign key pair is the method for recording the 1:m relationship between the two tables. This method of representing a relationship is illustrated using shading and arrows for the two USA stocks. In the **stock** table, **natcode** is italicized to indicate that it is a foreign key. This method, like underlining a primary key, is a useful reminder.

Although the same name has been used for the primary key and the foreign key in this example, it is not mandatory. The two columns can have different names, and in some cases you are forced to use different names. When possible, we find it convenient to use identical column names to help us remember that the tables are related. To distinguish between columns with identical names, they must be **qualified** by prefixing the table name. In this case, use **nation.natcode** and **stock.natcode**. Thus, **nation.natcode** refers to the **natcode** column in the table **nation**.

Although a nation can have many stocks, it is not mandatory to have any. That is, in data modeling terminology, many can be zero, one, or more, but it is mandatory to have a value for **natcode** in **nation** for every value of **natcode** in **stock**. This requirement, known as the **referential integrity constraint**, maintains the accuracy of a database. Its application means that every foreign key in a table has an identical primary key in that same table or another table. In this example, it means that for every value of **natcode** in **stock**, there is a corresponding entry in **nation**. As a result, a primary key row must be created before its corresponding foreign key row. In other words, details for a **nation** must be added before any data about its listed stocks are entered.

Every foreign key must have a matching primary key (referential integrity rule), and every primary key must be non-null (entity integrity rule). A foreign key cannot be null when a relationship is mandatory, as in the case where a stock must belong to a nation. If a relationship is optional (a person can have a boss), then a foreign key can be null (i.e., a person is the head of the organization, and thus has no boss). The ideas of mandatory and optional will be discussed later in this book.

Why is the foreign key in the table at the “many” end of the relationship? Because each instance of **stock** is associated with exactly one instance of **nation**. The rule is that a **stock** must be listed in one, and only one, **nation**. Thus, the foreign key field is single-valued when it is at the “many” end of a relationship. The foreign key is not at the “one” end of the relationship because each instance of **nation** can be associated with more than one instance of **stock**, and this implies a multivalued foreign key. The relational model does not support multivalued fields.

Using SQL, the two tables are defined in a similar manner to the way we created a single table in Chapter 3. Here are the SQL statements:

```
CREATE TABLE nation (  
    natcode CHAR(3),  
    natname VARCHAR(20),  
    exchrates DECIMAL(9,5),  
    PRIMARY KEY(natcode));
```

```
CREATE TABLE stock (  
    stkcode CHAR(3),  
    stkfirm VARCHAR(20),  
    stkprice DECIMAL(6,2),  
    stkqty DECIMAL(8),  
    stkdiv DECIMAL(5,2),  
    stkpe DECIMAL(5),  
    natcode CHAR(3),  
    PRIMARY KEY(stkcode),  
    CONSTRAINT fk_has_nation FOREIGN KEY(natcode)  
    REFERENCES nation(natcode) ON DELETE RESTRICT);
```

Notice that the definition of `stock` includes an additional phrase to specify the foreign key and the referential integrity constraint. The `CONSTRAINT` clause defines the column or columns in the table being created that constitute the foreign key. A referential integrity constraint can be named, and in this case, the constraint's name is `fk_has_nation`. The foreign key is the column `natcode` in `STOCK`, and it references the primary key of `nation`, which is `natcode`.

The `ON DELETE` clause specifies what processing should occur if an attempt is made to delete a row in `nation` with a primary key that is a foreign key in `stock`. In this case, the `ON DELETE` clause specifies that it is not permissible (the meaning of `RESTRICT`) to delete a primary key row in `nation` while a corresponding foreign key in `stock` exists. In other words, the system will not execute the delete. You must first delete all corresponding rows in `stock` before attempting to delete the row containing the primary key. `ON DELETE` is the default clause for most RDBMSs, so we will dispense with specifying it for future foreign key constraints.

Observe that both the primary and foreign keys are defined as `CHAR(3)`. The relational model requires that a primary key–foreign key pair have the same data type and are the same length.

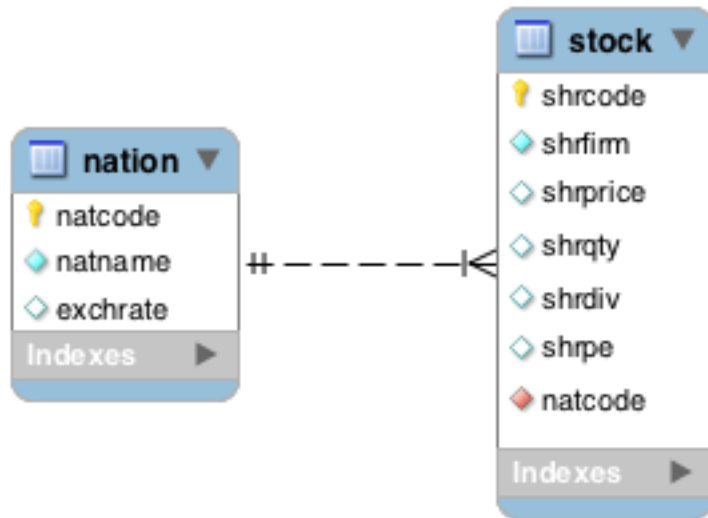
### *Skill builder*

The university architect has asked you to develop a data model to record details of the campus buildings. A building can have many rooms, but a room can be in only one building. Buildings have names, and rooms have a size and purpose (e.g., lecture, laboratory, seminar). Draw a data model for this situation and create the matching relational database.

## MySQL Workbench

In Workbench, a 1:m relationship is represented in a similar manner to the method you have just learned. Also, note that the foreign key is shown in the entity at the many end with a red diamond. We omit the foreign key when data modeling because it can be inferred. You will observe some additional symbols on the line between the two entities, and these will be explained later, but take note of the crow's foot indicating the 1:m relationship between `nation` and `stock`. Because Workbench can generate automatically the SQL to create the tables,<sup>11</sup> we use lowercase table names and abbreviated column names.

*Specifying a 1:m relationship in MySQL Workbench*



## Querying a two-table database

A two-table database offers the opportunity to learn more SQL and another relational algebra operation: join.

### Join

Join creates a new table from two existing tables by matching on a column common to both tables. Usually, the common column is a primary key–foreign key pair: The primary key of one table is matched with the foreign key of another table. Join is frequently used to get the data for a query into a single row. Consider the tables **nation** and **stock**. If we want to calculate the value—in British pounds—of a stock, we multiply stock price by stock quantity and then exchange rate. To find the appropriate exchange rate for a stock, get its **natcode** from **stock** and then find the exchange rate in the matching row in **nation**, the one with the same value for **natcode**. For example, to calculate the value of Georgia Peach, which has **natcode** = 'US', find the row in **nation** that also has **natcode** = 'US'. In this case, the stock's value is  $2.35 \times 387333 \times 0.67 = \text{£}609,855.81$ .

Calculation of stock value is very easy once a join is used to get the three values in one row. The SQL command for joining the two tables is:

```
SELECT * FROM stock JOIN nation
ON stock.natcode = nation.natcode;
```

### *The join of stock and nation*

The columns **stkprice** and **stkdiv** record values in the country's currency. Thus, the price of Bombay Duck is 25.55 Indian rupees. To find the value in U.K. pounds (GPB), multiply the price by 0.0228, because one rupee is worth 0.0228 GPB. The value of one share of Bombay Duck in U.S. dollars (USD) is  $25.55 \times 0.0228 / 0.67$  because one USD is worth 0.67 GBP.

There are several things to notice about the SQL command and the result:

- To avoid confusion because **natcode** is a column name in both **stock** and **nation**, it needs to be qualified. If **natcode** is not qualified, the system will reject the query because it cannot distinguish between the two columns titled **natcode**.



Table 50: Displaying records 1 - 10

stkcode	stkfirm	stkprice	stkqty	stkdiv	stkpe	natcode	natcode	natname	exchrates
IR	Indooroopilly Ruby	15.92	56147	0.50	20	AUS	AUS	Australia	0.4600
NE	Narembeen Emu	12.34	45619	1.00	8	AUS	AUS	Australia	0.4600
QD	Queensland Diamond	6.73	89251	0.50	7	AUS	AUS	Australia	0.4600
BD	Bombay Duck	25.55	167382	1.00	12	IND	IND	India	0.0228
AR	Abyssinian Ruby	31.82	22010	1.32	13	UK	UK	United Kingdom	1.0000
BE	Burmese Elephant	0.07	154713	0.01	3	UK	UK	United Kingdom	1.0000
BS	Bolivian Sheep	12.75	231678	1.78	11	UK	UK	United Kingdom	1.0000
CS	Canadian Sugar	52.78	4716	2.50	15	UK	UK	United Kingdom	1.0000
FC	Freedonia Copper	27.50	10529	1.84	16	UK	UK	United Kingdom	1.0000
ILZ	Indian Lead & Zinc	37.75	6390	3.00	12	UK	UK	United Kingdom	1.0000

Table 51: 1 records

count(*)
3

- The new table has the **natcode** column replicated. Both are called **natcode**. The naming convention for the replicated column varies with the RDBMS. The columns, for example, should be labeled **stock.natcode** and **nation.natcode**.
- The SQL command specifies the names of the tables to be joined, the columns to be used for matching, and the condition for the match (equality in this case).
- The number of columns in the new table is the sum of the columns in the two tables.
- The stock value calculation is now easily specified in an SQL command because all the data are in one row.

Remember that during data modeling we created two entities, STOCK and NATION, and defined the relationship between them. We showed that if the data were stored in one table, there could be updating problems. Now, with a join, we have combined these data. So why separate the data only to put them back together later? There are two reasons. First, we want to avoid update anomalies. Second, as you will discover, we do not join the same tables every time.

Join comes in several flavors. The matching condition can be =, <>, <=, <, >=, and >. This generalized version is called a **theta join**. Generally, when people refer to a join, they mean an equijoin, when the matching condition is equality.

*In an alphabetical list of employees, how many appear before Clare?*

```
SELECT count(*) FROM emp A JOIN emp B
  ON A.empfname > B.empfname
 WHERE A.empfname = "Clare"
```

A join can be combined with other SQL commands.

*Report the value of each stockholding in UK pounds. Sort the report by nation and firm.*

```
SELECT natname, stkfirm, stkprice, stkqty, exchrates,
       stkprice*stkqty*exchrates as stkvalue
  FROM stock JOIN nation
```

Table 52: Displaying records 1 - 10

natname	stkfirm	stkprice	stkqty	exchrte	stkvalue
Australia	Indooroopilly Ruby	15.92	56147	0.4600	411175.71
Australia	Narembeen Emu	12.34	45619	0.4600	258951.69
Australia	Queensland Diamond	6.73	89251	0.4600	276303.25
India	Bombay Duck	25.55	167382	0.0228	97506.71
United Kingdom	Abyssinian Ruby	31.82	22010	1.0000	700358.20
United Kingdom	Bolivian Sheep	12.75	231678	1.0000	2953894.50
United Kingdom	Burmese Elephant	0.07	154713	1.0000	10829.91
United Kingdom	Canadian Sugar	52.78	4716	1.0000	248910.48
United Kingdom	Freedonia Copper	27.50	10529	1.0000	289547.50
United Kingdom	Indian Lead & Zinc	37.75	6390	1.0000	241222.50

```
ON stock.natcode = nation.natcode
ORDER BY natname, stkfirm;
```

### Control break reporting

The purpose of a join is to collect the necessary data for a report. When two tables in a 1:m relationship are joined, the report will contain repetitive data. If you re-examine the report from the previous join, you will see that **nation** and **exchrte** are often repeated because the same values apply to many stocks. A more appropriate format is shown in the following figure, an example of a **control break report**.

Nation	Exchange rate			
Firm		Price	Quantity	Value
<b>Australia</b>	<b>0.4600</b>			
Indooroopilly Ruby		15.92	56,147	411,175.71
Narembeen Emu		12.34	45,619	258,951.69
Queensland Diamond		6.73	89,251	276,303.25
<b>India</b>	<b>0.0228</b>			
Bombay Duck		25.55	167,382	97,506.71
<b>United Kingdom</b>	<b>1.0000</b>			
Abyssinian Ruby		31.82	22,010	700,358.20
Bolivian Sheep		12.75	231,678	2,953,894.50
Burmese Elephant		0.07	154,713	10,829.91
Canadian Sugar		52.78	4,716	248,910.48
Freedonia Copper		27.50	10,529	289,547.50
Indian Lead & Zinc		37.75	6,390	241,222.50
Nigerian Geese		35.00	12,323	431,305.00
Patagonian Tea		55.25	12,635	698,083.75
Royal Ostrich Farms		33.75	1,234,923	41,678,651.25
Sri Lankan Gold		50.37	32,868	1,655,561.16
<b>United States</b>	<b>0.6700</b>			
Georgia Peach		2.35	387,333	609,855.81
Minnesota Gold		53.87	816,122	29,456,209.73

A control break report recognizes that the values in a particular column or columns seldom change. In this case, **natname** and **exchrte** are often the same from one row to the next, so it makes sense to report these data only when they change. The report is also easier to read. The column **natname** is known as a *control field*. Notice that there are four groups of data, because **natname** has four different values.

Table 53: 4 records

natname	stkvalue
Australia	946430.65
India	97506.71
United Kingdom	48908364.25
United States	30066065.54

Table 54: 4 records

natname	COUNT(*)	stkvalue
Australia	3	946430.65
India	1	97506.71
United Kingdom	10	48908364.25
United States	2	30066065.54

Many RDBMS packages have report-writing languages to facilitate creating a control break report. These languages typically support summary reporting for each group of rows having the same value for the control field(s). A table must usually be sorted on the control break field(s) before the report is created.

**GROUP BY—reporting by groups** The GROUP BY clause is an elementary form of control break reporting. It permits grouping of rows that have the same value for a specified column or columns, and it produces one row for each different value of the grouping column(s).

*Report by nation the total value of stockholdings.*

```
SELECT natname, sum(stkprice*stkqty*exchrates) AS stkvalue
FROM stock JOIN nation ON stock.natcode = nation.natcode
GROUP BY natname;
```

SQL's built-in functions (COUNT, SUM, AVERAGE, MIN, and MAX) can be used with the GROUP BY clause. They are applied to a group of rows having the same value for a specified column. You can specify more than one function in a SELECT statement. For example, we can compute total value and number of different stocks and group by nation using:

*Report the number of stocks and their total value by nation.*

```
SELECT natname, COUNT(*), SUM(stkprice*stkqty*exchrates) AS stkvalue
FROM stock JOIN nation ON stock.natcode = nation.natcode
GROUP BY natname;
```

You can group by more than one column name; however, all column names appearing in the SELECT clause must be associated with a built-in function or be in a GROUP BY clause.

*List stocks by nation, and for each nation show the number of stocks for each PE ratio and the total value of those stock holdings in UK pounds.*

```
SELECT natname,stkpe,COUNT(*),
SUM(stkprice*stkqty*exchrates) AS stkvalue
FROM stock JOIN nation ON stock.natcode = nation.natcode
GROUP BY natname, stkpe;
```

Table 55: Displaying records 1 - 10

natname	stkpe	COUNT(*)	stkvalue
Australia	20	1	411175.71
Australia	8	1	258951.69
Australia	7	1	276303.25
India	12	1	97506.71
United Kingdom	13	1	700358.20
United Kingdom	3	1	10829.91
United Kingdom	11	1	2953894.50
United Kingdom	15	1	248910.48
United Kingdom	16	2	1945108.66
United Kingdom	12	1	241222.50

Table 56: 3 records

natname	stkvalue
Australia	946430.6
United Kingdom	48908364.2
United States	30066065.5

In this example, stocks are grouped by both **natname** and **stkpe**. In most cases, there is only one stock for each pair of **natname** and **stkpe**; however, there are two situations (U.K. stocks with PEs of 10 and 16) where details of multiple stocks are grouped into one report line. Examining the values in the COUNT column helps you to identify these stocks.

**HAVING—the WHERE clause of groups** HAVING does in the GROUP BY what the WHERE clause does in a SELECT. It restricts the number of groups reported, whereas WHERE restricts the number of rows reported. Used with built-in functions, HAVING is always preceded by GROUP BY and is always followed by a function (SUM, AVG, MAX, MIN, or COUNT).

*Report the total value of stocks for nations with two or more listed stocks.*

```
SELECT natname, SUM(stkprice*stkqty*exchrates) AS stkvalue
FROM stock JOIN nation ON stock.natcode = nation.natcode
GROUP BY natname
HAVING COUNT(*) >= 2;
```

### *Skill builder*

Report by nation the total value of dividends.

## Regular expression—pattern matching

Regular expression was introduced in the previous chapter, and we will now continue to learn some more of its features.

**Search for a string not containing specified characters** The ^ (carat) is the symbol for NOT. It is used when we want to find a string not containing a character in one or more specified strings. For example,

— *f*

Table 57: 2 records

natname
United Kingdom
United States

Table 58: 5 records

stkfirm
Bolivian Sheep
Freedonia Copper
Narembreen Emu
Nigerian Geese
Queensland Diamond

means any character not in the set containing a, b, c, d, e, or f.

*List the names of nations with non-alphabetic characters in their names*

```
SELECT natname FROM nation WHERE natname REGEXP '[^a-z|A-Z]'
```

Notice that the nations reported have a space in their name, which is a character not in the range a-z and not in A-Z.

**Search for string containing a repeated pattern or repetition** A pair of curly brackets is used to denote the repetition factor for a pattern. For example, {n} means repeat a specified pattern n times.

*List the names of firms with a double ‘e’.*

```
SELECT stkfirm FROM stock WHERE stkfirm REGEXP '[e]{2}';
```

**Search combining alternation and repetition** Regular expressions becomes very powerful when you combine several of the basic capabilities into a single search expression.

*List the names of firms with a double ‘s’ or a double ‘n’.*

```
SELECT stkfirm FROM stock WHERE stkfirm REGEXP '[s]{2}|[n]{2}';
```

**Search for multiple versions of a string** If you are interested in find a string containing several specified string, you can use the square brackets to indicate the sought strings. For example,

*ea*

Table 59: 2 records

stkfirm
Abyssinian Ruby
Minnesota Gold

Table 60: 3 records

stkfirm
Abyssinian Ruby
Freedonia Copper
Patagonian Tea

Table 61: 1 records

stkfirm
Patagonian Tea

means any character from the set containing e and a.

*List the names of firms with names that include ‘inia’ or ‘onia’.*

```
SELECT stkfirm FROM stock WHERE stkfirm REGEXP '[io]nia';
```

**Search for a string in a particular position** Sometimes you might be interested in identifying a string with a character in a particular position.

*Find firms with ‘t’ as the third letter of their name.*

```
SELECT stkfirm FROM stock WHERE stkfirm REGEXP '^(.){2}t';
```

The regular expression has three elements:

- ^ indicates start searching at the beginning of the string;
- (.){2} specifies that anything is acceptable for the next two characters;
- t indicates what the next character, the third, must be.

You have seen a few of the features of a very powerful tool. To learn more about regular expressions, see [regexlib.com](http://regexlib.com),<sup>12</sup> which contains a library of regular expressions and a feature for finding expressions to solve specific problems. Check out the regular expression for checking whether a character string is a valid email address.

## Subqueries

A subquery, or nested SELECT, is a SELECT nested within another SELECT. A subquery can be used to return a list of values subsequently searched with an IN clause.

*Report the names of all Australian stocks.*

```
SELECT stkfirm FROM stock
  WHERE natcode IN
    (SELECT natcode FROM nation
     WHERE natname = 'Australia');
```

Conceptually, the subquery is evaluated first. It returns a list of values for natcode (‘AUS’) so that the query then is the same as:

Table 62: 3 records

stkfirm
Indooroopilly Ruby
Narembeen Emu
Queensland Diamond

Table 64: 4 records

natname	stkfirm	stkqty
Australia	Queensland Diamond	89251
United Kingdom	Bolivian Sheep	231678
United Kingdom	Royal Ostrich Farms	1234923
United States	Minnesota Gold	816122

```
SELECT stkfirm FROM stock
WHERE natcode IN ('AUS');
```

When discussing subqueries, sometimes a subquery is also called an **inner query**. The term **outer query** is applied to the SQL preceding the inner query. In this case, the outer and inner queries are:

Outer query	SELECT stkfirm FROM stock
	WHERE natcode IN
Inner query	(SELECT natcode FROM nation
	WHERE natname = 'Australia');

Note that in this case we do not have to qualify **natcode**. There is no identity crisis, because **natcode** in the inner query is implicitly qualified as **nation.natcode** and **natcode** in the outer query is understood to be **stock.natcode**.

This query also can be run as a join by writing:

```
SELECT stkfirm FROM stock JOIN nation
ON stock.natcode = nation.natcode
AND natname = 'Australia';
```

**Correlated subquery** In a correlated subquery, the subquery cannot be evaluated independently of the outer query. It depends on the outer query for the values it needs to resolve the inner query. The subquery is evaluated for each value passed to it by the outer query. An example illustrates when you might use a correlated subquery and how it operates.

*Find those stocks where the quantity is greater than the average for that country.*

An approach to this query is to examine the rows of stock one at a time, and each time compare the quantity of stock to the average for that country. This means that for each row, the subquery must receive the outer query's country code so it can compute the average for that country.

```
SELECT natname, stkfirm, stkqty FROM stock JOIN nation
ON stock.natcode = nation.natcode
WHERE stkqty >
(SELECT avg(stkqty) FROM stock
WHERE stock.natcode = nation.natcode);
```

Conceptually, think of this query as stepping through the join of **stock** and **nation** one row at a time and executing the subquery each time. The first row has **natcode** = 'AUS' so the subquery becomes

```
SELECT AVG(stkqty) FROM stock
WHERE stock.natcode = 'AUS';
```

Since the average stock quantity for Australian stocks is 63,672.33, the first row in the join, Narembeen Emu, is not reported. Neither is the second row reported, but the third is.

The term **correlated subquery** is used because the inner query's execution depends on receiving a value for a variable (`nation.natcode` in this instance) from the outer query. Thus, the inner query of the correlated subquery cannot be evaluated once and for all. It must be evaluated repeatedly—once for each value of the variable received from the outer query. In this respect, a correlated subquery is different from a subquery, where the inner query needs to be evaluated only once. The requirement to compare each row of a table against a function (e.g., average or count) for some rows of a column is usually a clue that you need to write a correlated subquery.

#### *Skill builder*

Why are no Indian stocks reported in the correlated subquery example? How would you change the query to report an Indian stock?

Report only the three stocks with the largest quantities (i.e., do the query without using ORDER BY).

## Views—virtual tables

You might have noticed that in these examples we repeated the join and stock value calculation for each query. Ideally, we should do this once, store the result, and be able to use it with other queries. We can do so if we create a **view**, a virtual table. A view does not physically exist as stored data; it is an imaginary table constructed from existing tables as required. You can treat a view as if it were a table and write SQL to query it.

A view contains selected columns from one or more tables. The selected columns can be renamed and rearranged. New columns based on arithmetic expressions can be created. GROUP BY can also be used when creating a view. Remember, a view contains no actual data. It is a virtual table.

This SQL command does the join, calculates stock value, and saves the result as a view:

```
CREATE VIEW stkvalue
(nation, firm, price, qty, exchrates, value)
AS SELECT natname, stkfirm, stkprice, stkqty, exchrates,
       stkprice*stkqty*exchrates
FROM stock JOIN nation
ON stock.natcode = nation.natcode;
```

There are several things to notice about creating a view:

- The six names enclosed by parentheses are the column names for the view.
- There is a one-to-one correspondence between the names in parentheses and the names or expressions in the SELECT clause. Thus the view column named `value` contains the result of the arithmetic expression `stkprice*stkqty*exchrates`.

A view can be used in a query, such as:

*Find stocks with a value greater than £100,000.*

```
SELECT nation, firm, value FROM stkvalue WHERE value > 100000;
```

There are two main reasons for creating a view. *First*, as we have seen, query writing can be simplified. If you find that you are frequently writing the same section of code for a variety of queries, then isolate the



Table 65: Displaying records 1 - 10

nation	firm	value
Australia	Indooroopilly Ruby	411175.7
Australia	Narembeen Emu	258951.7
Australia	Queensland Diamond	276303.2
United Kingdom	Abyssinian Ruby	700358.2
United Kingdom	Bolivian Sheep	2953894.5
United Kingdom	Canadian Sugar	248910.5
United Kingdom	Freedonia Copper	289547.5
United Kingdom	Indian Lead & Zinc	241222.5
United Kingdom	Nigerian Geese	431305.0
United Kingdom	Patagonian Tea	698083.8

common section and put it in a view. This means that you will usually create a view when a fact, such as stock value, is derived from other facts in the table.

The *second* reason is to restrict access to certain columns or rows. For example, the person who updates **stock** could be given a view that excludes **stkqty**. In this case, changes in stock prices could be updated without revealing confidential information, such as the value of the stock portfolio.

*Skill builder*

How could you use a view to solve the following query that was used when discussing the correlated subquery? | *Find those stocks where the quantity is greater than the average for that country.*

### Summary

Entities are related to other entities by relationships. The 1:m (one-to-many) relationship occurs frequently in data models. An additional entity is required to represent a 1:m relationship to avoid update anomalies. In a relational database, a 1:m relationship is represented by an additional column, the foreign key, in the table at the many end of the relationship. The referential integrity constraint insists that a foreign key must always exist as a primary key in a table. A foreign key constraint is specified in a CREATE statement.

Join creates a new table from two existing tables by matching on a column common to both tables. Often the common column is a primary key–foreign key combination. A theta-join can have matching conditions of =, <>, <=, <, >=, and >. An equijoin describes the situation where the matching condition is equality. The GROUP BY clause is used to create an elementary control break report. The HAVING clause of GROUP BY is like the WHERE clause of SELECT. A subquery, which has a SELECT statement within another SELECT statement, causes two SELECT statements to be executed—one for the inner query and one for the outer query. A correlated subquery is executed as many times as there are rows selected by the outer query. A view is a virtual table that is created when required. Views can simplify report writing and restrict access to specified columns or rows.

### Key terms and concepts

	—
	—
Constraint	JOIN
Control break reporting	One-to-many (1:m) relationship
Correlated subquery	Referential integrity
Delete anomalies	Relationship

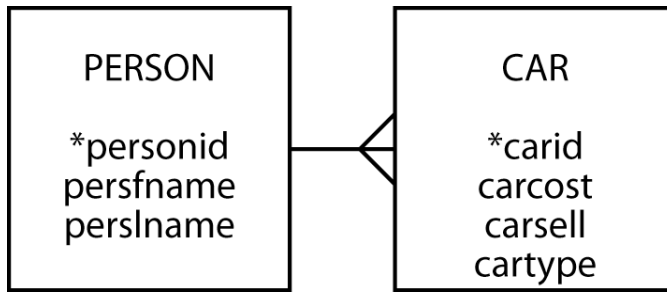
---

Equijoin	Theta-join
Foreign key	Update anomalies
GROUP BY	Views
HAVING	Virtual table
Insert anomalies	

---

## Exercises

- Draw data models for the following situations. In each case, make certain that you show the attributes and feasible identifiers:
  - A farmer can have many cows, but a cow belongs to only one farmer.
  - A university has many students, and a student can attend at most one university.
  - An aircraft can have many passengers, but a passenger can be on only one flight at a time.
  - A nation can have many states and a state many cities.
  - An art researcher has asked you to design a database to record details of artists and the museums in which their paintings are displayed. For each painting, the researcher wants to know the size of the canvas, year painted, title, and style. The nationality, date of birth, and death of each artist must be recorded. For each museum, record details of its location and specialty, if it has one.
- Report all values in British pounds:
  - Report the value of stocks listed in Australia.
  - Report the dividend payment of all stocks.
  - Report the total dividend payment by nation.
  - Create a view containing nation, firm, price, quantity, exchange rate, value, and yield.
  - Report the average yield by nation.
  - Report the minimum and maximum yield for each nation.
  - Report the nations where the average yield of stocks exceeds the average yield of all stocks.
- How would you change the queries in exercise 4-2 if you were required to report the values in American dollars, Australian dollars, or Indian rupees?
- What is a foreign key and what role does it serve?
- What is the referential integrity constraint? Why should it be enforced?
- Kisha, against the advice of her friends, is simultaneously studying data management and Shakespearean drama. She thought the two subjects would be an interesting contrast. However, the classes are very demanding and often enter her midsummer dreams. Last night, she dreamed that William Shakespeare wanted her to draw a data model. He explained, before she woke up in a cold sweat, that a play had many characters but the same character never appeared in more than one play. “Methinks,” he said, “the same name may have appeareth more than the once, but ’twas always a person of a different ilk.” He then, she hazily recollects, went on to spout about the quality of data dropping like the gentle rain. Draw a data model to keep old Bill quiet and help Kisha get some sleep.
- An orchestra has four broad classes of instruments (strings, woodwinds, brass, and percussion). Each class contains musicians who play different instruments. For example, the strings section of a full symphony orchestra contains 2 harps, 16 to 18 first violins, 14 to 16 second violins, 12 violas, 10 cellos, and 8 double basses. A city has asked you to develop a database to store details of the musicians in its three orchestras. All the musicians are specialists and play only one instrument for one orchestra.



8. Answer the following queries based on the following database for a car dealer:

- What is the personid of Sheila O'Hara?
- List sales personnel sorted by last name and within last name, first name.
- List details of the sales made by Bruce Bush.
- List details of all sales showing the gross profit (selling price minus cost price).
- Report the number of cars sold of each type.
- What is the average selling price of cars sold by Sue Lim?
- Report details of all sales where the gross profit is less than the average.
- What was the maximum selling price of any car?
- What is the total gross profit?
- Report the gross profit made by each salesperson who sold at least three cars.
- Create a view containing all the details in the car table and the gross profit

9. Find stocks where the third or fourth letter in their name is an 'm'.

10. An electricity supply company needs a database to record details of solar panels installed on its customers' homes so it can estimate how much solar energy will be generated based on the forecast level of solar radiation for each house's location. A solar panel has an area, measured in square meters, and an efficiency expressed as a percentage (e.g., 22% efficiency means that 22% of the incident solar energy is converted into electrical energy). Create a data model. How will you identify each customer and each panel?

## 5 The Many-to-Many Relationship

*Fearful concatenation of circumstances.*

Daniel Webster

### Learning objectives

Students completing this chapter will be able to

- model a many-to-many relationship between two entities;
- define a database with a many-to-many relationship;
- write queries for a database with a many-to-many relationship.

## The many-to-many relationship

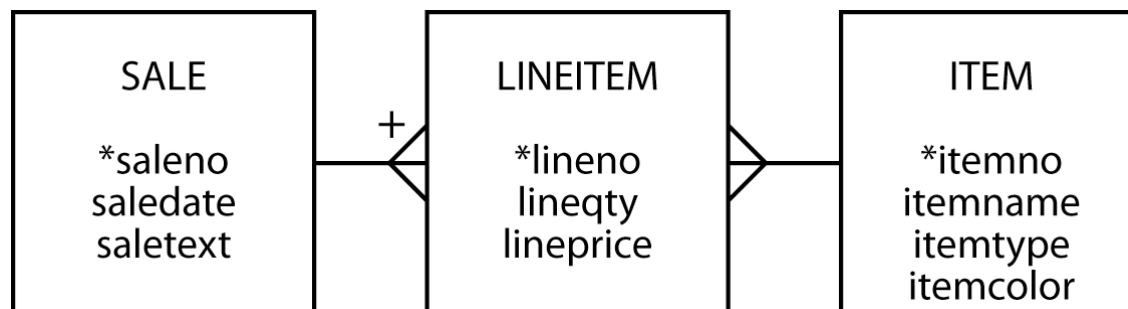
Consider the case when items are sold. We can immediately identify two entities: SALE and ITEM. A sale can contain many items, and an item can appear in many sales. We are not saying the same item can be sold many times, but the particular type of item (e.g., a compass) can be sold many times; thus we have a many-to-many (m:m) relationship between SALE and ITEM. When we have an m:m relationship, we create a third entity to link the entities through two 1:m relationships. Usually, it is fairly easy to name this third entity. In this case, this third entity, typically known as an **associative entity**, is called LINE ITEM. A typical old style sales form lists the items purchased by a customer. Each of the lines appearing on the order form is generally known in retailing as a line item, which links an item and a sale.

<i>The Expeditioner</i> Sale of Goods					
Sale#			Date:		
Item#	Description		Quantity	Unit price	Total
1					
2					
3					
4					
5					
6					
Grand total					

*A sales form*

The representation of this m:m relationship is shown. We say many-to-many because there are two relationships—an ITEM is related to many SALES, and a SALE is related to many ITEMS. This data model can also be read as: “a sale has many line items, but a line item refers to only one sale. Similarly, an item can appear as many line items, but a line item references only one item.”

*An m:m relationship between SALE and ITEM*



The entity SALE is identified by *saleno* and has the attributes *saledate* and *saletext* (a brief comment on the customer—soft information). LINEITEM is partially identified by *lineno* and has attributes *lineqty* (the number of units sold) and *lineprice* (the unit selling price for this sale). ITEM is identified by *itemno* and has attributes *itemname*, *itemtype* (e.g., clothing, equipment, navigation aids, furniture, and so on), and *itemcolor*.

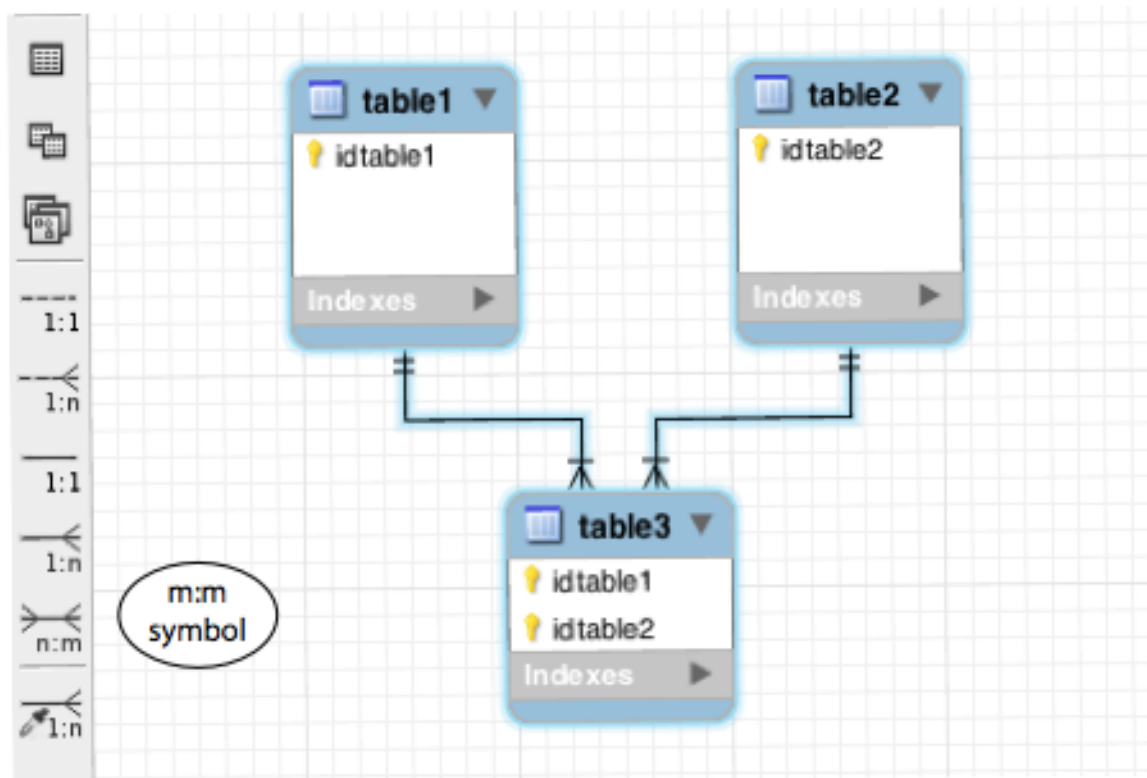
If you look carefully at the m:m relationship figure, you will notice that there is a plus sign (+) above the crow's foot at the “many” end of the 1:m relationship between SALE and LINEITEM. This plus sign provides information about the identifier of LINEITEM. As you know, every entity must have a unique identifier. A sales order is a series of rows or lines, and *lineno* is unique only within a particular order. If we just use *lineno* as the identifier, we cannot guarantee that every instance of LINEITEM is unique. If we use *saleno* and *lineno* together, however, we have a unique identifier for every instance of LINEITEM. Identifier *saleno* is unique for every sale, and *lineno* is unique within any sale. The plus indicates that LINEITEM's unique identifier is the concatenation of *saleno* and *lineno*. The order of concatenation does not matter.

LINEITEM is termed a **weak entity** because it relies on another entity for its existence and identification.

## MySQL Workbench

Workbench automatically creates an associative entity for an m:m relationship and populates it with a composite primary key based on concatenating the primary keys of the two entities forming the m:m relationship. *First*, draw the two tables and enter their respective primary keys and columns. *Second*, select the m:m symbol and connect the two tables through clicking on one and dragging to the second and releasing. You can then modify the associative entity as required, such as changing its primary key. The capability to automatically create an associative entity for an m:m relationship is a very useful Workbench feature.

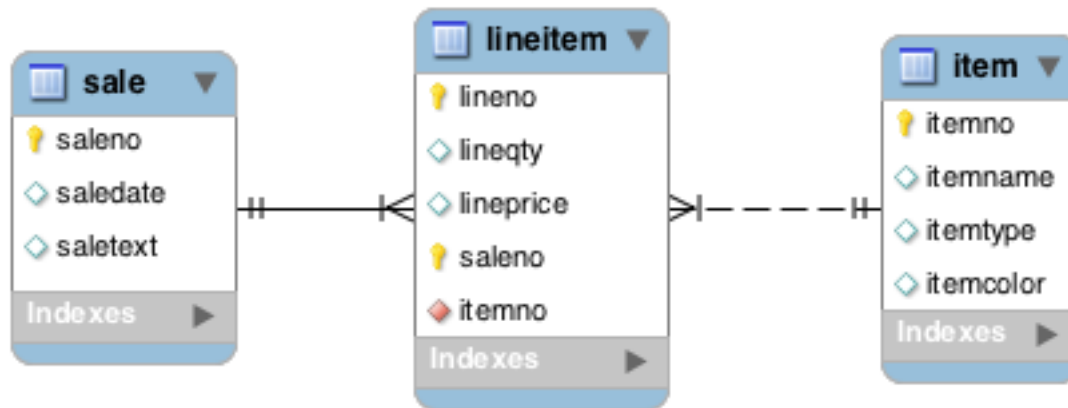
*An m:m relationship with Workbench*



Workbench distinguishes between two types of relationships. An **identifying relationship**, shown by a solid line, is used when the entity at the many end of the relationship is a weak entity and needs the identifier of the one end of the relationship to uniquely identify an instance of the relationship, as in LINEITEM. An identifying relationship corresponds to the + sign associated with a crow's foot. The other type of relationship, shown by a dashed line, is known as a *non-identifying* relationship. The mapping between the type of relationship and the representation (i.e., dashed or solid line) is arbitrary and thus not always easily recalled. We think that using a + on the crow's foot is a better way of denoting weak entities.

When the relationship between SALE and ITEM is drawn in Workbench, as shown in the following figure, there are two things to notice. *First*, the table, lineitem, maps the associative entity generated for the m:m relationship. *Second*, lineitem has an identifying relationship with sale and a non-identifying relationship with item.

*An m:m relationship between SALE and ITEM in MySQL Workbench*



### Why did we create a third entity?

When we have an m:m relationship, we create an associative entity to store data about the relationship. In this case, we have to store data about the items sold. We cannot store the data with SALE because a sale can have many items, and an instance of an entity stores only single-value facts. Similarly, we cannot store data with ITEM because an item can appear in many sales. Since we cannot store data in SALE or ITEM, we must create another entity to store data about the m:m relationship.

You might find it useful to think of the m:m relationship as two 1:m relationships. An item can appear on many line item listings, and a line item entry refers to only one item. A sale has many line items, and each line item entry refers to only one sale.

### *Social Security number is notunique!*

Two girls named Sarah Lee Ferguson were born on May 3, 1959. The U.S. government considered them one and the same and issued both the same Social Security number (SSN), a nine-digit identifier of U.S. residents. Now Sarah Lee Ferguson Boles and Sarah Lee Ferguson Johnson share the same SSN.<sup>15</sup>

Mrs. Boles became aware of her SSN twin in 1987 when the Internal Revenue Service claimed there was a discrepancy in her reported income. Because SSN is widely used as an attribute or identifier in many computer systems, Mrs. Boles encountered other incidents of mistaken identity. Some of Mrs. Johnson's purchases appeared on Mrs. Boles' credit reports.

In late 1989, the Social Security Administration notified Mrs. Boles that her original number was given to her in error and she had to provide evidence of her age, identity, and citizenship to get a new number. When Mrs. Boles got her new SSN, it is likely she had to also get a new driver's license and establish a new credit history.

<sup>15</sup>"Two women share a name, birthday, and S.S. number!" Athens Daily News, January 29 1990, 7A. Also, see > <https://www.computerworld.com/article/3004659/a-tale-of-two-women-same-birthday-same-social-security-number-same-big-data-mess.html>

## Creating a relational database with an m:m relationship

As before, each entity becomes a table in a relational database, the entity name becomes the table name, each attribute becomes a column, and each identifier becomes a primary key. Remember, a 1:m relationship is mapped by adding a column to the entity of the many end of the relationship. The new column contains the identifier of the one end of the relationship.

Conversion of the foregoing data model results in the three tables following. Note the one-to-one correspondence between attributes and columns for sale and item. Observe the lineitem has two additional columns, saleno and itemno. Both of these columns are foreign keys in lineitem (remember the use of italics to signify foreign keys). Two foreign keys are required to record the two 1:m relationships. Notice in lineitem that saleno is both part of the primary key and a foreign key.

*Tables sale, lineitem, and item*

		<u>saleno</u>	saledate	saletext
1	2020-01-15			Scruffy Australian—called himself Bruce.
2	2020-01-15			Man. Rather fond of hats.
3	2020-01-15			Woman. Planning to row Atlantic—lengthwise!
4	2020-01-15			Man. Trip to New York—thinks NY is a jungle!
5	2020-01-16			Expedition leader for African safari.

<u>lineno</u>	lineqty	lineprice	<i>saleno</i>	<i>itemno</i>
1	1	4.50	1	2
1	1	25.00	2	6
2	1	20.00	2	16
3	1	25.00	2	19
4	1	2.25	2	2
1	1	500.00	3	4
2	1	2.25	3	2
1	1	500.00	4	4
2	1	65.00	4	9
3	1	60.00	4	13
4	1	75.00	4	14
5	1	10.00	4	3
6	1	2.25	4	2
1	50	36.00	5	10
2	50	40.50	5	11
3	8	153.00	5	12
4	1	60.00	5	13
5	1	0.00	5	2

	<u>itemno</u>	itemname	itemtype	itemcolor
1		Pocket knife—Nile	E	Brown
2		Pocket knife—Avon	E	Brown
3		Compass	N	—
4		Geopositioning system	N	—
5		Map measure	N	—
6		Hat—Polar Explorer	C	Red
7		Hat—Polar Explorer	C	White

	<u>itemno</u>	<u>itemname</u>	<u>itemtype</u>	<u>itemcolor</u>
8		Boots—snake proof	C	Green
9		Boots—snake proof	C	Black
10		Safari chair	F	Khaki
11		Hammock	F	Khaki
12		Tent—8 person	F	Khaki
13		Tent—2 person	F	Khaki
14		Safari cooking kit	E	—
15		Pith helmet	C	Khaki
16		Pith helmet	C	White
17		Map case	N	Brown
18		Sextant	N	—
19		Stetson	C	Black
20		Stetson	C	Brown

The SQL commands to create the three tables are as follows:

```
CREATE TABLE sale (
    saleno      INTEGER,
    saledate    DATE NOT NULL,
    saletext    VARCHAR(50),
    PRIMARY KEY(saleno));
```

```
CREATE TABLE item (
    itemno      INTEGER,
    itemname    VARCHAR(30) NOT NULL,
    itemtype    CHAR(1) NOT NULL,
    itemcolor   VARCHAR(10),
    PRIMARY KEY(itemno));
```

```
CREATE TABLE lineitem (
    lineno      INTEGER,
    lineqty     INTEGER NOT NULL,
    lineprice   DECIMAL(7,2) NOT NULL,
    saleno      INTEGER,
    itemno      INTEGER,
    PRIMARY KEY(lineno,saleno),
    CONSTRAINT fk_has_sale FOREIGN KEY(saleno)
        REFERENCES sale(saleno),
    CONSTRAINT fk_has_item FOREIGN KEY(itemno)
        REFERENCES item(itemno));
```

Although the `sale` and `item` tables are created in a similar fashion to previous examples, there are two things to note about the definition of `lineitem`. *First*, the primary key is a composite of `lineno` and `saleno`, because together they uniquely identify an instance of `lineitem`. *Second*, there are two foreign keys, because `lineno` is at the “many” end of two 1: m relationships.

### *Skill builder*

A hamburger shop makes several types of hamburgers, and the same type of ingredient can be used with several types of hamburgers. This does not literally mean the same piece of lettuce is used many times, but lettuce is used with several types of hamburgers. Draw the data model for this situation. What is a good name for the associative entity?



Table 70: 5 records

itemname	lineqty	lineprice	total
Safari chair	50	36.0	1800
Hammock	50	40.5	2025
Tent - 8 person	8	153.0	1224
Tent - 2 person	1	60.0	60
Pocket knife - Avon	1	0.0	0

## Querying an m:m relationship

### A three-table join

The join operation can be easily extended from two tables to three or more merely by specifying the tables to be joined and the matching conditions. For example:

```
SELECT * FROM sale JOIN lineitem
  ON sale.salenno = lineitem.salenno
  JOIN item
  ON item.itemno = lineitem.itemno;
```

There are two matching conditions: one for `sale` and `lineitem` (`sale.salenno = lineitem.salenno`) and one for the `item` and `lineitem` tables (`item.itemno = lineitem.itemno`). The table `lineitem` is the link between `sale` and `item` and must be referenced in both matching conditions.

You can tailor the join to be more precise and report some columns rather than all.

*List the name, quantity, price, and value of items sold on January 16, 2011.*

```
SELECT itemname, lineqty, lineprice, lineqty*lineprice AS total
  FROM sale, lineitem, item
  WHERE lineitem.salenno = sale.salenno
  AND item.itemno = lineitem.itemno
  AND saledate = '2011-01-16';
```

### EXISTS—does a value exist

EXISTS is used in a WHERE clause to test whether a table contains at least one row satisfying a specified condition. It returns the value **true** if and only if some row satisfies the condition; otherwise it returns **false**. EXISTS represents the **existential quantifier** of formal logic. The best way to get a feel for EXISTS is to examine a query.

*Report all clothing items (type “C”) for which a sale is recorded.*

```
SELECT itemname, itemcolor FROM item
  WHERE itemtype = 'C'
  AND EXISTS (SELECT * FROM lineitem
  WHERE lineitem.itemno = item.itemno);
```

Conceptually, we can think of this query as evaluating the subquery for each row of `item`. The first item with `itemtype = 'C'`, Hat—Polar Explorer (red), in `item` has `itemno = 6`. Thus, the query becomes

Table 71: 4 records

itemname	itemcolor
Hat - Polar explorer	Red
Boots - snake proof	Black
Pith helmet	White
Stetson	Black

Table 72: 4 records

itemname	itemcolor
Hat - Polar explorer	White
Boots - snake proof	Green
Pith helmet	Khaki
Stetson	Brown

```
SELECT itemname, itemcolor FROM item
  WHERE itemtype = 'C'
 AND EXISTS (SELECT * FROM lineitem
            WHERE lineitem.itemno = 6);
```

Because there is at least one row in `lineitem` with `itemno = 6`, the subquery returns *true*. The item has been sold and should be reported. The second clothing item, Hat—Polar Explorer (white), in `item` has `itemno = 7`. There are no rows in `lineitem` with `itemno = 7`, so the subquery returns *false*. That item has not been sold and should not be reported.

You can also think of the query as, “Select clothing items for which a sale exists.” Remember, for EXISTS to return *true*, there needs to be only one row for which the condition is *true*.

### NOT EXISTS—select a value if it does not exist

NOT EXISTS is the negative of EXISTS. It is used in a WHERE clause to test whether all rows in a table fail to satisfy a specified condition. It returns the value *true* if there are no rows satisfying the condition; otherwise it returns *false*.

*Report all clothing items that have not been sold.*

```
SELECT itemname, itemcolor FROM item
  WHERE itemtype = 'C'
 AND NOT EXISTS
    (SELECT * FROM lineitem
     WHERE item.itemno = lineitem.itemno);
```

You can also think of the query as, “Select clothing items for which no sales exist.” Also remember, for NOT EXISTS to return *true*, no rows should satisfy the condition.

### Skill builder

Report all red items that have not been sold. Write the query twice, once using EXISTS and once without EXISTS.

Table 73: 1 records

itemno	itemname
2	Pocket knife - Avon

### Divide (and be conquered)

In addition to the existential quantifier that you have already encountered, formal logic has a **universal quantifier** known as **forall** that is necessary for queries such as

Find the items that have appeared in all sales.

If a universal quantifier were supported by SQL, this query could be phrased as, “Select item names where *forall* sales there *exists* a *lineitem* row recording that this item was sold.” A quick inspection of the first set of tables shows that one item satisfies this condition (*itemno* = 2).

While SQL does not directly support the universal quantifier, formal logic shows that *forall* can be expressed using EXISTS. The query becomes, “Find items such that there does not exist a sale in which this item does not appear.” The equivalent SQL expression is

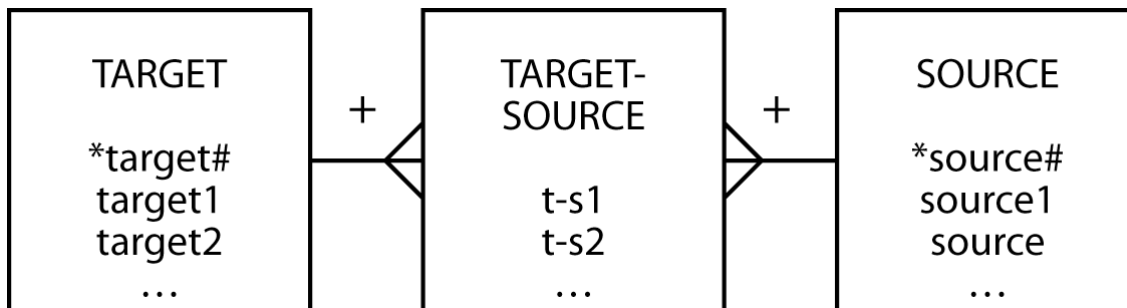
```
SELECT itemno, itemname FROM item
  WHERE NOT EXISTS
    (SELECT * FROM sale
      WHERE NOT EXISTS
        (SELECT * FROM lineitem
          WHERE lineitem.itemno = item.itemno
            AND lineitem.salenno = sale.salenno));
```

If you are interested in learning the inner workings of the preceding SQL for divide, see the additional material for Chapter 5 on the book’s Web site.

Relational algebra (Chapter 9) has the divide operation, which makes divide queries easy to write. Be careful: Not all queries containing the word *all* are divides. With experience, you will learn to recognize and conquer divide.

To save the tedium of formulating this query from scratch, we have developed a template for dealing with these sorts of queries. Divide queries typically occur with m:m relationships.

*A template for divide*



An appropriate generic query and template SQL command are

*Find the target1 that have appeared in all sources.*

```
SELECT target1 FROM target
  WHERE NOT EXISTS
```

```
(SELECT * FROM source
 WHERE NOT EXISTS
  (SELECT * FROM target-source
   WHERE target-source.target# = target.target#
   AND target-source.source# = source.source#));
```

### *Skill builder*

Find the brown items that have appeared in all sales.

## Beyond the great divide

Divide proves troublesome to most people because of the double negative—we just don’t think that way. If divide sends your neurons into knots, then try the following approach.

The query “Find the items that have appeared in all sales” can be rephrased as “Find the items for which the number of sales that include this item is equal to the total number of sales.” This is an easier query to write than “Find items such that there does not exist a sale in which this item does not appear.” The rephrased query has two parts. *First*, determine the total number of sales. Here we mean distinct sales (i.e., the number of rows with a distinct value for saleno). The SQL is

```
SELECT COUNT (DISTINCT saleno) FROM sale;
```

*Second*, group the items sold by itemno and itemname and use a HAVING clause with COUNT to calculate the number of sales in which the item has occurred. Forcing the count in the HAVING clause to equal the result of the first query, which becomes an inner query, results in a list of items appearing in all sales.

```
SELECT item.itemno, item.itemname
 FROM item JOIN lineitem
   ON item.itemno = lineitem.itemno
  GROUP BY item.itemno, item.itemname
   HAVING COUNT(DISTINCT saleno)
        = (SELECT COUNT(DISTINCT saleno) FROM sale);
```

## Set operations

Set operators are useful for combining the values derived from two or more SQL queries. The UNION operation is equivalent to *or*, and INTERSECT is equivalent to *and*.

*List items that were sold on January 16, 2011, or are brown.*

Resolution of this query requires two tables: one to report items sold on January 16, 2011, and one to report the brown items. UNION (i.e., or) then combines the results of the tables, including *any* rows in both tables and excluding duplicate rows.

```
SELECT itemname FROM item JOIN lineitem
   ON item.itemno = lineitem.itemno
 JOIN sale
   ON lineitem.salenno = sale.salenno
 WHERE saledate = '2011-01-16'
UNION
 SELECT itemname FROM item WHERE itemcolor = 'Brown';
```

Table 74: 8 records

itemname
Safari chair
Hammock
Tent - 8 person
Tent - 2 person
Pocket knife - Avon
Pocket knife - Nile
Map case
Stetson

List items that were sold on January 16, 2011, and are brown.

This query uses the same two tables as the previous query. In this case, INTERSECT (i.e., and) then combines the results of the tables including *only* rows in both tables and excluding duplicates.<sup>16</sup>

```
SELECT itemname FROM item JOIN lineitem
  ON item.itemno = lineitem.itemno
  JOIN sale
  ON lineitem.salenno = sale.salenno
 WHERE saledate = '2011-01-16'
INTERSECT
SELECT itemname FROM item WHERE itemcolor = 'Brown';
```

#### Skill builder

List the items that contain the words “Hat”, “Helmet”, or “Stetson” in their names

## Summary

There can be a many-to-many (m:m) relationship between entities, which is represented by creating an associative entity and two 1:m relationships. An associative entity stores data about an m:m relationship. The join operation can be extended from two tables to three or more tables. EXISTS tests whether a table has at least one row that meets a specified condition. NOT EXISTS tests whether all rows in a table do not satisfy a specified condition. Both EXISTS and NOT EXISTS can return *true* or *false*. The relational operation divide, also known as *forall*, can be translated into a double negative. It is represented in SQL by a query containing two NOT EXISTS statements. Set operations enable the results of queries to be combined.

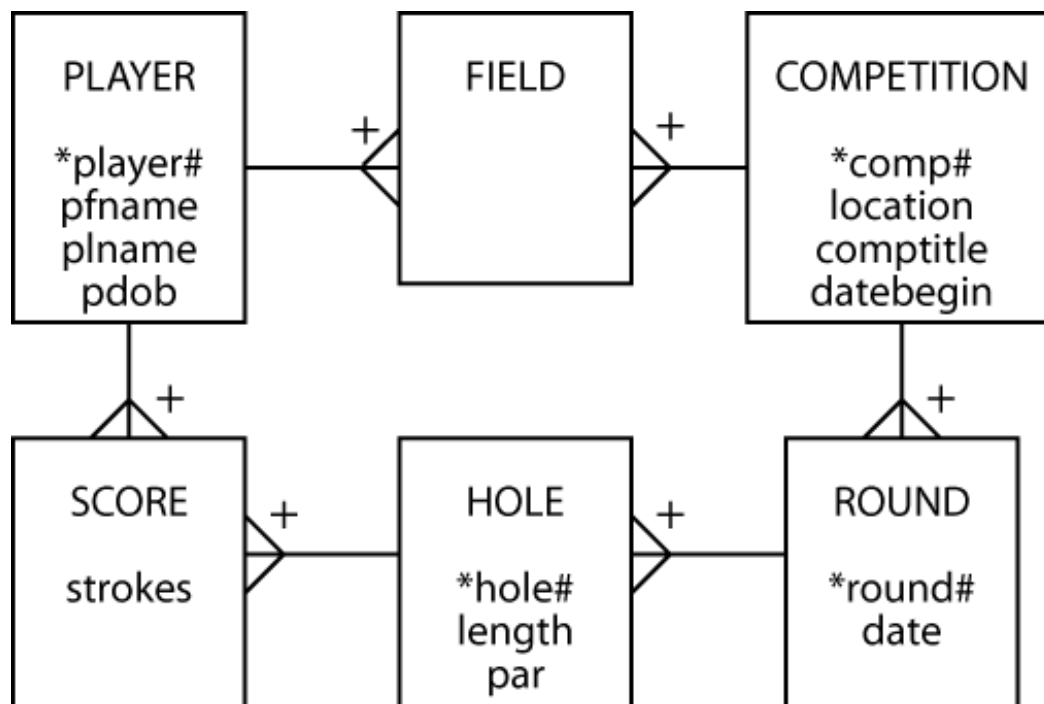
## Key terms and concepts

Associative entity	Many-to-many (m:m) relationship
Divide	NOT EXISTS
Existential quantifier	UNION
EXISTS	Universal quantifier
INTERSECT	

<sup>16</sup>MySQL does not support INTERSECT. Use another AND in the WHERE statement.

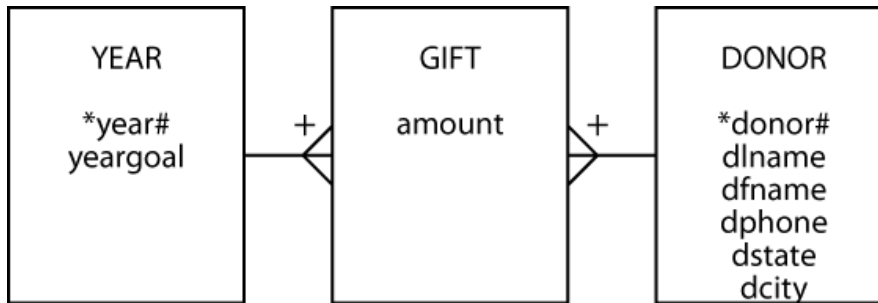
## Exercises

1. Draw data models for the following situations. In each case, think about the names you give each entity:
  - a. Farmers can own cows or share cows with other farmers.
  - b. A track and field meet can have many competitors, and a competitor can participate in more than one event.
  - c. A patient can have many physicians, and a physician can have many patients.
  - d. A student can attend more than one class, and the same class can have many students.
  - e. *The Marathoner*, a monthly magazine, regularly reports the performance of professional marathon runners. It has asked you to design a database to record the details of all major marathons (e.g., Boston, London, and Paris). Professional marathon runners compete in several races each year. A race may have thousands of competitors, but only about 200 or so are professional runners, the ones *The Marathoner* tracks. For each race, the magazine reports a runner's time and finishing position and some personal details such as name, gender, and age.
2. The data model shown was designed by a golf statistician. Write SQL statements to create the corresponding relational database.



3. Write the following SQL queries for the database described in this chapter:
  - a. List the names of items for which the quantity sold is greater than one for any sale.
  - b. Compute the total value of sales for each item by date.
  - c. Report all items of type "F" that have been sold.
  - d. List all items of type "F" that have not been sold.
  - e. Compute the total value of each sale.
4. Why do you have to create a third entity when you have an m:m relationship?

5. What does a plus sign near a relationship arc mean?
6. How does EXISTS differ from other clauses in an SQL statement?
7. Answer the following queries based on the described relational database.



- a. List the phone numbers of donors Hays and Jeffs.
  - b. How many donors are there in the donor table?
  - c. How many people made donations in 1999?
  - d. What is the name of the person who made the largest donation in 1999?
  - e. What was the total amount donated in 2000?
  - f. List the donors who have made a donation every year.
  - g. List the donors whose average donation is more than twice the average donation of all donors.
  - h. List the total amount given by each person across all years; sort the report by the donor's name.
  - i. Report the total donations in 2001 by state.
  - j. In which years did the total donated exceed the goal for the year?
8. The following table records data found on the side of a breakfast cereal carton. Use these data as a guide to develop a data model to record nutrition facts for a meal. In this case, a meal is a cup of cereal and 1/2 cup of skim milk.

---

#### Nutrition facts

Serving size 1 cup (30g)

Servings per container about 17

Amount per serving	Cereal	with 1/2 cup of skim milk
Calories	110	150
Calories from Fat	10	10
		% Daily Value
Total Fat 1g	1%	2%
Saturated Fat 0g	0%	0%
Polyunsaturated Fat 0g		
Monounsaturated Fat 0g		
Cholesterol 0mg	0%	1%
Sodium 220mg	9%	12%
Potassium 105 mg	3%	9%
Total Carbohydrate 24g	8%	10%
Dietary Fiber 3g	13%	13%
Sugars 4g		
Other Carbohydrate 17g		
Protein 3g		

Nutrition facts		
Serving size 1 cup (30g)		
Servings per container about 17		
Vitamin A	10%	15%
Vitamin C	10%	10%
Calcium	2%	15%
Iron	45%	45%
Vitamin D	10%	25%
Thiamin	50%	50%
Riboflavin	50%	50%
Niacin	50%	50%
Vitamin B12	50%	60%
Phosphorus	10%	20%
Magnesium	8%	10%
Zinc	50%	50%
Copper	4%	4%

## 6 One-to-One and Recursive Relationships

*Self-reflection is the school of wisdom.*

Baltasar Gracián, *The Art of Worldly Wisdom*, 1647

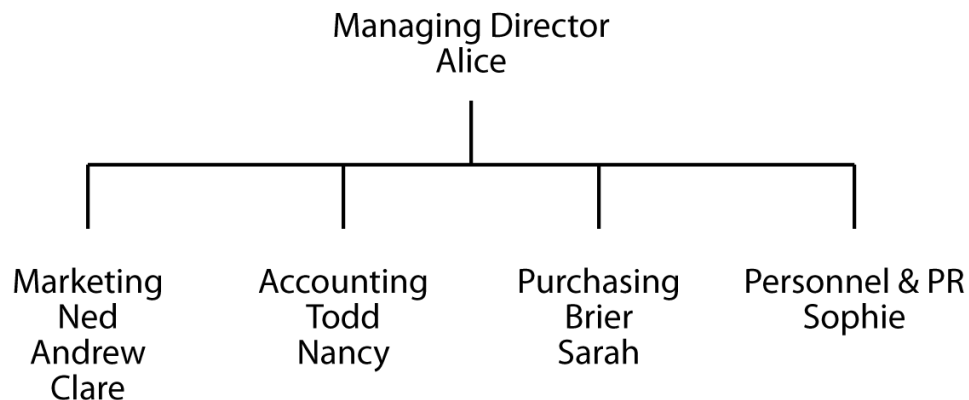
### Learning objectives

Students completing this chapter will be able to

- model one-to-one and recursive relationships;
- define a database with one-to-one and recursive relationships;
- write queries for a database with one-to-one and recursive relationships.

An organizational chart, as shown in the following diagram, can be modeled with several relationships.

*The Expeditioner's organizational chart*

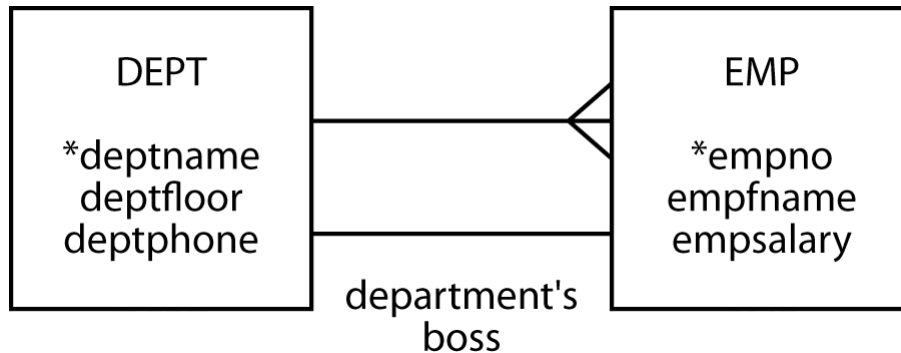




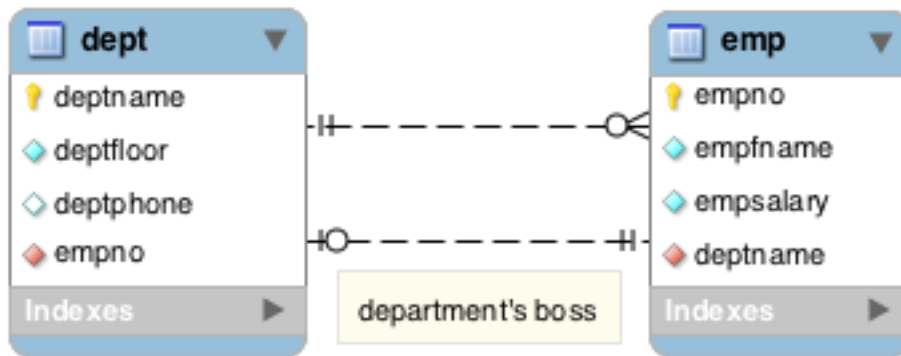
## Modeling a one-to-one relationship

Initially, the organization chart appears to record two relationships. *First*, a department has one or more employees, and an employee belongs to one department. *Second*, a department has one boss, and a person is boss of only one department. That is, boss is a 1:1 relationship between DEPT and EMP. The data model for this situation is shown.

*A data model illustrating a 1:m and 1:1 relationship*



*MySQL Workbench version of a 1:m and 1:1 relationship*

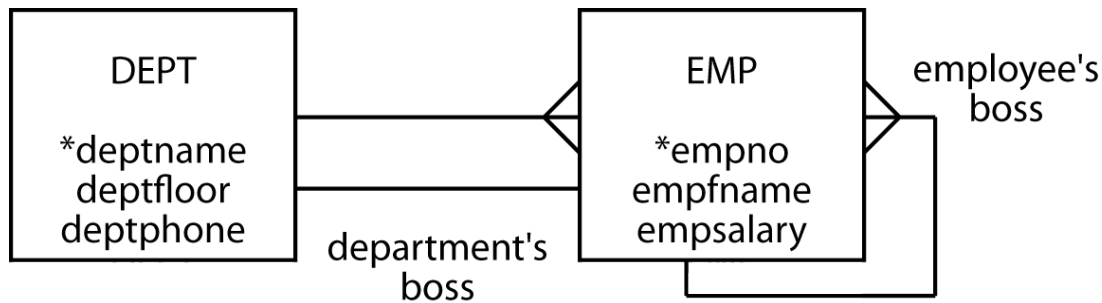


As a general rule, the 1:1 relationship is labeled to avoid confusion because the meaning of such a relationship cannot always be inferred. This label is called a **relationship descriptor**. The 1:m relationship between DEPT and EMP is not labeled because its meaning is readily understood by reading the model. Use a relationship descriptor when there is more than one relationship between entities or when the meaning of the relationship is not readily inferred from the model.

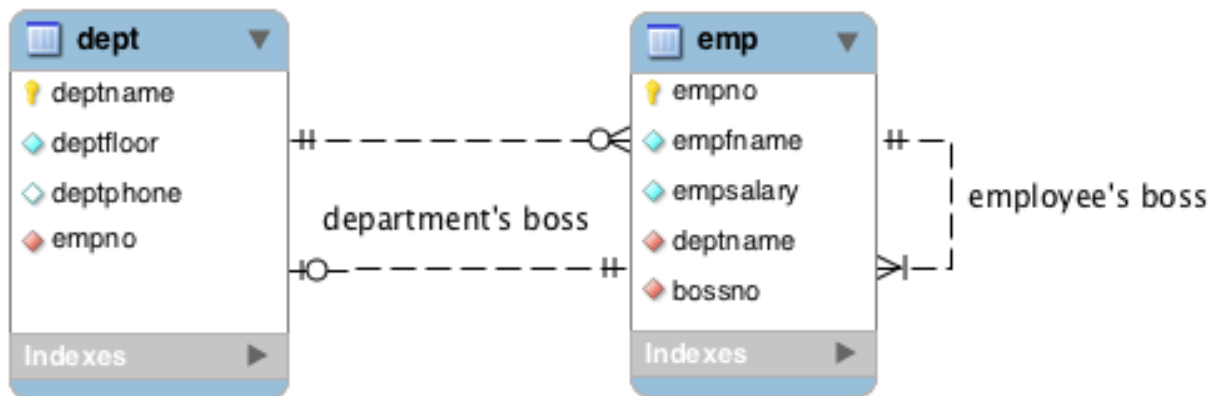
If we think about this problem, we realize there is more to boss than just a department. People also have a boss. Thus, Alice is the boss of all the other employees. In this case, we are mainly interested in who directly bosses someone else. So, Alice is the direct boss of Ned, Todd, Brier, and Sophie. We need to record the person-boss relationship as well as the department-boss relationship.

The person-boss relationship is a **recursive 1:m relationship** because it is a relationship between employees—an employee has one boss and a boss can have many employees. The data model is shown in the following figure.

*A data model illustrating a recursive 1:m relationship*



*MySQL Workbench version of a recursive 1:m relationship*



It is a good idea to label the recursive relationship, because its meaning is often not obvious from the data model.

## Mapping a one-to-one relationship

Since mapping a 1:1 relationship follows the same rules as for any other data model, the major consideration is where to place the foreign key(s). There are three alternatives:

1. Put the foreign key in dept.  
Doing so means that every instance of dept will record the empno of the employee who is boss. Because it is mandatory that all departments, in this case have a boss, the foreign key will always be non-null.
2. Put the foreign key in emp.  
Choosing this alternative means that every instance of emp should record deptname of the department this employee bosses. Since many employees are not bosses, the value of the foreign key column will generally be null.
3. Put a foreign key in both dept and emp.  
The consequence of putting a foreign key in both tables in the 1:1 relationship is the combination of points 1 and 2.

The best approach is to put the foreign key in dept, because it is mandatory for each department to have a boss, and the foreign key will always be non-null.

## Mapping a recursive one-to-many relationship

A recursive 1:m relationship is mapped like a standard 1:m relationship. An additional column, for the foreign key, is created for the entity at the “many” end of the relationship. Of course, in this case the “one”

and “many” ends are the same entity, so an additional column is added to **emp**. This column contains the key **empno** of the “one” end of the relationship. Since **empno** is already used as a column name, a different name needs to be selected. In this case, it makes sense to call the foreign key column **bossno** because it stores the boss’s employee number.

The mapping of the data model is shown in the following table. Note that **deptname** becomes a column in **emp**, the “many” end of the 1:m relationship, and **empno** becomes a foreign key in **dept**, an end of the 1:1 relationship.

*The tables dept and emp*

<u>deptname</u>	<u>deptfloor</u>	<u>deptphone</u>	<u>empno</u>
Management	5	2001	1
Marketing	1	2002	2
Accounting	4	2003	5
Purchasing	4	2004	7
Personnel & PR	1	2005	9

<u>empno</u>	<u>empfname</u>	<u>empsalary</u>	<u>deptname</u>	<u>bossno</u>
1	<i>Alice</i>	75000	<i>Management</i>	
2	<i>Ned</i>	45000	<i>Marketing</i>	1
3	<i>Andrew</i>	25000	<i>Marketing</i>	2
4	<i>Clare</i>	22000	<i>Marketing</i>	2
5	<i>Todd</i>	38000	<i>Accounting</i>	1
6	<i>Nancy</i>	22000	<i>Accounting</i>	5
7	<i>Brier</i>	43000	<i>Purchasing</i>	1
8	<i>Sarah</i>	56000	<i>Purchasing</i>	7
9	<i>Sophie</i>	35000	<i>Personnel &amp; PR</i>	1

If you examine **emp**, you will see that the boss of the employee with **empno** = 2 (Ned) has **bossno** = 1. You can then look up the row in **emp** with **empno** = 1 to find that Ned’s boss is Alice. This “double lookup” is frequently used when manually interrogating a table that represents a recursive relationship. Soon you will discover how this is handled with SQL.

Here is the SQL to create the two tables:

```
CREATE TABLE dept (
    deptname      VARCHAR(15),
    deptfloor     SMALLINT NOT NULL,
    deptphone     SMALLINT NOT NULL,
    empno         SMALLINT NOT NULL,
    PRIMARY KEY(deptname));

CREATE TABLE emp (
    empno         SMALLINT,
    empfname      VARCHAR(10),
    empsalary     DECIMAL(7,0),
    deptname      VARCHAR(15),
    bossno        SMALLINT,
    PRIMARY KEY(empno),
    CONSTRAINT fk_belong_dept FOREIGN KEY(deptname)
        REFERENCES dept(deptname),
```

```
CONSTRAINT fk_has_boss FOREIGN KEY(bossno)
REFERENCES emp(empno));
```

You will notice that there is no foreign key definition for `empno` in `dept` (the 1:1 department's boss relationship). Why? Observe that `deptname` is a foreign key in `emp`. If we make `empno` a foreign key in `dept`, then we have a *deadly embrace*. A new department cannot be added to the `dept` table until there is a boss for that department (i.e., there is a person in the `emp` table with the `empno` of the boss); however, the other constraint states that an employee cannot be added to the `emp` table unless there is a department to which that person is assigned. If we have both foreign key constraints, we cannot add a new department until we have added a boss, and we cannot add a boss until we have added a department for that person. Nothing, under these circumstances, can happen if both foreign key constraints are in place. Thus, only one of them is specified.

In the case of the recursive employee relationship, we can create a constraint to ensure that `bossno` exists for each employee, except of course the person, Alice, who is top of the pyramid. This form of constraint is known as a **self-referential** foreign key. However, we must make certain that the first person inserted into `emp` is Alice. The following statements illustrate that we must always insert a person's boss before we insert the person.

```
INSERT INTO emp (empno, empfname, empsalary, deptname) VALUES (1,'Alice',75000,'Management');
INSERT INTO emp VALUES (2,'Ned',45000,'Marketing',1);
INSERT INTO emp VALUES (3,'Andrew',25000,'Marketing',2);
INSERT INTO emp VALUES (4,'Clare',22000,'Marketing',2);
INSERT INTO emp VALUES (5,'Todd',38000,'Accounting',1);
INSERT INTO emp VALUES (6,'Nancy',22000,'Accounting',5);
INSERT INTO emp VALUES (7,'Brier',43000,'Purchasing',1);
INSERT INTO emp VALUES (8,'Sarah',56000,'Purchasing',7);
INSERT INTO emp VALUES (9,'Sophie',35000,'Personnel',1);
```

In more complex modeling situations, such as when there are multiple relationships between a pair of entities, use of a FOREIGN KEY clause may result in a deadlock. Always consider the consequences of using a FOREIGN KEY clause before applying it.

*Skill builder* A consulting company has assigned each of its employees to a specialist group (e.g., database management). Each specialist group has a team leader. When employees join the company, they are assigned a mentor for the first year. One person might mentor several employees, but an employee has at most one mentor.

## Querying a one-to-one relationship

Querying a 1:1 relationship presents no special difficulties but does allow us to see additional SQL features.

*List the salary of each department's boss.*

```
SELECT empfname, deptname, empsalary FROM emp
WHERE empno IN (SELECT empno FROM dept);
```

or

```
SELECT empfname, dept.deptname, empsalary
FROM emp JOIN dept
ON dept.empno = emp.empno;
```

Table 79: 5 records

empfname	deptname	empsalary
Todd	Accounting	38000
Alice	Management	75000
Ned	Marketing	45000
Sophie	Personnel	35000
Brier	Purchasing	43000

## Querying a recursive 1:m relationship

Querying a recursive relationship is puzzling until you realize that you can join a table to itself by creating two copies of the table. In SQL, you use the WITH clause, also known as the *common table expression (CTE)* to create a temporary copy, a **table alias**. First, use WITH to define two aliases, **wrk** and **boss** for **emp**. Table aliases are required so that SQL can distinguish which copy of the table is referenced. To demonstrate:

*Find the salary of Nancy's boss.*

First, use WITH to define two aliases, **wrk** and **boss** for **emp**.

```
WITH
wrk AS (SELECT * FROM emp),
boss AS (SELECT * FROM emp)
SELECT wrk.empfname, wrk.empsalary, boss.empfname, boss.empsalary
FROM wrk JOIN boss
ON wrk.bossno = boss.empno
WHERE wrk.empfname = 'Nancy';
```

Many queries are solved by getting all the data you need to answer the request in one row. In this case, the query is easy to answer once the data for Nancy and her boss are in the one row. Thus, think of this query as joining two copies of the table **emp** to get the worker and her boss's data in one row. Notice that there is a qualifier (**wrk** and **boss**) for each copy of the table to distinguish between them. It helps to use a qualifier that makes sense. In this case, the **wrk** and **boss** qualifiers can be thought of as referring to the worker and boss tables, respectively. You can understand how the query works by examining the following table illustrating the self-join.

			wrk		boss				
empno	empfname	empsalary	deptname	bossno	empno	empfname	empsalary	deptname	bossno
2	Ned	45000	Marketing	1	1	Alice	75000	Management	
3	Andrew	25000	Marketing	2	2	Ned	45000	Marketing	1
4	Clare	22000	Marketing	2	2	Ned	45000	Marketing	1
5	Todd	38000	Accounting	1	1	Alice	75000	Management	
6	Nancy	22000	Accounting	5	5	Todd	38000	Accounting	1
7	Brier	43000	Purchasing	1	1	Alice	75000	Management	
8	Sarah	56000	Purchasing	7	7	Brier	43000	Purchasing	1
9	Sophie	35000	Personnel & PR	1	1	Alice	75000	Management	

The result of the SQL query is now quite clear once we apply the WHERE clause (see the highlighted row in the preceding table):

Table 81: 1 records

empfname	empsalary	empfname	empsalary
Nancy	22000	Todd	38000

Table 82: 1 records

empfname
Sarah

```
WITH
wrk AS (SELECT * FROM emp),
boss AS (SELECT * FROM emp)
SELECT wrk.empfname, wrk.empsalary, boss.empfname, boss.empsalary
FROM wrk JOIN boss
ON wrk.bossno = boss.empno
WHERE wrk.empfname = 'Nancy';
```

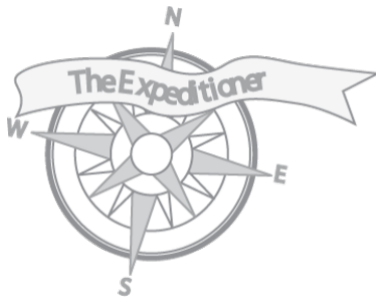
*Find the names of employees who earn more than their boss.*

This would be very easy if the employee and boss data were in the same row. We could simply compare the salaries of the two people. To get the data in the one row, we repeat the self-join with a different WHERE condition. The result is as follows:

```
WITH
wrk AS (SELECT * FROM emp),
boss AS (SELECT * FROM emp)
SELECT wrk.empfname FROM wrk JOIN boss
ON wrk.bossno = boss.empno
WHERE wrk.empsalary > boss.empsalary;
```

### *Skill builder*

Find the name of Sophie's boss.

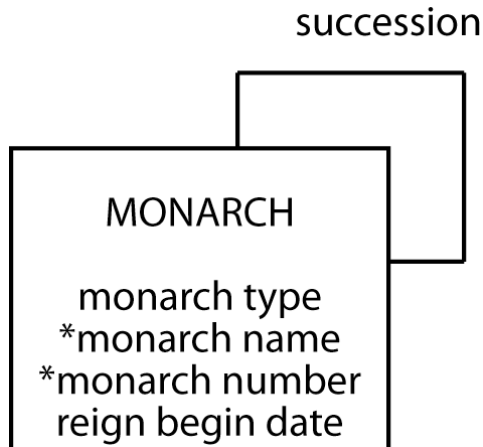


Alice has found several histories of The Expeditioner from a variety of eras. Because many expeditions they outfitted were conducted under royal patronage, it was not uncommon for these histories to refer to British monarchs. Alice could remember very little about British history, let alone when various kings and queens reigned. This sounded like another database problem. She would ask Ned to create a database that recorded details of each monarch. She thought it also would be useful to record details of royal succession.

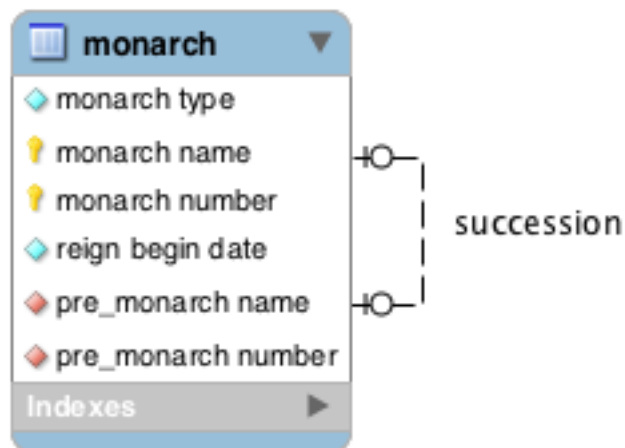
## Modeling a recursive one-to-one relationship

The British monarchy can be represented by a simple one-entity model. A monarch has one direct successor and one direct predecessor. The sequencing of monarchs can be modeled by a recursive 1:1 relationship, shown in the following figure.

*A data model illustrating a recursive 1:1 relationship*



*MySQL Workbench version of a recursive 1:1 relationship*



## Mapping a recursive one-to-one relationship

The recursive 1:1 relationship is mapped by adding a foreign key to **monarch**. You can add a foreign key to represent either the successor or predecessor relationship. In this case, for no particular reason, the preceding relationship is selected. Because each instance of a monarch is identified by a composite key, two columns are added to **monarch** for the foreign key. Data for recent monarchs are shown in the following table.

*The table monarch*

montype	monname	monnum	rgnbeg	premonname	premonnum
King	William	IV	1830/6/26		
Queen	Victoria	I	1837/6/20	William	IV
King	Edward	VII	1901/1/22	Victoria	I

	montype	monname	monnum	rgnbeg	premonname	premonnum
	King	George	V	1910/5/6	Edward	VII
	King	Edward	VIII	1936/1/20	George	V
	King	George	VI	1936/12/11	Edward	VIII
	Queen	Elizabeth	II	1952/02/06	George	VI

The SQL statements to create the table are very straightforward.

```
CREATE TABLE monarch (
  montype      CHAR(5) NOT NULL,
  monname      VARCHAR(15),
  monnum       VARCHAR(5),
  rgnbeg       DATE,
  premonname   VARCHAR(15),
  premonnum    VARCHAR(5),
  PRIMARY KEY (monname, monnum),
  CONSTRAINT fk_monarch FOREIGN KEY (premonname, premonnum)
    REFERENCES monarch(monname, monnum);
```

Because the 1:1 relationship is recursive, you cannot insert Queen Victoria without first inserting King William IV. What you can do is first insert King William, without any reference to the preceding monarch (i.e., a null foreign key). The following code illustrates the order of record insertion so that the referential integrity constraint is obeyed.

```
INSERT INTO MONARCH (montype, monname, monnum, rgnbeg) VALUES ('King', 'William', 'IV', '1830-06-26');
INSERT INTO MONARCH VALUES ('Queen', 'Victoria', 'I', '1837-06-20', 'William', 'IV');
INSERT INTO MONARCH VALUES ('King', 'Edward', 'VII', '1901-01-22', 'Victoria', 'I');
INSERT INTO MONARCH VALUES ('King', 'George', 'V', '1910-05-06', 'Edward', 'VII');
INSERT INTO MONARCH VALUES ('King', 'Edward', 'VIII', '1936-01-20', 'George', 'V');
INSERT INTO MONARCH VALUES ('King', 'George', 'VI', '1936-12-11', 'Edward', 'VIII');
INSERT INTO MONARCH VALUES ('Queen', 'Elizabeth', 'II', '1952-02-06', 'George', 'VI');
```

### *Skill builder*

In a competitive bridge competition, the same pair of players play together for the entire tournament. Draw a data model to record details of all the players and the pairs of players.

## Querying a recursive one-to-one relationship

Some queries on the monarch table demonstrate querying a recursive 1:1 relationship.

*Who preceded Elizabeth II?*

```
SELECT premonname, premonnum FROM monarch
WHERE monname = 'Elizabeth' and monnum = 'II';
```

This is simple because all the data are in one row. A more complex query is:

*Was Elizabeth II's predecessor a king or queen?*



Table 84: 1 records

premonname	premonnum
George	VI

Table 85: 1 records

montype
King

```
WITH
cur AS (SELECT * FROM monarch),
pre AS (SELECT * FROM monarch)
SELECT pre.montype FROM cur JOIN pre
  ON cur.premonname = pre.monname AND cur.premonnum = pre.monnum
  WHERE cur.monname = 'Elizabeth'
  AND cur.monnum = 'II';
```

Notice in the preceding query how to specify the ON clause when you have a composite key.

This is very similar to the query to find the salary of Nancy's boss. The `monarch` table is joined with itself to create a row that contains all the details to answer the query.

*List the kings and queens of England in ascending chronological order.*

```
SELECT montype, monname, monnum, rgnbeg
  FROM monarch ORDER BY rgnbeg;
```

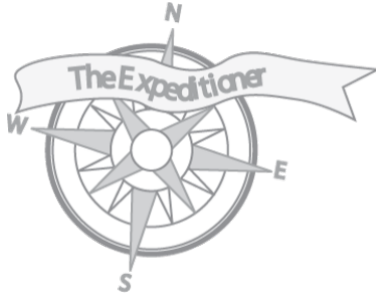
This is a simple query because `rgnbeg` is like a ranking column. It would not be enough to store just the year in `rgnbeg`, because two kings started their reigns in 1936; hence, the full date is required.

*Skill builder*

Who succeeded Queen Victoria?

Table 86: 7 records

montype	monname	monnum	rgnbeg
King	William	IV	1830-06-26
Queen	Victoria	I	1837-06-20
King	Edward	VII	1901-01-22
King	George	V	1910-05-06
King	Edward	VIII	1936-01-20
King	George	VI	1936-12-11
Queen	Elizabeth	II	1952-02-06

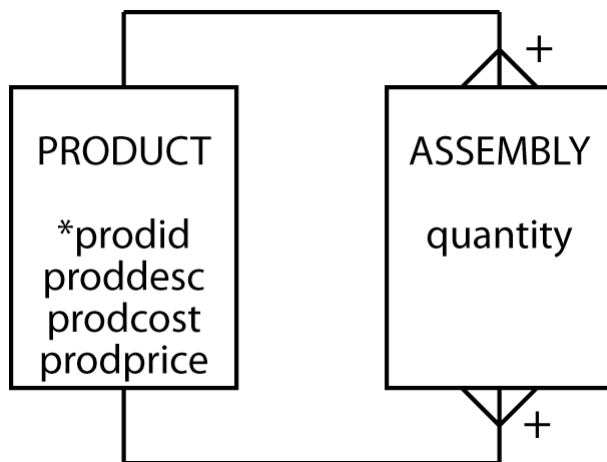


Of course, Alice soon had another project for Ned. The Expeditioner keeps a wide range of products that are sometimes assembled into kits to make other products. For example, the animal photography kit is made up of eight items that The Expeditioner also sells separately. In addition, some kits became part of much larger kits. The animal photography kit is included as one of 45 items in the East African Safari package. All of the various items are considered products, and each has its own product code. Ned was now required to create a product database that would keep track of all the items in The Expeditioner's stock.

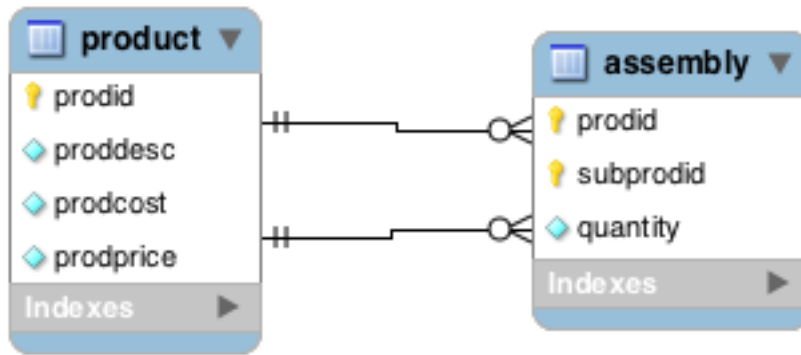
## Modeling a recursive many-to-many relationship

The assembly of products to create other products is very common in business. Manufacturing even has a special term to describe it: a bill of materials. The data model is relatively simple once you realize that a product can appear as part of many other products and can be composed of many other products; that is, we have a recursive many-to-many (m:m) relationship for product. As usual, we turn an m:m relationship into two one-to-many (1:m) relationships. Thus, we get the data model displayed in the following figure.

*A data model illustrating a recursive m:m relationship*



*MySQL Workbench version of a recursive m:m relationship*



Tables product and assembly

	<u>prodid</u>	proddesc	prodcost	prodprice
1000	Animal photography kit			725
101	Camera		150	300
102	Camera case		10	15
103	70-210 zoom lens		125	200
104	28-85 zoom lens		115	185
105	Photographer's vest		25	40
106	Lens cleaning cloth		1	1.25
107	Tripod		35	45
108	16 GB SDHC memory card		30	37

quantity	<u>prodid</u>	<u>subprodid</u>
1	1000	101
1	1000	102
1	1000	103
1	1000	104
1	1000	105
2	1000	106
1	1000	107
10	1000	108

## Mapping a recursive many-to-many relationship

Mapping follows the same procedure described previously, producing the two tables shown below. The SQL statements to create the tables are shown next. Observe that **assembly** has a composite key, and there are two foreign key constraints.

```

CREATE TABLE product (
    prodid          INTEGER,
    proddesc        VARCHAR(30),
    prodcost        DECIMAL(9,2),
    prodprice       DECIMAL(9,2),
    PRIMARY KEY(prodid));
  
```

Table 89: 8 records

subprodid
101
102
103
104
105
106
107
108

```
CREATE TABLE assembly (
  quantity      INTEGER NOT NULL,
  prodid        INTEGER,
  subprodid     INTEGER,
  PRIMARY KEY (prodid, subprodid),
  CONSTRAINT fk_assembly_product FOREIGN KEY (prodid)
    REFERENCES product (prodid),
  CONSTRAINT fk_assembly_subproduct FOREIGN KEY (subprodid)
    REFERENCES product (prodid));
```

#### *Skill builder*

An army is broken up into many administrative units (e.g., army, brigade, platoon). A unit can contain many other units (e.g., a regiment contains two or more battalions), and a unit can be part of a larger unit (e.g., a squad is a member of a platoon). Draw a data model for this situation.

## Querying a recursive many-to-many relationship {-}

*List the product identifier of each component of the animal photography kit.*

```
SELECT subprodid FROM product JOIN assembly
  ON product.prodid = assembly.prodid
 WHERE proddesc = 'Animal photography kit';
```

Why are the values for `subprodid` listed in no apparent order? Remember, there is no implied ordering of rows in a table, and it is quite possible, as this example illustrates, for the rows to have what appears to be an unusual ordering. If you want to order rows, use the `ORDER BY` clause.

*List the product description and cost of each component of the animal photography kit.*

```
SELECT proddesc, prodcost FROM product
  WHERE prodid IN
    (SELECT subprodid FROM product JOIN assembly
      ON product.prodid = assembly.prodid
     WHERE proddesc = 'Animal photography kit');
```

In this case, first determine the `prodid` of those products in the animal photography kit (the inner query), and then report the description of these products. Alternatively, a three-way join can be done using two copies of `product`.

Table 90: 8 records

proddesc	prodcost
35mm camera	150.00
Camera case	10.00
70-210 zoom lens	125.00
28-85 zoom lens	115.00
Photographers vest	25.00
Lens cleaning cloth	1.00
Tripod	35.00
24/100 ASA 35mm color neg film	0.85

```

WITH
a AS (SELECT * FROM product),
b AS (SELECT * FROM product)
SELECT b.proddesc, b.prodcost FROM a JOIN assembly
      ON a.prodid = assembly.prodid
      JOIN b
      ON assembly.subprodid = b.prodid
      WHERE a.proddesc = 'Animal photography kit';

```

### *Skill builder*

How many lens cleaning cloths are there in the animal photography kit?

## Summary

Relationships can be one-to-one and recursive. A recursive relationship is within a single entity rather than between entities. Recursive relationships are mapped to the relational model in the same way as other relationships. A self-referential foreign key constraint permits a foreign key reference to a key within the same table. Resolution of queries involving recursive relationships often requires a table to be joined with itself. Recursive many-to-many relationships occur in business in the form of a bill of materials.

## Key terms and concepts

JOIN	Relationship
One-to-many (1:m) relationship	Relationship descriptor
One-to-one (1:1) relationship	Self-join
Recursive 1:1 relationship	Self-referential foreign key
Recursive 1:m relationship	Theta-join
Recursive m:m relationship	Update anomalies
Recursive relationship	Views
Referential integrity	Virtual table

## Exercise

1. Draw data models for the following two problems:

- a. (i) A dairy farmer, who is also a part-time cartoonist, has several herds of cows. He has assigned each cow to a particular herd. In each herd, the farmer has one cow that is his favorite—often that cow is featured in a cartoon.
  - (ii) A few malcontents in each herd, mainly those who feel they should have appeared in the cartoon, disagree with the farmer's choice of a favorite cow, whom they disparagingly refer to as the sacred cow. As a result, each herd now has elected a herd leader.
  - b. The originator of a pyramid marketing scheme has a system for selling ethnic jewelry. The pyramid has three levels—gold, silver, and bronze. New associates join the pyramid at the bronze level. They contribute 30 percent of the revenue of their sales of jewelry to the silver chief in charge of their clan. In turn, silver chiefs contribute 30 percent of what they receive from bronze associates to the gold master in command of their tribe. Finally, gold masters pass on 30 percent of what they receive to the originator of the scheme.
  - c. The legion, the basic combat unit of the ancient Roman army, contained 3,000 to 6,000 men, consisting primarily of heavy infantry (hoplites), supported by light infantry (velites), and sometimes by cavalry. The hoplites were drawn up in three lines. The hastati (youngest men) were in the first, the principes (seasoned troops) in the second, and the triarii (oldest men) behind them, reinforced by velites. Each line was divided into 10 maniples, consisting of two centuries (60 to 80 men per century) each. Each legion had a commander, and a century was commanded by a centurion. Julius Caesar, through one of his Californian channelers, has asked you to design a database to maintain details of soldiers. Of course, Julius is a little forgetful at times, and he has not supplied the titles of the officers who command maniples, lines, and hoplites, but he expects that you can handle this lack of fine detail.
  - d. A travel agency is frequently asked questions about tourist destinations. For example, customers want to know details of the climate for a particular month, the population of the city, and other geographic facts. Sometimes they request the flying time and distance between two cities. The manager has asked you to create a database to maintain these facts.
  - e. The Center for the Study of World Trade keeps track of trade treaties between nations. For each treaty, it records details of the countries signing the treaty and where and when it was signed.
  - f. Design a database to store details about U.S. presidents and their terms in office. Also, record details of their date and place of birth, gender, and political party affiliation (e.g., Caluthumpian Progress Party). You are required to record the sequence of presidents so that the predecessor and successor of any president can be identified. How will you model the case of Grover Cleveland, who served nonconsecutive terms as president? Is it feasible that political party affiliation may change? If so, how will you handle it?
  - g. The IS department of a large organization makes extensive use of software modules. New applications are built, where possible, from existing modules. Software modules can also contain other modules. The IS manager realizes that she now needs a database to keep track of which modules are used in which applications or other modules. (Hint: It is helpful to think of an application as a module.)
  - h. Data modeling is finally getting to you. Last night you dreamed you were asked by Noah to design a database to store data about the animals on the ark. All you can remember from Sunday school is the bit about the animals entering the ark two-by-two, so you thought you should check the real thing.  
Take with you seven pairs of every kind of clean animal, a male and its mate, and two of every kind of unclean animal, a male and its mate, and also seven pair of every kind of bird, male and female. Genesis 7:2  
Next time Noah disturbs your sleep, you want to be ready. So, draw a data model and make certain you record the two-by-two relationship.
2. Write SQL to answer the following queries using the DEPT and EMP tables described in this chapter:
    - a. Find the departments where all the employees earn less than their boss.

- b. Find the names of employees who are in the same department as their boss (as an employee).
  - c. List the departments having an average salary greater than \$25,000.
  - d. List the departments where the average salary of the employees, excluding the boss, is greater than \$25,000.
  - e. List the names and manager of the employees of the Marketing department who have a salary greater than \$25,000.
  - f. List the names of the employees who earn more than any employee in the Marketing department.
3. Write SQL to answer the following queries using the monarch table described in this chapter:
  - a. Who succeeded Victoria I?
  - b. How many days did Victoria I reign?
  - c. How many kings are there in the table?
  - d. Which monarch had the shortest reign?
4. Write SQL to answer the following queries using the product and assembly tables:
  - a. How many different items are there in the animal photography kit?
  - b. What is the most expensive item in the animal photography kit?
  - c. What is the total cost of the components of the animal photography kit?
  - d. Compute the total quantity for each of the items required to assemble 15 animal photography kits.

## 7 Data Modeling

*Man is a knot, a web, a mesh into which relationships are tied. Only those relationships matter.*

Antoine de Saint-Exupéry in *Flight to Arras*

### Learning objectives

Students completing this chapter will be able to create a well-formed, high-fidelity data model.

### Modeling

Modeling is widely used within business to learn about organizational problems and design solutions. To understand where data modeling fits within the broader context, it is useful to review the full range of modeling activities. The types of modeling methods follow the 5W-H model of journalism.<sup>17</sup>

*A broad perspective on modeling*

		Scope	Model	Technology
<b>Motivation</b>	Why	Goals	Business plan canvas	Groupware
<b>People</b>	Who	Business units	Organization chart	System interface
<b>Time</b>	When	Key events	PERT chart	Scheduling
<b>Data</b>	What	Key entities	Data model	Relational database

<sup>17</sup> Roberto Franzosi, “On Quantitative Narrative Analysis,” in *Varieties of Narrative Analysis*, ed. James A. Holstein and Jaber F. Gubrium (SAGE Publications, Inc., 2012), 75–96.

		Scope	Model	Technology
<b>Network</b>	Where	Locations	Logistics network	System architecture
<b>Function</b>	How	Key processes	Process model	Application software

Modeling occurs at multiple levels. At the highest level, an organization needs to determine the scope of its business by identifying the major elements of its environment, such as its goals, business units, where it operates, and critical events, entities, and processes. At the top level, textual models are frequently used. For example, an organization might list its major goals and business processes. A map will be typically used to display where the business operates.

Once senior managers have clarified the scope of a business, models can be constructed for each of the major elements. Goals will be converted into a business plan, business units will be shown on an organizational chart, and so on. The key elements, from an IS perspective, are data and process modeling, which are typically covered in the core courses of an IS program.

Technology, the final stage, converts models into operational systems to support the organization. Many organizations rely extensively on e-mail and Web technology for communicating business decisions. People connect to systems through an interface, such as a Web browser. Ensuring that events occur at the right time is managed by scheduling software. This could be implemented with operating system procedures that schedule the execution of applications. In some database management systems (DBMSs), triggers can be established. A **trigger** is a database procedure that is automatically executed when some event is recognized. For example, U.S. banks are required to report all deposits exceeding USD 10,000, and a trigger could be coded for this event.

Data models are typically converted into relational databases, and process models become computer programs. Thus, you can see that data modeling is one element of a comprehensive modeling activity that is often required to design business systems. When a business undergoes a major change, such as a reengineering project, many dimensions can be altered, and it may be appropriate to rethink many elements of the business, starting with its goals. Because such major change is very disruptive and costly, it occurs less frequently. It is more likely that data modeling is conducted as a stand-alone activity or part of process modeling to create a new business application.

## Data modeling

You were introduced to the basic building blocks of data modeling in the earlier chapters of this section. Now it is time to learn how to assemble blocks to build a data model. Data modeling is a method for determining what data and relationships should be stored in the database. It is also a way of communicating a database design.

The goal of data modeling is to identify the facts that must be stored in a database. A data model is not concerned with how the data will be stored. This is the concern of those who implement the database. A data model is not concerned with how the data will be processed. This is the province of process modeling. The goal is to create a data model that is an accurate representation of data needs and real-world data relationships.

Building a data model is a partnership between a client, a representative of the eventual owners of the database, and a designer. Of course, there can be a team of clients and designers. For simplicity, we assume there is one client and one designer.

Drawing a data model is an iterative process of trial and revision. A data model is a working document that will change as you learn more about the client's needs and world. Your early versions are likely to be quite different from the final product. Use a product such as MySQL Workbench for drawing and revising a data model.



## The building blocks

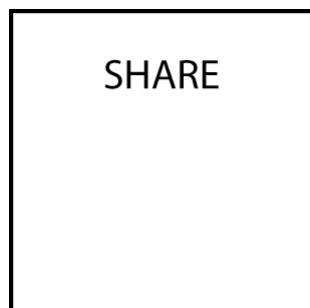
The purpose of a database is to store data about things, which can include facts (e.g., an exchange rate), plans (e.g., scheduled production of two-person tents for June), estimates (e.g., forecast of demand for smart phones), and a variety of other data. A data model describes these things and their relationships with other things using four components: entity, attribute, relationship, and identifier.

### Entity

The entity is the basic building block of a data model. an entity is a thing about which data should be stored, something we need to describe. Each entity in a data model has a unique name that we write in singular form. Why singular? Because we want to emphasize that an entity describes an instance of a thing. Thus, we previously used the word SHARE to define an entity because it describes each instance of a share rather than shares in general.

We have already introduced the convention that an entity is represented by a rectangle, and the name of the entity is shown in uppercase letters, as follows.

*The entity SHARE*



A large data model might contain over a 1,000 entities. A database can easily contain hundreds of millions of instances (rows) for any one entity (table). Imagine the number of instances in a national tax department's database.

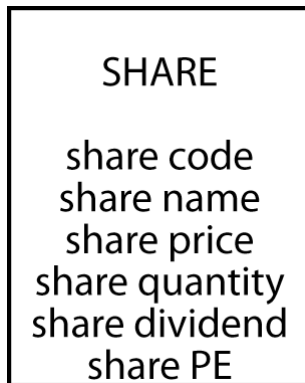
How do you begin to identify entities? One approach is to underline any *nouns* in the system proposal or provided documentation. Most nouns are possible entities, and underlining ensures that you do not overlook any potential entities. Start by selecting an entity that seems central to the problem. If you were designing a student database, you might start with the student. Once you have picked a central entity, describe it. Then move to the others.

### Attribute

An attribute describes an entity. When an entity has been identified, the next step is to determine its attributes, that is, the data that should be kept to describe fully the entity. An attribute name is singular and unique within the data model. You may need to use a modifier to make an attribute name unique (e.g., “hire date” and “sale date” rather than just “date”).

Our convention is that the name of an attribute is recorded in lowercase letters within the entity rectangle. The following figure illustrates that SHARE has attributes *share code*, *share name*, *share price*, *share quantity*, *share dividend*, and *share PE*. Notice the frequent use of the modifier “share”. It is possible that other entities in the database might also have a price and quantity, so we use a modifier to create unique attribute names. Note also that *share code* is the identifier.

*The entity SHARE with its attributes*



Defining attributes generally takes considerable discussion with the client. You should include any attribute that is likely to be required for present or future decision making, but don't get carried away. Avoid storing unnecessary data. For example, to describe the entity **STUDENT** you usually record date of birth, but it is unlikely that you would store height. If you were describing an entity **PATIENT**, on the other hand, it might be necessary to record height, because that is sometimes relevant to medical decision making.

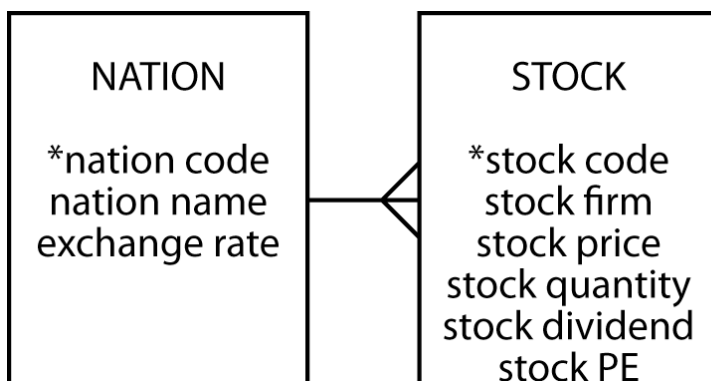
An attribute has a single value, which may be null. Multiple values are not allowed. If you need to store multiple values, it is a signal that you have a one-to-many (1:m) relationship and need to define another entity.

## Relationship

Entities are related to other entities. If there were no relationships between entities, there would be no need for a data model and no need for a relational database. A simple, flat file (a single-entity database) would be sufficient. A relationship is binary. It describes a linkage between two entities and is represented by a line between them.

Because a relationship is binary, strictly speaking it has two relationship descriptors, one for each entity. Each relationship descriptor has a degree stating how many instances of the other entity may be related to each instance of the described entity. Consider the entities **STOCK** and **NATION** and their 1:m relationship (see the following figure). We have two relationship descriptors: *stocks of nation* for **NATION** and *nation of stock* for **STOCK**. The relationship descriptor *stocks of nation* has a degree of m because a nation may have zero or more listed stocks. The relationship descriptor *nation of stock* has a degree of 1 because a stock is listed in at most one nation. To reduce data model clutter, where necessary for clarification, we will use a single label for the pair of relationship descriptors. Experience shows this approach captures the meaning of the relationship and improves readability.

*A 1:m relationship between STOCK and NATION*

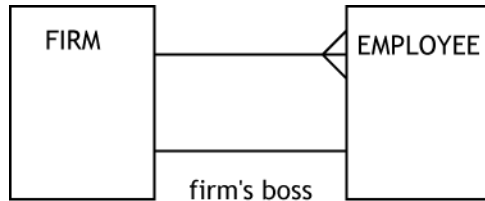


Because the meaning of relationships frequently can be inferred, there is no need to label every one; however,

each additional relationship between two entities should be labeled to clarify meaning. Also, it is a good idea to label one-to-one (1:1) relationships, because the meaning of the relationship is not always obvious.

Consider the fragment in the following figure. The descriptors of the 1:m relationship can be inferred as a firm has employees and an employee belongs to a firm. the 1:1 relationship is clarified by labeling the 1:1 relationship as firm's boss.

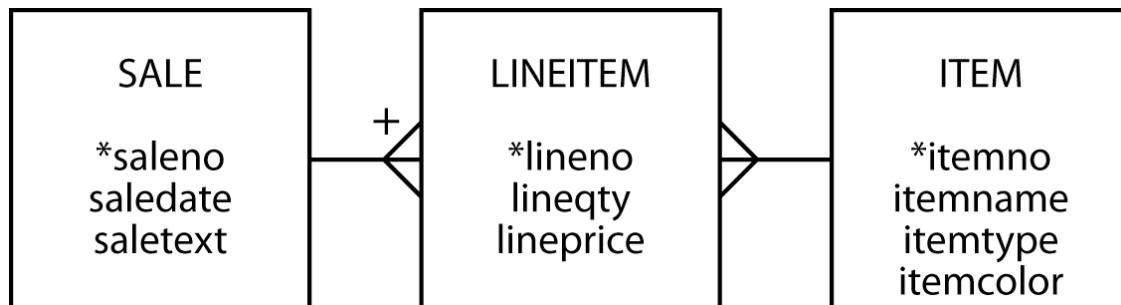
*Relationship labeling*



## Identifier

An identifier uniquely distinguishes an instance of an entity. An identifier can be one or more attributes and may include the identifier of a related entity. When a related entity's identifier is part of an identifier, a plus sign is placed on the line closest to the entity being identified. In the following figure, an instance of the entity LINEITEM is identified by the composite of *lineno* and *saleno*, the identifier of SALE. The value *lineno* does not uniquely identify an instance of LINEITEM, because it is simply a number that appears on the sales form. Thus, the identifier must include *saleno* (the identifier of SALE). So, any instance of LINEITEM is uniquely identified by the composite *saleno* and *lineno*.

*A related entity's identifier as part of the identifier*



Occasionally, there will be several possible identifiers, and these are each given a different symbol. Our convention is to prefix an identifier with an asterisk (\*). If there are multiple identifiers, use other symbols such as #, !, or &. Be careful: If you have too many possible identifiers, your model will look like comic book swearing. In most cases, you will have only one identifier.

No part of an identifier can be null. If this were permitted, there would be no guarantee that the other parts of the identifier were sufficiently unique to distinguish all instances in the database.

## Data model quality

There are two criteria for judging the quality of a data model. It must be well-formed and have high fidelity.

### A well-formed data model

A well-formed data model clearly communicates information to the client. Being well-formed means the construction rules have been obeyed. There is no ambiguity; all entities are named, and all entities have

identifiers. If identifiers are missing, the client may make incorrect inferences about the data model. All relationships are recorded using the proper notation and labeled whenever there is a possibility of confusion.

#### *Characteristics of a well-formed data model*

---

All construction rules are obeyed.

---

There is no ambiguity.

All entities are named.

Every entity has an identifier.

All relationships are represented, using the correct notation.

Relationships are labeled to avoid misunderstanding.

All attributes of each entity are listed.

All attribute names are meaningful and unique.

---

All the attributes of an entity are listed because missing attributes create two types of problems. *First*, it is unclear what data will be stored about each instance. *Second*, the data model may be missing some relationships. Attributes that can have multiple values become entities. It is only by listing all of an entity's attributes during data modeling that these additional entities are recognized.

In a well-formed data model, all attribute names are meaningful and unique. The names of entities, identifiers, attributes, and relationships must be meaningful to the client because they describe the client's world. Indeed, in nearly all cases they are the client's everyday names. Take care in selecting words because they are critical to communicating meaning. The acid test for comprehension is to get the client to read the data model to other potential users. Names need to be unique to avoid confusion.

### **A high-fidelity image**

Music lovers aspire to own a high-fidelity stereo system—one that faithfully reproduces the original performance with minimal or no distortion. A data model is a high-fidelity image when it faithfully describes the world it is supposed to represent. All relationships are recorded and are of the correct degree. There are no compromises or distortions. If the real-world relationship is many-to-many (m:m), then so is the relationship shown in the data model. A well-formed, high-fidelity data model is complete, understandable, accurate, and syntactically correct.

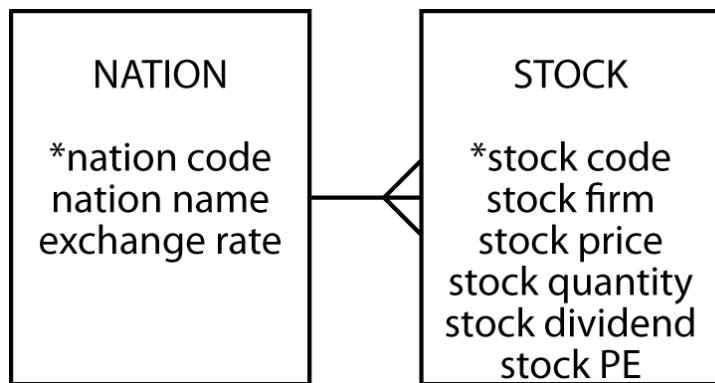
### **Quality improvement**

A data model is an evolving representation. Each change should be an incremental improvement in quality. Occasionally, you will find a major quality problem at the data model's core and have to change the model considerably.

### **Detail and context**

The quality of a data model can be determined only by understanding the context in which it will be used. Consider the data model (see the following figure) used in Chapter 4 to discuss the 1:m relationship. This fragment says a nation has many stocks. Is that really what we want to represent? Stocks are listed on a stock exchange, and a nation may have several stock exchanges. For example, the U.S. has the New York Stock Exchange and NASDAQ. Furthermore, some foreign stocks are listed on the New York Stock Exchange. If this is the world we have to describe, the data model is likely to differ from that shown in the figure. Try drawing the revised data model.

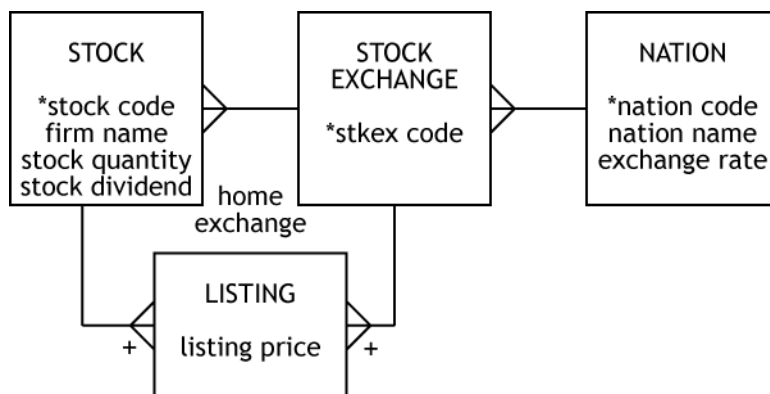
*A 1:m relationship between STOCK and NATION*



As you can see, the data model in the following figure is quite different from the initial data model of preceding figure. Which one is better? They both can be valid models; it just depends on the world you are trying to represent and possible future queries. In the first case, the purpose was to determine the value of a portfolio. There was no interest in on which exchange stocks were listed or their home exchange. So the first model has high fidelity for the described situation.

The second data model would be appropriate if the client needed to know the home exchange of stocks and their price on the various exchanges where they were listed. The second data model is an improvement on the first if it incorporates the additional facts and relationships required. If it does not, then the additional detail is not worth the extra cost.

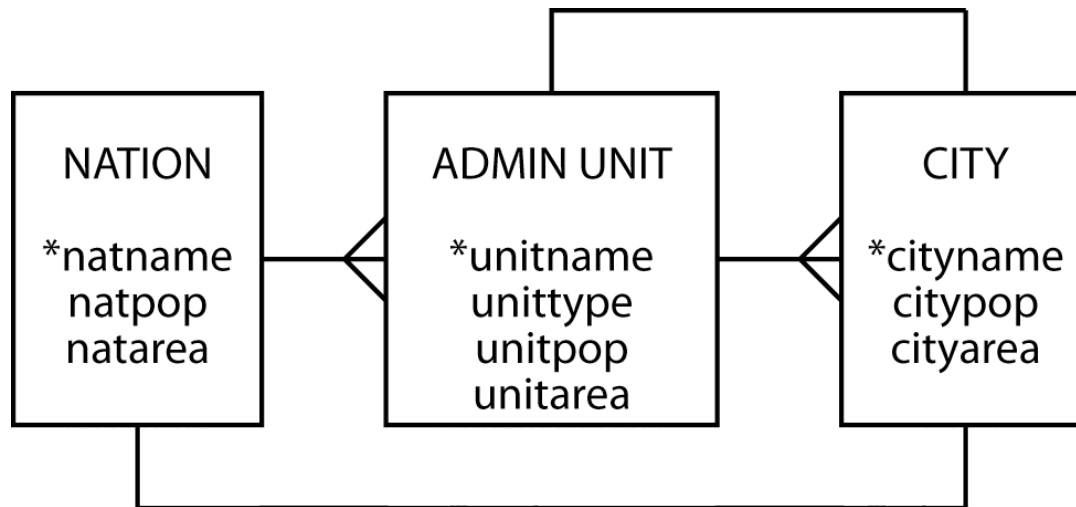
*Revised NATION-STOCK data*



## A lesson in pure geography

A data model must be an accurate representation of the world you want to model. The data model must account for all the exceptions—there should be no impurities. Let's say you want to establish a database to store details of the world's cities. You might start with the data model depicted in the following figure.

*A world's cities data model*



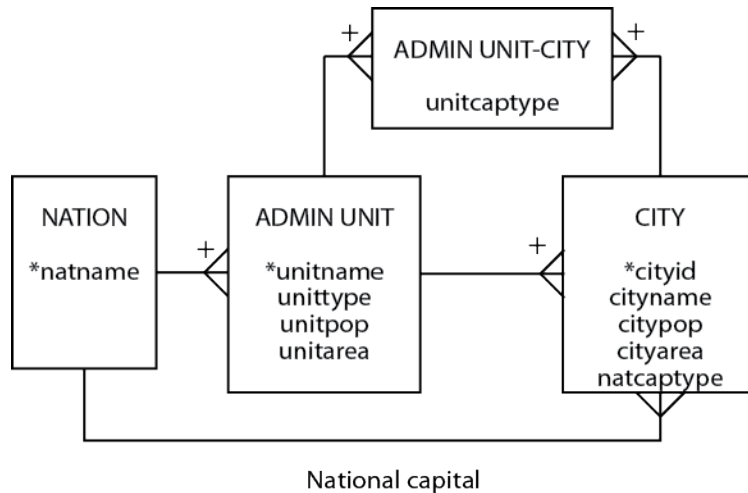
Before looking more closely at the data model, let's clarify the meaning of *unittype*. Because most countries are divided into administrative units that are variously called states, provinces, territories, and so forth, *unittype* indicates the type of administrative unit (e.g., state). How many errors can you find in the initial data model?

Problems and solutions for the initial world's cities data model

	Problem	Solution
1	City names are not unique. There is an Athens in Greece and the U.S. has an Athens in Georgia, Alabama, Ohio, Pennsylvania, Tennessee, and Texas.	To identify a city uniquely, you need to specify its administrative unit and country. Add a plus sign to the crow's feet between CITY and ADMIN UNIT and between ADMIN UNIT and NATION.
2	Administrative unit names are not necessarily unique. There used to be a Georgia in the old U.S.S.R., and there is a Georgia in the U.S. There is no guarantee that administrative unit names will always be unique.	To identify an administrative unit uniquely, you also need to know the country in which it is found. Add a plus sign to the crow's foot between ADMIN UNIT and NATION.
3	There is an unlabeled 1:1 relationship between CITY and ADMIN UNIT and CITY and NATION. What do these mean? They are supposed to indicate that an administrative unit has a capital, and a nation has a capital.	Label relationships (e.g., national capital city).
4	The assumption is that a nation has only one capital, but there are exceptions. South Africa has three capitals: Cape Town (legislative), Pretoria (administrative), and Bloemfontein (judicial). You need only one exception to lower the fidelity of a data model significantly.	Change the relationship between NATION and CITY to 1:m. Add an attribute natcaptype to CITY to distinguish between types of capitals.

	Problem	Solution
5	The assumption is that an administrative unit has only one capital, but there are exceptions. At one point, Chandigarh was the capital of two Indian states, Punjab and Haryana. The Indian state of Jammu & Kashmir has two capitals: Jammu (summer) and Srinagar (winter).	Change the relationship between ADMIN UNIT and CITY to m:m by creating an associative entity and include a distinguishing attribute of unitcaptype.
6	Some values can be derived. National population, natpop, and area, natarea, are the sum of the regional populations, regpop, and areas, regarea, respectively. The same rule does not apply to regions and cities because not everyone lives in a city.	Remove the attributes natpop and natarea from NATION.

*A revised world's cities data model*



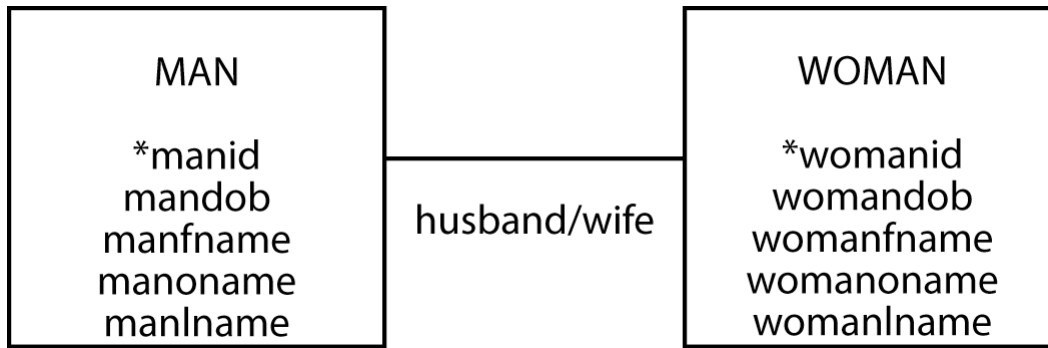
This geography lesson demonstrates how you often start with a simple model of low fidelity. By additional thinking about relationships and consideration of exceptions, you gradually create a high-fidelity data model.

**Skill builder** Write SQL to determine which administrative units have two capitals and which have a shared capital.

## Family matters

Families can be very complicated. It can be tricky to keep track of all those relations—maybe you need a relational database (this is the worst joke in the book; they improve after this). We start with a very limited view of marriage and gradually ease the restrictions to demonstrate how any and all aspects of a relationship can be modeled. An initial fragment of the data model is shown in the following figure. There are several things to notice about it.

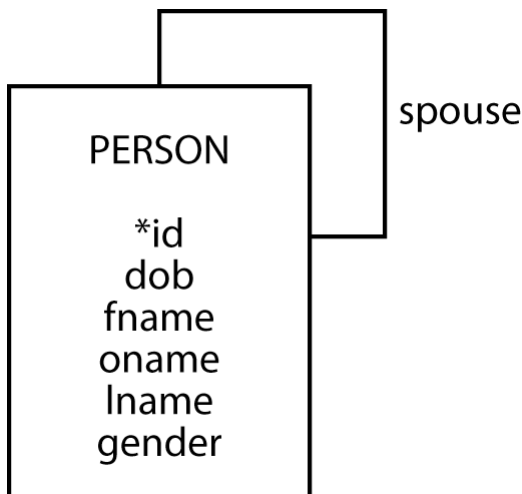
*A man-woman data model*



1. There is a 1:1 relationship between man and woman. The labels indicate that this relationship is marriage, but there is no indication of the marriage date. We are left to infer that the data model records the current marriage.
2. MAN and WOMAN have the same attributes. A prefix is used to make them unique. Both have an identifier (*id*, in the United States this would probably be a Social Security number), date of birth (*dob*), first name (*fname*), other names (*oname*), and last name (*lname*).
3. Other names (*oname*) looks like a multivalued attribute, which is not allowed. Should *oname* be single or multivalued? This is a tricky decision. It depends on how these data will be used. If queries such as “find all men whose other names include Herbert” are likely, then a person’s other names—kept as a separate entity that has a 1:m relationship for MAN and WOMAN—must be established. That is, a man or woman can have one or more other names. However, if you just want to store the data for completeness and retrieve it in its entirety, then *oname* is fine. It is just a text string. The key question to ask is, “What is the lowest level of detail possibly required for future queries?” Also, remember that SQL’s REGEXP clause can be used to search for values within a text string.

The use of a prefix to distinguish attribute names in different entities suggests you should consider combining the entities. In this case, we could combine the entities MAN and WOMAN to create an entity called PERSON. Usually when entities are combined, you have to create a new attribute to distinguish between the different types. In this example, the attribute *gender* is added. We can also generalize the relationship label to *spouse*. The revised data model appears in the following figure.

A PERSON data model

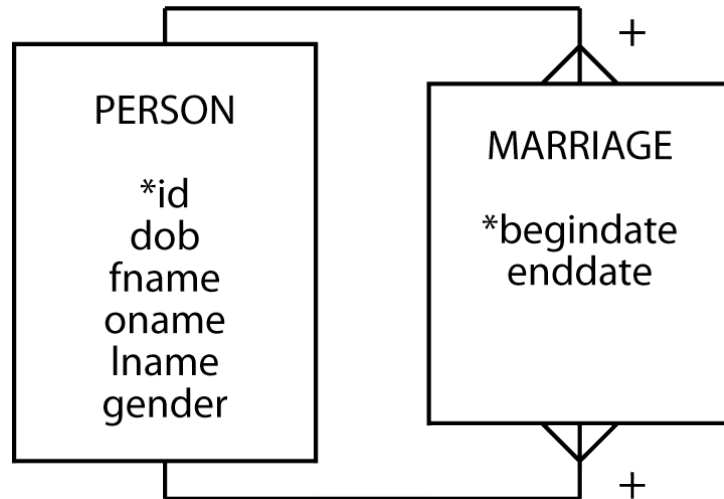


Now for a bit of marriage counseling. Marriage normally is a relationship between two people. Is that so? Well, some societies permit polygyny (one man can have many wives), others allow polyandry (one



woman can have many husbands), and some permit same-sex marriages. So marriage, if we consider all the possibilities, is an m:m relationship between people, and partner might be a better choice than spouse for labeling the relationship. Also, a person can be married more than once. To distinguish between different marriages, we really need to record the start and end date of each relationship and who was involved.

*A marriage data model*

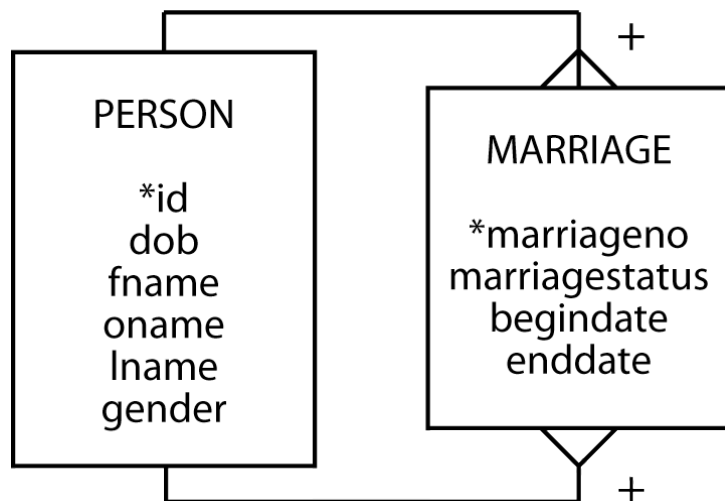


Marriage is an m:m relationship between two persons. It has attributes *begindate* and *enddate*. An instance of marriage is uniquely identified by a composite identifier: the two spouse identifiers and *begindate*. This means any marriage is uniquely identified by the composite of two person identifiers and the beginning date of the marriage. We need *begindate* as part of the identifier because the same couple might have more than one marriage (e.g., get divorced and remarry each other later). Furthermore, we can safely assume that it is impossible for a couple to get married, divorced, and remarried all on the one day. *Begindate* and *enddate* can be used to determine the current state of a marriage. If *enddate* is null, the marriage is current; otherwise, the couple has divorced.

This data model assumes a couple goes through some formal process to get married or divorced, and there is an official date for both of these events. What happens if they just gradually drift into cohabitation, and there is no official beginning date? Think about it. (The data model problem, that is—not cohabitation!) Many countries recognize this situation as a common-law marriage, so the data model needs to recognize it. The present data model cannot handle this situation because *begindate* cannot be null—it is an identifier. Instead, a new identifier is needed, and *begindate* should become an attribute.

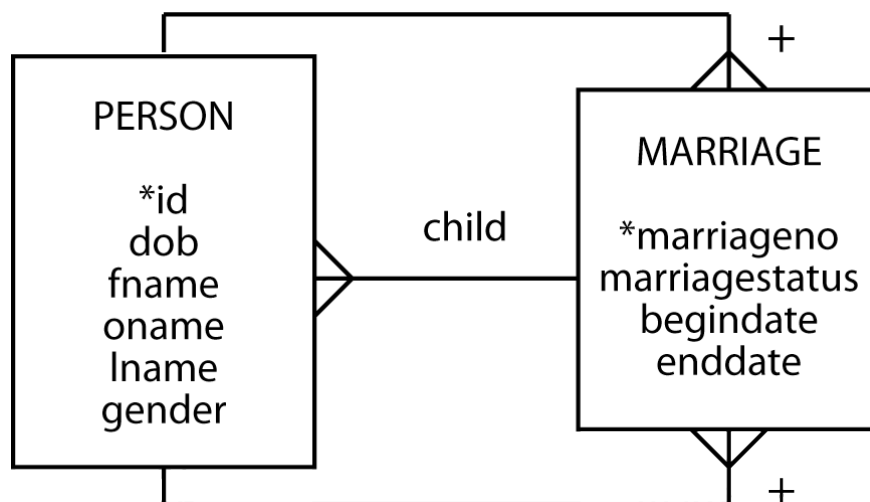
Two new attributes can handle a common-law marriage. *Marriageno* can count the number of times a couple has been married to each other. In the majority of cases, *marriageno* will be 1. *Marriagestatus* can record whether a marriage is current or ended. Now we have a data model that can also handle common-law marriages. This is also a high-quality data model in that the client does not have to remember to examine *enddate* to determine a marriage's current status. It is easier to remember to examine *marriagestatus* to check status. Also, we can allow a couple to be married, divorced, and remarried as many times as they like on the one day—which means we can now use the database in Las Vegas.

*A revised marriage data model*



All right, now that we have the couple successfully married, we need to start thinking about children. A marriage has zero or more children, and let's start with the assumption a child belongs to only one marriage. Therefore, we have a 1:m relationship between marriage and person to represent the children of a marriage.

*A marriage with children data model*



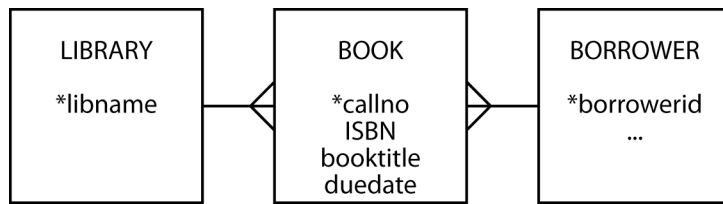
You might want to consider how the model would change to handle single-parent families, adopted children, and other aspects of human relationships.

**Skill builder** The International Commission for Border Resolution Disputes requires a database to record details of which countries have common borders. Design the database. Incidentally, which country borders the most other countries?

### When's a book not a book?

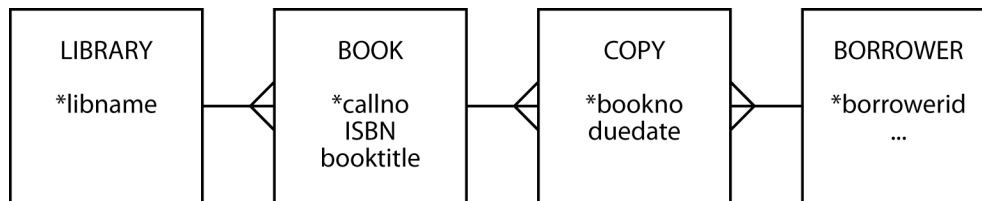
Sometimes we have to rethink our ideas of physical objects. Consider a data model for a library.

*A library data model fragment*



The preceding fragment assumes that a person borrows a book. What happens if the library has two copies of the book? Do we add an attribute to BOOK called *copy number*? No, because we would introduce redundancy by repeating the same information for each book. What you need to recognize is that in the realm of data modeling, a book is not really a physical thing, but the copy is, because it is what you borrow.

*A revised library data model fragment*



As you can see, a book has many copies, and a copy is of one book. A book can be identified by *callno*, which is usually a Library of Congress number or Dewey number. A copy is identified by a *bookno*. This is a unique number allocated by the library to the copy of the book. If you look in a library book, you will generally find it pasted on the inside back cover and shown in numeric and bar code format. Notice that it is called *book number* despite the fact that it really identifies a copy of a book. This is because most people, including librarians, think of the copy as a book.

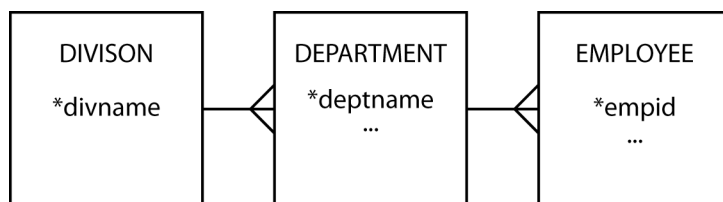
The International Standard Book Number (ISBN) uniquely identifies any instance of a book (not copy). Although it sounds like a potential identifier for BOOK, it is not. ISBNs were introduced in the second half of the twentieth century, and books published before then do not have an ISBN.

## A history lesson

Many organizations maintain historical data (e.g., a person's job history or a student's enrollment record). A data model can depict historical data relationships just as readily as current data relationships.

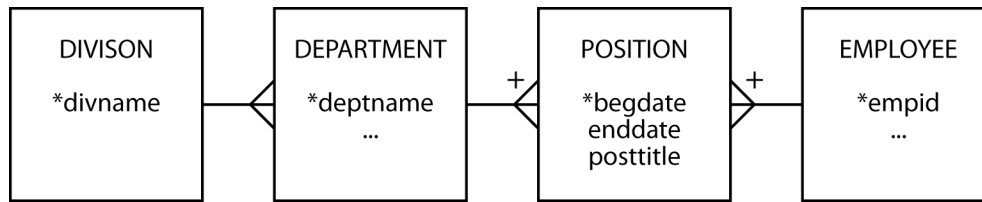
Consider the case of an employee who works for a firm that consists of divisions (e.g., production) and departments (e.g., quality control). The firm contains many departments, but a department belongs to only one division. At any one time, an employee belongs to only one department.

*Employment history—take 1*



The fragment in the figure can be amended to keep track of the divisions and departments in which a person works. While employed with a firm, a person can work in more than one department. Since a department can have many employees, we have an m:m relationship between DEPARTMENT and EMPLOYEE. We might call the resulting associative entity POSITION.

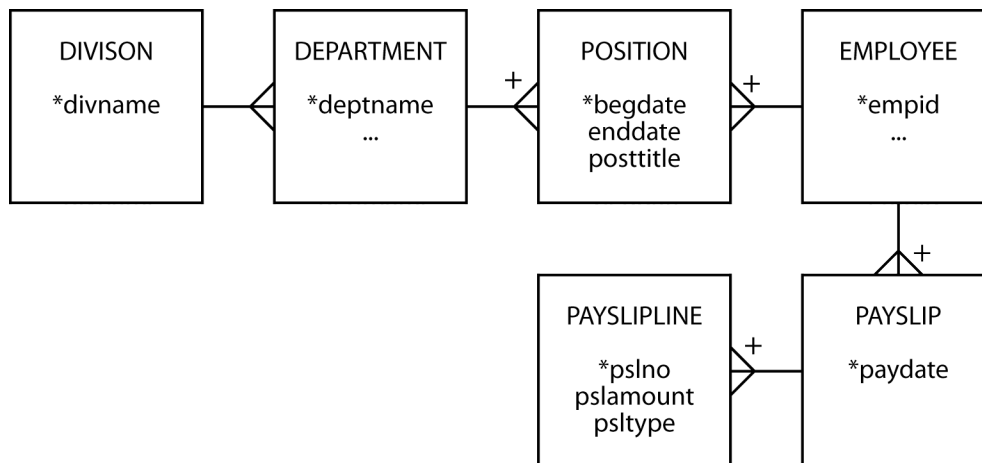
*Employment history—take 2*



The revised fragment records employee work history. Note that any instance of POSITION is identified by *begdate* and the identifiers for EMPLOYEE and DEPARTMENT. The fragment is typical of what happens when you move from just keeping current data to recording history. A 1:m becomes an m:m relationship to record history.

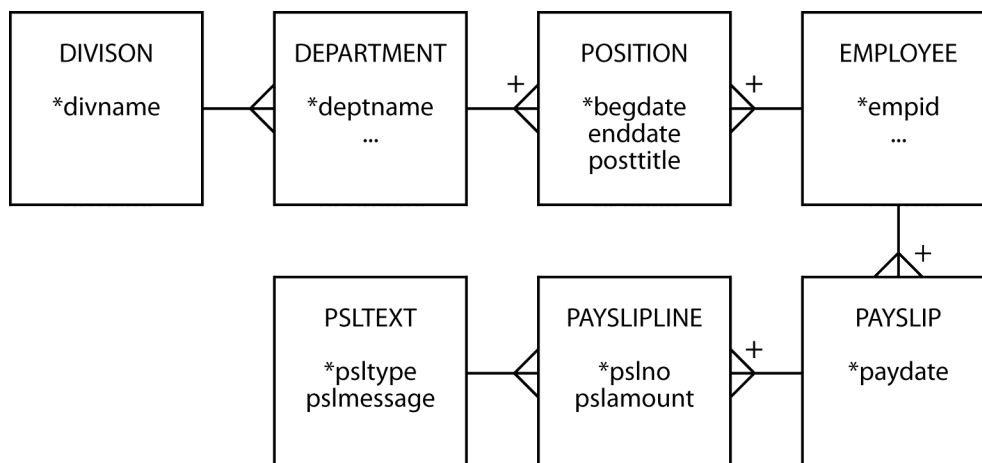
People who work get paid. How do we keep track of an employee's pay data? An employee has many pay slips, but a pay slip belongs to one employee. When you look at a pay slip, you will find it contains many items: gross pay and a series of deductions for tax, medical insurance, and so on. Think of a pay slip as containing many lines, analogous to a sales form. Now look at the revised data model.

*Employment history—take 3*



Typically, an amount shown on a pay slip is identified by a short text field (e.g., gross pay). PAYSLIPLINE contains an attribute *psltype* to identify the text that should accompany an amount, but where is the text? When dealing with codes like *psltype* and their associated text, the fidelity of a data model is improved by creating a separate entity, PSLTEXT, for the code and its text.

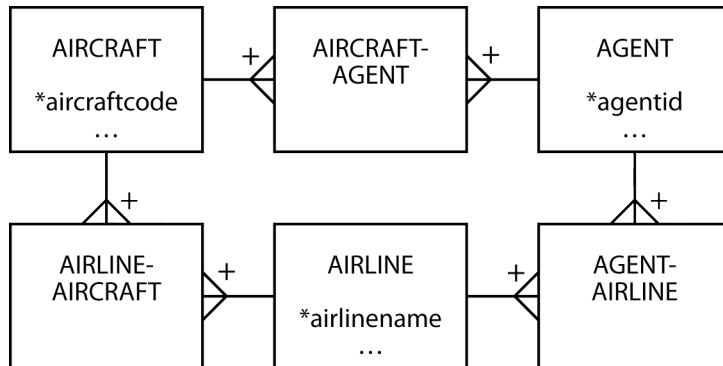
*Employment history—take 4*



## A ménage à trois for entities

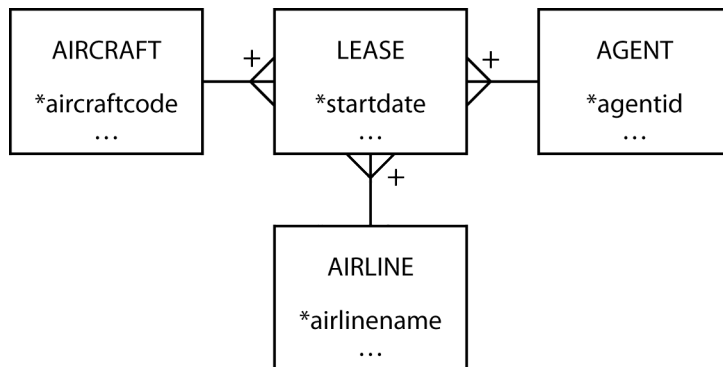
Consider aircraft leasing. A plane is leased to an airline for a specific period. When a lease expires, the plane can be leased to another airline. So, an aircraft can be leased many times, and an airline can lease many aircraft. Furthermore, there is an agent responsible for handling each deal. An agent can lease many aircraft and deal with many airlines. Over time, an airline will deal with many agents. When an agent reaches a deal with an airline to lease a particular aircraft, you have a transaction. If you analyze the aircraft leasing business, you discover there are three m:m relationships. You might think of these as three separate relationships.

*An aircraft-airline-agent data model*



The problem with three separate m:m relationships is that it is unclear where to store data about the lease. Is it stored in AIRCRAFT-AIRLINE? If you store the information there, what do you do about recording the agent who closed the deal? After you read the fine print and do some more thinking, you discover that a lease is the association of these three entities in an m:m relationship.

*A revised aircraft-airline-agent data model*



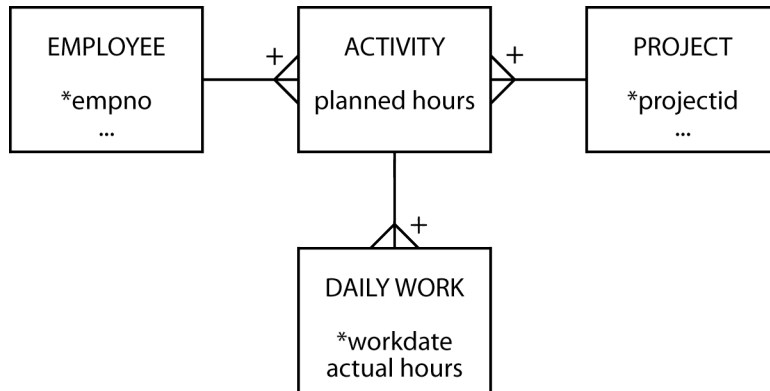
**Skill builder** Horse racing is a popular sport in some parts of the world. A horse competes in at most one race on a course at a particular date. Over time, a horse can compete in many races on many courses. A horse's rider is called a jockey, and a jockey can ride many horses and a horse can have many jockeys. Of course, there is only ever one jockey riding a horse at a particular time. Courses vary in their features, such as the length of the course and the type of surface (e.g., dirt or grass). Design a database to keep track of the results of horse races.

## Project management—planning and doing

Project management involves both planned and actual data. A project is divided into a number of activities that use resources. Planning includes estimation of the resources to be consumed. When a project is being

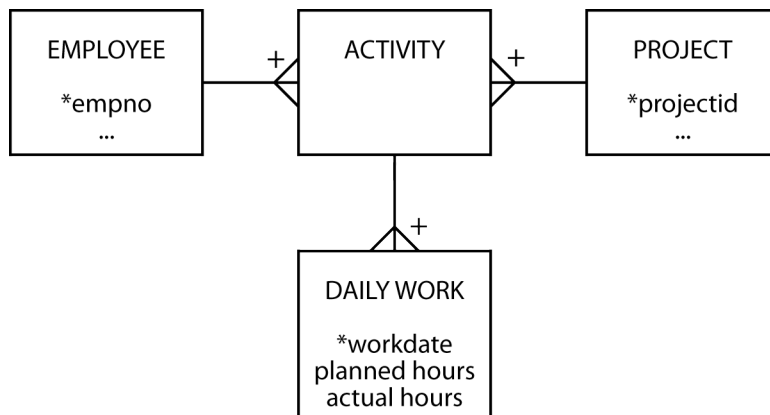
executed, managers keep track of the resources used to monitor progress and keep the project on budget. Planning data may not be as detailed as actual data and is typically fairly broad, such as an estimate of the number of hours that an activity will take. Actual data will be more detailed because they are usually collected by getting those assigned to the project to log a daily account of how they spent their time. Also, resources used on a project are allocated to a particular activity. For the purposes of this data model, we will focus only on recording time and nothing else.

*A project management data model*



Notice that *planned hours* is an attribute of ACTIVITY, and *actual hours* is an attribute of DAILY WORK. The hours spent on an activity are derived by summing *actual hours* in the associated DAILY WORK entity. This is a high-fidelity data model if planning is done at the activity level and employees submit daily worksheets. Planning can be done in greater detail, however, if planners indicate how many hours of each day each employee should spend on a project.

*A revised project management data model*



Now you see that *planned hours* and *actual hours* are both attributes of DAILY WORK. The message of this example, therefore, is not to assume that planned and actual data have the same level of detail, but do not be surprised if they do.

## Cardinality

Some data modeling languages are quite precise in specifying the cardinality, or multiplicity, of a relationship. Thus, in a 1:m relationship, you might see additional information on the diagram. For example, the “many” end of the relationship might contain notation (such as 0,n) to indicate zero or more instances of the entity at the “many” end of the relationship.

*Cardinality and modality options*

C ardin ality	Mo dal ity	Meaning
0,1	Op tio nal	There can be zero or one instance of the entity relative to the other entity.
0,n		There can be zero or many instances of the entity relative to the other entity.
1,1	Man dat ory	There is exactly one instance of the entity relative to the other entity.
1,n		The entity must have at least one and can have many instances relative to the other entity.

The data modeling method of this text, as you now realize, has taken a broad approach to cardinality (is it 1:1 or 1:m?). We have not been concerned with greater precision (is it 0,1 or 1,1?), because the focus has been on acquiring basic data modeling skills. Once you know how to model, you can add more detail. When learning to model, too much detail and too much terminology can get in the way of mastering this difficult skill. Also, it is far easier to sketch models on a whiteboard or paper when you have less to draw. Once you switch to a data modeling tool, then adding cardinality precision is easier and appropriate.

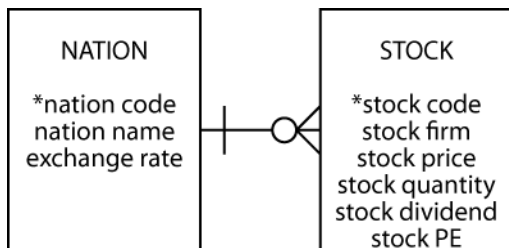
### Modality

Modality, also known as optionality, specifies whether an instance of an entity must participate in a relationship. Cardinality indicates the range of instances of an entity participating in a relationship, while modality defines the minimum number of instances. Cardinality and modality are linked, as shown in the table above. If an entity is optional, the minimum cardinality will be 0, and if mandatory, the minimum cardinality is 1.

By asking the question, “Does an occurrence of this entity require an occurrence of the other entity?” you can assess whether an entity is mandatory. The usual practice is to use “O” for optional and place it near the entity that is optional. Mandatory relationships are often indicated by a short line (or bar) at right angles to the relationship between the entities.

In the following figure, it is mandatory for an instance of STOCK to have an instance of NATION, and optional for an instance of NATION to have an instance of STOCK, which means we can record details of a nation for which stocks have not yet been purchased. During conversion of the data model to a relational database, the mandatory requirement (i.e., each stock must have a nation) is enforced by the foreign key constraint.

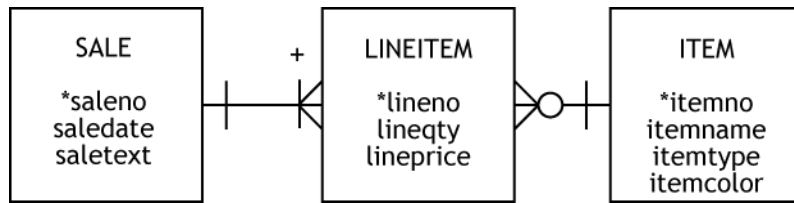
*1:m relationship showing modality*



Now consider an example of an m:m relationship. A SALE can have many LINEITEMs, and a LINEITEM has only one SALE. An instance of LINEITEM must be related to an instance of SALE, otherwise it can't exist. Similarly, it is mandatory for an instance of SALE to have an instance of LINEITEM, because it makes no sense to have a sale without any items being sold.

In a relational database, the foreign key constraint handles the mandatory relationship between instances of LINEITEM and SALE. The mandatory requirement between SALE and LINEITEM has to be built into the processing logic of the application that processes a sales transaction. Basic business logic requires that a sale include some items, so a sales transaction creates one instance of SALE and multiple instances of LINEITEM. It does not make sense to have a sale that does not sell something.

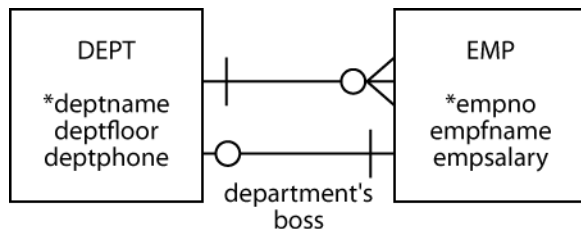
*An m:m relationship showing modality*



The relationship between ITEM and LINEITEM is similar in structure to that of NATION and STOCK, which we saw in the previous example. An instance of a LINEITEM has a mandatory requirement for an instance of ITEM. An instance of an ITEM can exist without the need for an instance of a LINEITEM (i.e., the items that have not been sold).

We gain further understanding of modality by considering a 1:1 relationship. We focus attention on the 1:1 because the 1:m relationship has been covered. The 1:1 implies that it is mandatory for an instance of DEPT to be related to one instance of EMP (i.e., a department must have a person who is the boss), and an instance of EMP is optionally related to one instance of DEPT (i.e., an employee can be a boss, but not all employees are departmental bosses).

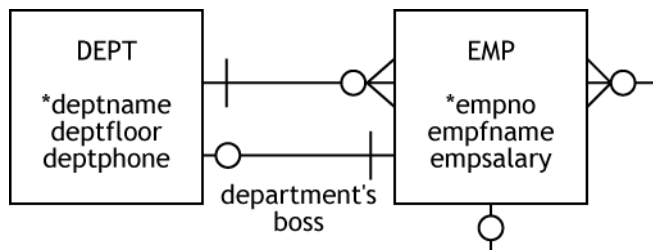
*A 1:1 relationship showing modality*



Earlier, when deciding where to store the foreign key for the 1:1 relationship, we settled on placing the foreign key in DEPT, because all departments have a boss. Now we have a rule. The foreign key goes with the entity for which the relationship is mandatory. When specifying the create statements, you will need to be aware of the chicken-and-egg connection between DEPT and EMP. We cannot insert an employee in EMP unless we know that person's department, and we cannot insert a department unless we have already inserted the boss for that department in EMP. We can get around this problem by using a **deferrable foreign key constraint**, but at this point we don't know enough about transaction processing to discuss this approach.

Let's now consider recursive relationships. The following figure shows a recursive 1:m between employees, representing that one employee can be the boss of many other employees and a person has one boss. It is optional that a person is a boss, and optional that everyone has a boss, mainly because there has to be one person who is the boss of everyone. However, we can get around this one exception by inserting employees in a particular order. By inserting the biggest boss first, we effectively make it mandatory that all employees have a boss. Nevertheless, data models cover every situation and exception, so we still show the relationship as optional.

*1:m recursive with modality*

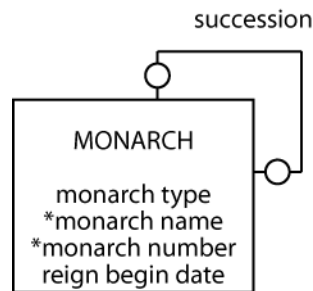


We rely again on the British monarchy. This time we use it to illustrate modality with a 1:1 recursive relationship. As the following figure shows, it is optional for a monarch to have a successor. We are not



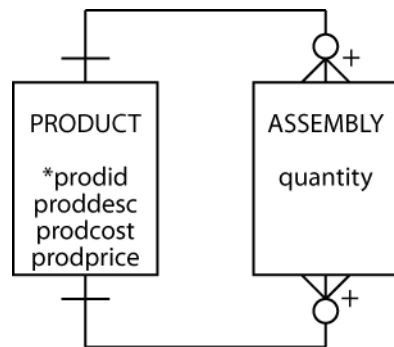
talking about the end of a monarchy, but rather how we address the data modeling issue of not knowing who succeeds the current monarch. Similarly, there must be a monarch who did not succeed someone (i.e., the first king or queen). Thus, both ends of the succession relationship have optional modality.

*1:1 recursive with modality*



Finally, in our exploration of modality, we examine the m:m recursive. The model indicates that it is optional for a product to have components and optional for a product to be a component in other products. However, every assembly must have associated products.

*m:m recursive with modality*



**Summary** Modality adds additional information to a data model, and once you have grasped the fundamental ideas of entities and relationships, it is quite straightforward to consider whether a relationship is optional or mandatory. When a relationship is mandatory, you need to consider what constraint you can add to a table's definition to enforce the relationship. As we have seen, in some cases the foreign key constraint handles mandatory relationships. In other cases, the logic for enforcing a requirement will be built into the application.

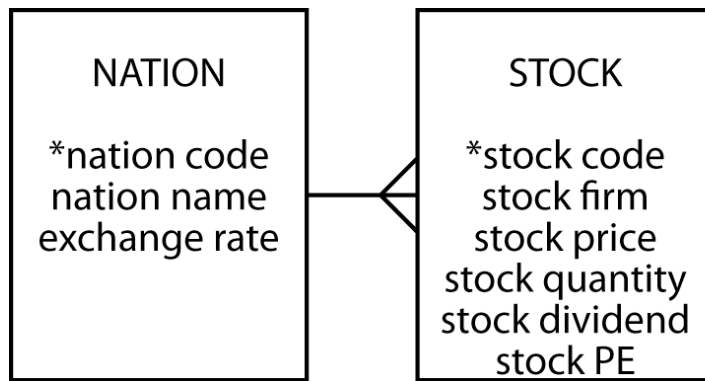
## Entity types

A data model contains different kinds of entities, which are distinguished by the format of their identifiers. Labeling each of these entities by type will help you to determine what questions to ask.

**Independent entity** An independent entity is often central to a data model and foremost in the client's mind. Independent entities are frequently the starting points of a data model. Remember that in the investment database, the independent entities are **STOCK** and **NATION**. Independent entities typically have clearly distinguishable names because they occur so frequently in the client's world. In addition, they usually have a single identifier, such as *stock code* or *nation code*.

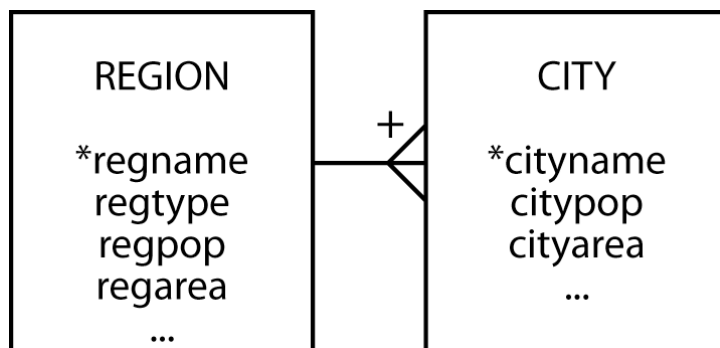
Independent entities are often connected to other independent entities in a 1:m or m:m relationship. The data model in the following figure shows two independent entities linked in a 1:m relationship.

*NATION and STOCK—independent entities*



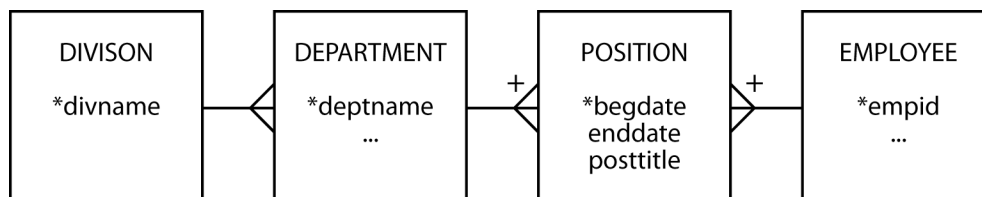
**Weak or dependent entity** A weak entity (also known as a dependent entity) relies on another entity for its existence and identification. It is recognized by a plus on the weak entity's end of the relationship. CITY cannot exist without REGION. A CITY is uniquely identified by *cityname* and *regname*. If the composite identifier becomes unwieldy, creating an arbitrary identifier (e.g., *cityno*) will change the weak entity into an independent one.

*CITY—a weak entity*



**Associative entity** Associative entities are by-products of m:m relationships. They are typically found between independent entities. Associative entities sometimes have obvious names because they occur in the real world. For instance, the associative entity for the m:m relationship between DEPARTMENT and EMPLOYEE is usually called POSITION. If the associative entity does not have a common name, the two entity names are generally hyphenated (e.g., DEPARTMENT-EMPLOYEE). Always search for the appropriate name, because it will improve the quality of the data model. Hyphenated names are a last resort.

*POSITION is an associative entity*



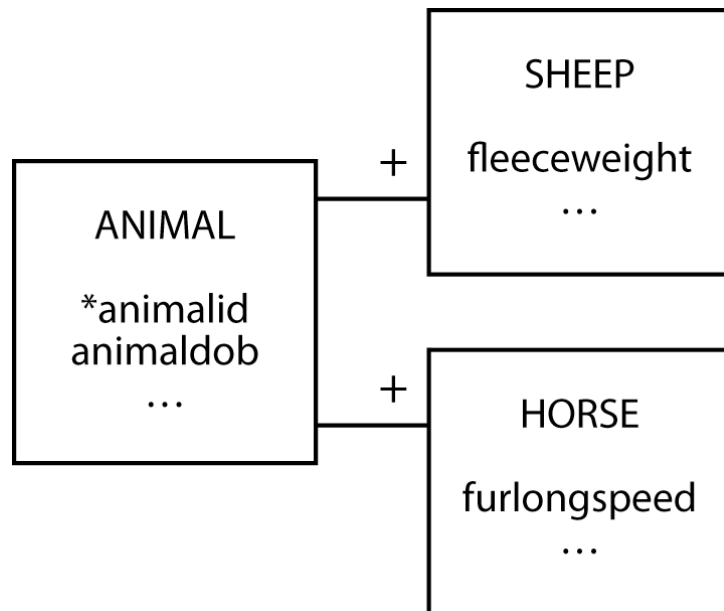
Associative entities can show either the current or the historic relationship between two entities. If an associative entity's only identifiers are the two related entities' identifiers, then it records the current relationship between the entities. If the associative entity has some time measure (e.g., date or hour) as a partial identifier, then it records the history of the relationship. Whenever you find an associative entity, ask the client whether the history of the relationship should be recorded.

Creating a single, arbitrary identifier for an associative entity will change it to an independent one. This is likely to happen if the associative entity becomes central to an application. For example, if a personnel department does a lot of work with POSITION, it may find it more expedient to give it a separate identifier (e.g., *position number*).

**Aggregate entity** An aggregate entity is created when several different entities have similar attributes that are distinguished by a preceding or following modifier to keep their names unique. For example, because components of an address might occur in several entities (e.g., CUSTOMER and SUPPLIER), an aggregate address entity can be created to store details of all addresses. Aggregate entities usually become independent entities. In this case, we could use *address number* to identify an instance of address uniquely.

**Subordinate entity** A subordinate entity stores data about an entity that can vary among instances. A subordinate entity is useful when an entity consists of mutually exclusive classes that have different descriptions. The farm animal database shown in the following figure indicates that the farmer requires different data for sheep and horses. Notice that each of the subordinate entities is identified by the related entity's identifier. You could avoid subordinate entities by placing all the attributes in ANIMAL, but then you make it incumbent on the client to remember which attributes apply to horses and which to sheep. This becomes an important issue for null fields. For example, is the attribute *hay consumption* null because it does not apply to sheep, or is it null because the value is unknown?

*SHEEP and HORSE are subordinate entities*



If a subordinate entity becomes important, it is likely to evolve into an independent entity. The framing of a problem often determines whether subordinate entities are created. Stating the problem as, “a farmer has many animals, and these animals can be horses or sheep,” might lead to the creation of subordinate entities. Alternatively, saying that “a farmer has many sheep and horses” might lead to setting up SHEEP and HORSE as independent entities.

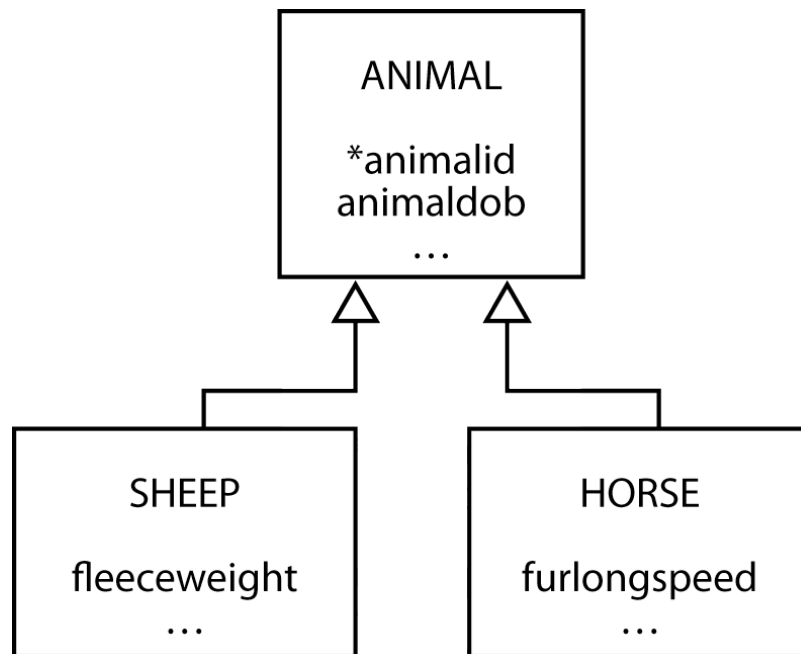
## Generalization and aggregation

Generalization and aggregation are common ideas in many modeling languages, particularly object-oriented modeling languages such as the Unified Modeling Language (UML), which we will use in this section.

**Generalization** A generalization is a relationship between a more general element and a more specific element. In the following figure, the general element is *animal*, and the specific elements are *sheep* and *horse*. A horse is a subtype of animal. A generalization is often called an “is a” relationship because the subtype element is a member of the generalization.

A generalization is directly mapped to the relational model with one table for each entity. For each of the subtype entities (i.e., SHEEP and HORSE), the primary key is that of the supertype entity (i.e., ANIMAL). You must also make this column a foreign key so that a subtype cannot be inserted without the presence of the matching supertype. A generalization is represented by a series of 1:1 relationships to weak entities.

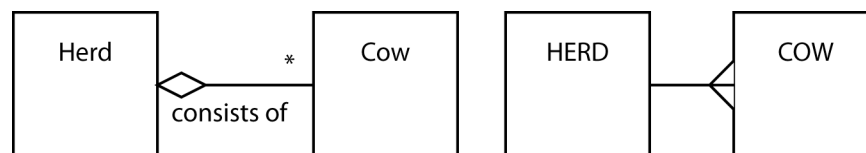
*A generalization hierarchy in UML*



## Aggregation

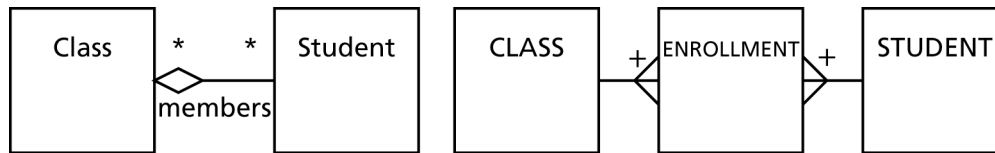
Aggregation is a part-whole relationship between two entities. Common phrases used to describe an aggregation are “consists of,” “contains,” and “is part of.” The following figure shows an aggregation where a herd consists of many cows. Cows can be removed or added, but it is still a herd. An open diamond next to the whole denotes an aggregation. In data modeling terms, there is a 1:m relationship between herd and cow.

*An aggregation in UML and data modeling*



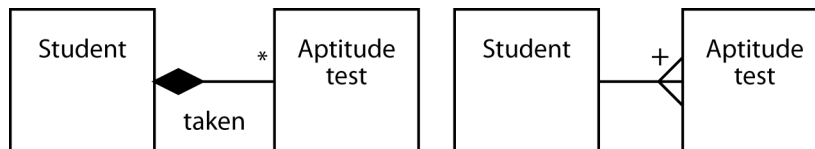
There are two special types of aggregation: shared and composition. With **shared aggregation**, one entity owns another entity, but other entities can own that entity as well. A sample shared aggregation is displayed in the following figure, which shows that a class has many students, and a student can be a member of many classes. Students, the parts in this case, can be part of many classes. An open diamond next to the whole denotes a shared aggregation, which we represent in data modeling as an m:m relationship.

*Shared aggregation in UML and data modeling*



In a **composition aggregation**, one entity exclusively owns the other entity. A solid diamond at the whole end of the relationship denotes composition aggregation. The appropriate data model is a weak entity, as shown in the following figure.

*Composition aggregation in UML and data modeling*



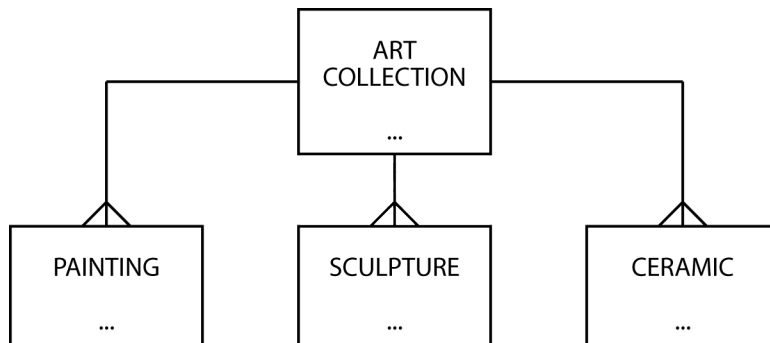
## Data modeling hints

A high-fidelity data model takes time to develop. The client must explain the problem fully so that the database designer can understand the client's requirements and translate them into a data model. Data modeling is like prototyping; the database designer gradually creates a model of the database as a result of interaction with the client. Some issues will frequently arise in this progressive creation, and the following hints should help you resolve many of the common problems you will encounter.

### The rise and fall of a data model

Expect your data model to both expand and contract. Your initial data model will expand as you extend the boundaries of the application. You will discover some attributes will evolve into entities, and some 1:m relationships will become m:m relationships. It will grow because you will add entities, attributes, and relationships to allow for exceptions. Your data model will grow because you are representing the complexity of the real world. Do not try to constrain growth. Let the data model be as large as is necessary. As a rough rule, expect your final data model to grow to about two to three times the number of entities of your initial data model.

Expect your data model to contract as you generalize structures. The following fragment is typical of the models produced by novice data modelers. It can be reduced by recognizing that PAINTING, SCULPTURE, and CERAMIC are all types of art. A more general entity, ART, could be used to represent the same data. Of course, it would need an additional attribute to distinguish the different types of art recordings.



*The shrunken data model*



As you discover new entities, your data model will grow. As you generalize, your data model will shrink.

## Identifier

If there is no obvious simple identifier, invent one. The simplest is a meaningless code (e.g., *person number* or *order number*). If you create an identifier, you can also guarantee its uniqueness.

Don't overwork an identifier. The worst case we have seen is a 22-character product code used by a plastics company that supposedly not only uniquely identified an item but told you its color, its manufacturing process, and type of plastic used to make it! Color, manufacturing process, and type of plastic are all attributes. This product code was unwieldy. Data entry error rates were extremely high, and very few people could remember how to decipher the code.

An identifier has to do only one thing: uniquely identify every instance of the entity. Sometimes trying to make it do double, or even triple, duty only creates more work for the client.

## Position and order

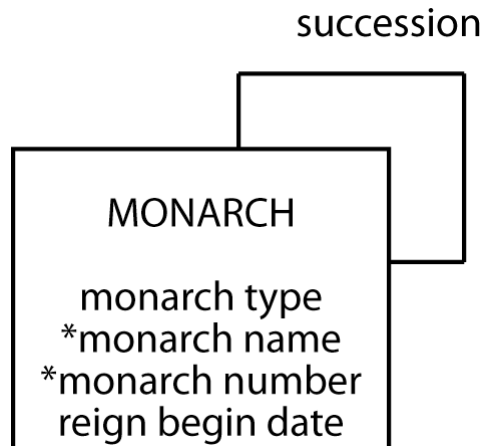
There is no ordering in a data model. Entities can appear anywhere. You will usually find a central entity near the middle of the data model only because you tend to start with prominent entities (e.g., starting with STUDENT when modeling a student information system). What really matters is that you identify all relevant entities.

Attributes are in no order. You find that you will tend to list them as they are identified. For readability, it is a good idea to list some attributes sequentially. For example, first name, other names, and last name are usually together. This is not necessary, but it speeds up verification of a data model's completeness.

Instances are also assumed to have no ordering. There is no first instance, next instance, or last instance. If you must recognize a particular order, create an attribute to record the order. For example, if ranking of potential investment projects must be recorded, include an attribute (*projrank*) to store this data. This does not mean the instances will be stored in *projrank* order, but it does allow you to use the ORDER BY clause of SQL to report the projects in rank order.

If you need to store details of a precedence relationship (i.e., there is an ordering of instances and a need to know the successor and predecessor of any instance), then use a 1:1 recursive relationship.

*Using an attribute to record an ordering of instances*



### Attributes and consistency

Attributes must be consistent, retaining the same meaning for every instance. An example of an inconsistent attribute would be an attribute *stock info* that contains a stock's PE ratio or its ROI (return on investment). Another attribute, say *stock info code*, is required to decipher which meaning applies to any particular instance. The code could be "1" for PE and "2" for ROI. If one attribute determines the meaning of another, you have inconsistency. Writing SQL queries will be extremely challenging because inconsistent data increases query complexity. It is better to create separate attributes for PE and ROI.

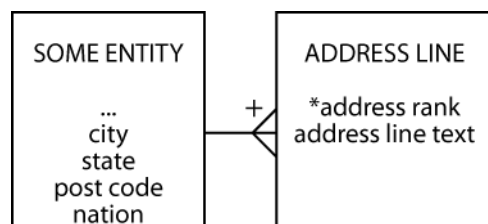
### Names and addresses

An attribute is the smallest piece of data that will conceivably form part of a query. If an attribute has segments (e.g., a person's name), determine whether these could form part of a query. If so, then make them separate attributes and reapply the query test. When you apply the query test to *person name*, you will usually decide to create three attributes: *first name*, *other names*, and *last name*.

What about titles? There are two sorts of modifier titles: preceding titles (e.g., Mr., Mrs., Ms., and Dr.) and following titles (e.g., Jr. and III). These should be separate attributes, especially if you have divided *name* into separate attributes.

Addresses seem to cause more concern than names because they are more variant. Business addresses tend to be the most complicated, and foreign addresses add a few more twists. The data model fragment in the following figure works in most cases.

#### Handling addresses



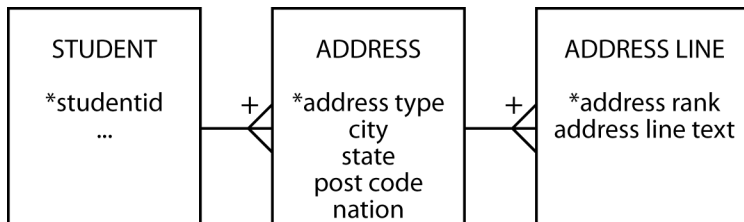
Common features of every address are city, state (province in Canada, county in Eire), postal code (ZIP code in the United States), and nation. Some of these can be null. For example, Singapore does not have any units corresponding to states. In the United States, city and state can be derived from the zip code, but this is not necessarily true for other countries. Furthermore, even if this were true for every nation, you would need a different derivation rule for each country, which is a complexity you might want to avoid. If

there is a lifetime guarantee that every address in the database will be for one country, then examine the possibility of reducing redundancy by just storing post code.

Notice that the problem of multiple address lines is represented by a 1:m relationship. An address line is a text string that appears as one line of an address. There is often a set sequence in which these are displayed. The attribute *address rank* records this order.

How do you address students? First names may be all right for class, but it is not adequate enough for student records. Students typically have multiple addresses: a home address, a school address, and maybe a summer address. The following data model fragment shows how to handle this situation. Any instance of ADDRESS LINE is identified by the composite of *studentid*, *addresstype*, and *address rank*.

*Students with multiple addresses*



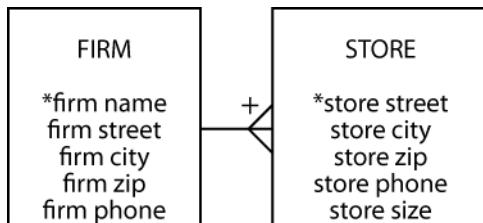
The identifier *address type* is used to distinguish among the different types of addresses. The same data model fragment works in situations where a business has different mailing and shipping addresses.

When data modeling, can you take a shortcut with names and addresses? It is time consuming to write out all the components of name and address. It creates clutter and does not add much fidelity. Our practical advice is to do two things. *First*, create a policy for all names and addresses (e.g., all names will be stored as three parts: first name, other names, last name). *Second*, use shorthand forms of attributes for names and addresses when they obey the policy. So from now on, we will sometimes use name and address as attributes with the understanding that when the database is created, the parts will become separate columns.

### Single-instance entities

Do not be afraid of creating an entity with a single instance. Consider the data model fragment in the following figure which describes a single fast-food chain. Because the data model describes only one firm, the entity FIRM will have only one instance. The inclusion of this single instance entity permits facts about the firm to be maintained. Furthermore, it provides flexibility for expansion. If two fast-food chains combine, the data model needs no amendment.

*FIRM—a single-instance entity*



### Picking words

Words are very important. They are all we have to make ourselves understood. Let the client choose the words, because the data model belongs to the client and describes the client's world. If you try to impose your words on the client, you are likely to create a misunderstanding.



## Synonyms

Synonyms are words that have the same meaning. For example, task, assignment, and project might all refer to the same real-world object. One of these terms, maybe the one in most common use, needs to be selected for naming the entity. Clients need to be told the official name for the entity and encouraged to adopt this term for describing it. Alternatively, create views that enable clients to use their own term. Synonyms are not a technical problem. It is a social problem of getting clients to agree on one word.

## Homonyms

Homonyms are words that sound the same but have different meanings. Homonyms can create real confusion. *Sale date* is a classic example. In business it can have several meanings. To the salespeople, it might be the day that the customer shakes hands and says, “We have a deal.” For the lawyers, it could be the day when the contract is signed, and for production, it might be the day the customer takes delivery. In this case, the solution is reasonably obvious; redefine *sale date* to be separate terms for each area (e.g., *sales date*, *contract date*, and *delivery date* for sales, legal, and production departments, respectively).

Homonyms cause real confusion when clients do not realize that they are using the same word for different entities or attributes. Hunt down homonyms by asking lots of questions and querying a range of clients.

## Exception hunting

Go hunting for exceptions. Keep asking the client questions such as

- Is it always like this?
- Would there be any situations where this could be an m:m relationship?
- Have there ever been any exceptions?
- Are things likely to change in the future?

Always probe for exceptions and look for them in the examples the client uses. Redesigning the data model to handle exceptions increases fidelity.

## Relationship labeling

Relationship labeling clutters a data model. In most cases, labels can be correctly inferred, so use labels only when there is a possibility of ambiguity.

## Keeping the data model in shape

As you add to a data model, maintain fidelity. Do not add entities without also completing details of the identifier and attributes. By keeping the data model well formed, you avoid ambiguity, which frequently leads to miscommunication.

## Used entities

Would you buy a used data model? Certainly, because a used data model is more likely to have higher fidelity than a brand new one. A used data model has been subjected to much scrutiny and revision and should be a more accurate representation of the real world.

## Meaningful identifiers

An identifier is said to be meaningful when some attributes of the entity can be inferred from the identifier's value (e.g., a code of B78 for an item informs a clerk that the item is black). The word “meaningful” in everyday speech usually connotes something desirable, but many data managers agree that meaningful identifiers or codes create problems. While avoiding meaningful identifiers is generally accepted as good data management practice, they are still widely used. After a short description of some examples of meaningful identifiers, this section discusses the pros and cons of meaningful identifiers.

The invoice number “12dec0001” is a meaningful identifier. The first five positions indicate in which period the invoice was generated. The last four digits of the invoice number have a meaning. The 0001 indicates it was the first invoice sent in December. In addition, e-mail addresses, like [mvdpas@bizzo.nl](mailto:mvdpas@bizzo.nl), have a meaning.

When coding a person's gender, some systems use the codes “m” and “f” and others “1” and “2.” The pair (m, f) is meaningful, but (1, 2) is not and puts the onus on the user to remember the meaning. The codes “m” and “f” are derived from the reality that they represent. They are the first letters of the words male and female. The “1” and “2” are not abstracted from reality, and therefore they do not have an intuitive meaning.

However, if you want to set an international standard suitable for many languages, then you need to revert to numeric codes, because “m” and “f” are not the first letters in all languages for male and female, respectively. Thus the English version of the International Organization for Standardization (ISO) information interchange standard for the representation of human sexes (ISO 5218) states:

*No significance is to be placed on the fact that “Male” is coded “1” and “Female” is coded “2.” This standard was developed based upon predominant practices of the countries involved and does not convey any meaning of importance, ranking or any other basis that could imply discrimination.*

In determining whether an identifier is meaningful, one can look at the way new values of that identifier are generated. If the creation takes place on the basis of characteristics that appear in reality, the identifier is meaningful. That is why the identifier of the invoice “12dec0002,” which is the identifier assigned to the second invoice generated in December 2012, is meaningful. It is also why the e-mail address [rwatson@terry.uga.edu](mailto:rwatson@terry.uga.edu) is meaningful.

Meaningful identifiers clearly have some advantages and disadvantages, and these are now considered.

### *Advantages and disadvantages of meaningful identifiers*

Advantages	Disadvantages
Recognizable and rememberable	Identifier exhaustion
Administrative simplicity	Reality changes
	Loss of meaningfulness

### Advantages of meaningful identifiers

**Recognizable and rememberable** People can often recognize and remember the significance of meaningful identifiers. If, for example, someone receives an e-mail from [rwatson@terry.uga.edu](mailto:rwatson@terry.uga.edu), this person can probably deduce the sender's name and workplace. If a manufacturer, for instance, uses the last two digits of its furniture code to indicate the color (e.g., 08 = beech), employees can then quickly determine the color of a packed piece of furniture. Of course, the firm could also show the color of the furniture on the label, in which case customers, who are most unlikely to know the coding scheme, can quickly determine the color of the enclosed product.

**Administrative simplicity** By using meaningful identifiers, administration can be relatively easily decentralized. Administering e-mail addresses, for instance, can be handled at the domain level. This way, one can avoid the process of synchronizing with other domains whenever a new e-mail address is created.

A second example is the EAN (European article number), the bar code on European products. Issuing EANs can be administered per country, since each country has a unique number in the EAN. When issuing a new number, a country need only issue a code that is not yet in use in that country. The method of contacting all countries to ask whether an EAN has already been issued is time consuming and open to error. On the other hand, creating a central database of issued EAN codes is an option. If every issuer of EAN codes can access such a database, alignment among countries is created and duplicates are prevented.

### **Disadvantages of meaningful identifiers**

**Identifier exhaustion** Suppose a company with a six-digit item identifier decides to use the first three digits to specify the product group (e.g., stationery) uniquely. Within this group, the remaining three digits are used to define the item (e.g., yellow lined pad). As a consequence, only one thousand items can be specified per product group. This problem remains even when some numbers in a specific product group, or even when entire product groups, are not used.

Consider the case when product group “010” is exhausted and is supplemented by the still unused product group code “940.” What seems to be a simple quick fix causes problems, because a particular product group is not uniquely identified by a single identifier. Clients have to remember there is an exception.

A real-world example of identifier exhaustion is a holding company that identifies its subsidiaries by using a code, where the first character is the first letter of the subsidiary’s name. When the holding company acquired its seventh subsidiary, the meaningfulness of the coding system failed. The newly acquired subsidiary name started with the same letter as one of the existing subsidiaries. The system had already run out of identifiers.

When existing identifiers have to be converted because of exhaustion, printed codes on labels, packages, shelves, and catalogs will have to be redone. Recoding is an expensive process and should be avoided.

Exhaustion is not only a problem with meaningful identifiers. It can happen with non-meaningful identifiers. The problem is that meaningful identifier systems tend to exhaust sooner. To avoid identifier exhaustion and maintain meaningful identifiers, some designers increase identifier lengths (e.g., four digits for product group).

**Reality changes** The second problem of meaningful identifiers is that the reality they record changes. For example, a company changes its name. The College of Business at the University of Georgia was renamed the Terry College of Business. E-mail addresses changed (e.g., [rwatson@cba.uga.edu](mailto:rwatson@cba.uga.edu) became [rwatson@terry.uga.edu](mailto:rwatson@terry.uga.edu)). An identifier can remain meaningful over a long period only if the meaningful part rarely or never changes.

Non-meaningful identifiers avoid reality changes. Consider the case of most telephone billing systems, which use an account ID to identify a customer rather than a telephone number. This means a customer can change telephone numbers without causing any coding problems because telephone number is not the unique identifier of each customer.

**A meaningful identifier can lose its meaningfulness** Consider the problems that can occur with the identifier that records both the color and the material of an article. Chipboard with a beech wood veneer is coded BW. A black product, on the other hand, is coded BL. Problems arise whenever there is an overlap between these characteristics. What happens with a product that is finished with a black veneer on beech wood? Should the code be BW or BL? What about a black-painted product that is made out of beech wood? Maybe a new code, BB for black beech wood, is required. There will always be employees and customers who do not know the meaning of these not-so-meaningful identifiers and attribute the wrong meaning to them.

## The solution—non-meaningful identifiers

Most people are initially inclined to try to make identifiers meaningful. There is a sense that something is lost by using a simple numeric to identify a product or customer. Nothing, however, is lost and much is gained. Non-meaningful identifiers serve their sole purpose well—to identify an entity uniquely. Attributes are used to describe the characteristics of the entity (e.g., color, type of wood). A clear distinction between the role of identifiers and attributes creates fewer data management problems now and in the future.

### *Vehicle identification*

In most countries, every road vehicle is uniquely identified. The Vehicle Identification Number (VIN) was originally described in ISO Standard 3779 in February 1977 and last revised in 1983. It is designed to identify motor vehicles, trailers, motorcycles, and mopeds. A VIN is 17 characters, A through Z and 0 through 9.

The European Union and the United States have different implementations of the standard. The U.S. VIN is divided into four parts

- World Manufacturer's Identification (WMI) - three characters
- Vehicle Description Section (VDS) - five characters
- Check digit
- Vehicle Identification Section (VIS) - eight characters

When decoded, a VIN specifies the country and year of manufacture; make, model, and serial number; assembly plant; and even some equipment specifications.

VINs are normally located in several locations on a car, but the most common places are in the door frame of the front doors, on the engine itself, around the steering wheel, or on the dash near the window. What do you think of the VIN as a method of vehicle identification? How do you reconcile it with the recommendation to use meaningless identifiers? If you were consulted on a new VIN system, what would advise?

---

## The seven habits of highly effective data modelers

There is often a large gap between the performance of *average* and *expert* data modelers. An insight into the characteristics that make some data modelers more skillful than others should improve your data modeling capabilities.

### Immerse

Find out what the client wants by immersing yourself in the task environment. Spend some time following the client around and participate in daily business. Firsthand experience of the problem will give you a greater understanding of the client's requirements. Observe, ask questions, reflect, and talk to a wide variety of people (e.g., managers, operations personnel, customers, and suppliers). The more you learn about the problem, the better equipped you are to create a high-fidelity data model.

### Challenge

Challenge existing assumptions; dig out the exceptions. Test the boundaries of the data model. Try to think about the business problem from different perspectives (e.g., how might the industry leader tackle this problem?). Run a brainstorming session with the client to stimulate the search for breakthrough solutions.

## Generalize

Reduce the number of entities whenever possible by using generalized structures (remember the Art Collection model) to simplify the data model. Simpler data models are usually easier to understand and less costly to implement. Expert data modelers can see beyond surface differences to discern the underlying similarities of seemingly different entities.

## Test

Test the data model by reading it to yourself and several people intimately familiar with the problem. Test both directions of every relationship (e.g., a farmer has many cows and a cow belongs to only one farmer). Build a prototype so that the client can experiment and learn with a concrete model. Testing is very important because it costs very little to fix a data model but a great deal to repair a system based on an incorrect data model.

## Limit

Set reasonable limits to the time and scope of data modeling. Don't let the data modeling phase continue forever. Discover the boundaries early in the project and stick to these unless there are compelling reasons to extend the project. Too many projects are allowed to expand because it is easier to say *yes* than *no*. In the long run, however, you do the client a disservice by promising too much and extending the life of the project. Determine the core entities and attributes that will solve most of the problems, and confine the data model to this core. Keep sight of the time and budget constraints of the project.

## Integrate

Step back and reflect on how your project fits with the organization's information architecture. Integrate with existing systems where feasible and avoid duplication. How does your data model fit with the corporate data model and those of other projects? Can you use part of an existing data model? A skilled data modeler has to see both the fine-grained detail of a project data model and the big picture of the corporate data resource.

## Complete

Good data modelers don't leave data models ill-defined. All entities, attributes, and relationships are carefully defined, ambiguities are resolved, and exceptions are handled. Because the full value of a data model is realized only when the system is complete, the data modeler should stay involved with the project until the system is implemented. The data modeler, who generally gets involved in the project from its earliest days, can provide continuity through the various phases and ensure the system solves the problem.

## Summary

Data modeling is both a technique for modeling data and its relationships and a graphical representation of a database. It communicates a database's design. The goal is to identify the facts that must be stored in a database. Building a data model is a partnership between a client, a representative of the eventual owners of the database, and a designer. The building blocks are an entity, attribute, identifier, and relationship. A well-formed data model, which means the construction rules have been obeyed, clearly communicates information to the client. A high-fidelity data model faithfully describes the world it represents.

A data model is an evolving representation. Each change should be an incremental improvement in quality. The quality of a data model can be determined only by understanding the context in which it will be used. A

data model can model historical data relationships just as readily as current ones. Cardinality specifies the precise multiplicity of a relationship. Modality indicates whether a relationship is optional or mandatory. The five different types of entities are independent, dependent, associative, aggregate, and subordinate. Expect a data model to expand and contract. A data model has no ordering. Introduce an attribute if ordering is required. An attribute must have the same meaning for every instance. An attribute is the smallest piece of data that will conceivably form part of a query. Synonyms are words that have the same meaning; homonyms are words that sound the same but have different meanings. Identifiers should generally have no meaning. Highly effective data modelers immerse, challenge, generalize, test, limit, integrate, and complete.

## Key terms and concepts

---

Aggregate entity	Homonym
Aggregation	Identifier
Associative entity	Independent entity
Attribute	Instance
Cardinality	Mandatory
Composite aggregation	Modality
Data model	Modeling
Data model quality	Optional
Dependent entity	Relationship
Determinant	Relationship descriptor
Domain	Relationship label
Entity	Shared aggregation
Generalization	Synonym
High-fidelity image	

---

## References and additional readings

Carlis, J. V. 1991. *Logical data structures*. Minneapolis, MN: University of Minnesota.

Hammer, M. 1990. Reengineering work: Don't automate, obliterate. *Harvard Business Review* 68 (4):104–112.

## Exercises

### Short answers

1. What is data modeling?
2. What is a useful technique for identifying entities in a written description of a data modeling problem?
3. When do you label relationships?
4. When is a data model well formed, and when is it high-fidelity?
5. How do you handle exceptions when data modeling?
6. Describe the different types of entities.
7. Why might a data model grow?

8. Why might a data model contract?
9. How do you indicate ordering of instances in a data model?
10. What is the difference between a synonym and a homonym?

## Data modeling

Create a data model from the following narratives, which are sometimes intentionally incomplete. You will have to make some assumptions. Make certain you state these alongside your data model. Define the identifier(s) and attributes of each entity.

1. The president of a book wholesaler has told you that she wants information about publishers, authors, and books.
2. A university has many subject areas (e.g., MIS, Romance languages). Professors teach in only one subject area, but the same subject area can have many professors. Professors can teach many different courses in their subject area. An offering of a course (e.g., Data Management 457, French 101) is taught by only one professor at a particular time.
3. Kids'n'Vans retails minivans for a number of manufacturers. Each manufacturer offers several models of its minivan (e.g., SE, LE, GT). Each model comes with a standard set of equipment (e.g., the Acme SE comes with wheels, seats, and an engine). Minivans can have a variety of additional equipment or accessories (radio, air-conditioning, automatic transmission, airbag, etc.), but not all accessories are available for all minivans (e.g., not all manufacturers offer a driver's side airbag). Some sets of accessories are sold as packages (e.g., the luxury package might include stereo, six speakers, cocktail bar, and twin overhead foxtails).
4. Steve operates a cinema chain and has given you the following information:  
 "I have many cinemas. Each cinema can have multiple theaters. Movies are shown throughout the day starting at 11 a.m. and finishing at 1 a.m. Each movie is given a two-hour time slot. We never show a movie in more than one theater at a time, but we do shift movies among theaters because seating capacity varies. I am interested in knowing how many people, classified by adults and children, attended each showing of a movie. I vary ticket prices by movie and time slot. For instance, Lassie Get Lost is 50 cents for everyone at 11 a.m. but is 75 cents at 11 p.m."
5. A university gymnastics team can have as many as 10 gymnasts. The team competes many times during the season. A meet can have one or more opponents and consists of four events: vault, uneven bars, beam, and floor routine. A gymnast can participate in all or some of these events though the team is limited to five participants in any event.
6. A famous Greek shipping magnate, Stell, owns many container ships. Containers are collected at one port and delivered to another port. Customers pay a negotiated fee for the delivery of each container. Each ship has a sailing schedule that lists the ports the ship will visit over the next six months. The schedule shows the expected arrival and departure dates. The daily charge for use of each port is also recorded.
7. A medical center employs several physicians. A physician can see many patients, and a patient can be seen by many physicians, though not always on the one visit. On any particular visit, a patient may be diagnosed to have one or more illnesses.
8. A telephone company offers a 10 percent discount to any customer who phones another person who is also a customer of the company. To be eligible for the discount, the pairing of the two phone numbers must be registered with the telephone company. Furthermore, for billing purposes, the company records both phone numbers, start time, end time, and date of call.

9. Global Trading (GT), Inc. is a conglomerate. It buys and sells businesses frequently and has difficulty keeping track of what strategic business units (SBUs) it owns, in what nations it operates, and what markets it serves. For example, the CEO was recently surprised to find that GT owns 25 percent of Dundee's Wild Adventures, headquartered in Zaire, that has subsidiaries operating tours of Australia, Zaire, and New York. You have been commissioned to design a database to keep track of GT's businesses. The CEO has provided you with the following information:

SBUs are headquartered in one country, not necessarily the United States. Each SBU has subsidiaries or foreign agents, depending on local legal requirements, in a number of countries. Each subsidiary or foreign agent operates in only one country but can operate in more than one market. GT uses the standard industrial code (SIC) to identify a market (e.g., newspaper publishing). The SIC is a unique four-digit code.

While foreign agents operate as separate legal entities, GT needs to know in what countries and markets they operate. On the other hand, subsidiaries are fully or partly owned by GT, and it is important for GT to know who are the other owners of any subsidiary and what percentage of the subsidiary they own. It is not unusual for a corporation to have shares in several of GT's subsidiary companies and for several corporations to own a portion of a subsidiary. Multiple ownership can also occur at the SBU level.

10. A real estate investment company owns many shopping malls. Each mall contains many shops. To encourage rental of its shops, the company gives a negotiated discount to retailers who have shops in more than one mall. Each shop generates an income stream that can vary from month to month because rental is based on a flat rental charge and a negotiated percentage of sales revenue. Also, each shop has monthly expenses for scheduled and unscheduled maintenance. The company uses the data to compute its monthly net income per square meter for each shop and for ad hoc querying.
11. Draw a data model for the following table taken from a magazine that evaluates consumer goods. The reports follow a standard fashion of listing a brand and model, price, overall score, and then an evaluation of a series of attributes, which can vary with the product. For example, the sample table evaluates stereo systems. A table for evaluating microwave ovens would have a similar layout, but different features would be reported (e.g., cooking quality).

Brand and model	Price	Overall score	Sound quality	Taping quality	FM tuning	CD handling	Ease of use
Philips SC-A K103	140	62	Very good	Good	Very good	Excellent	Fair
Panasonic MC-50	215	55	Good	Good	Very good	Very good	Good
Rio G300	165	38	Good	Good	Fair	Very good	Poor

12. Draw a data model for the following freight table taken from a mail order catalog.

Merchandise subtotals	Regular delivery 7–10 days	Rush delivery 4–5 business days	Express delivery 1–2 business days
Up to \$30.00	\$4.95	\$9.95	\$12.45
\$ 30.01–\$65.00	\$6.95	\$11.95	\$15.45



Merchandise subtotals	Regular delivery 7–10 days	Rush delivery 4–5 business days	Express delivery 1–2 business days
\$6 5.01–\$125.00	\$8.95	\$13.95	\$20.45
\$125.01+	\$9.95	\$15.95	\$25.45

## Reference 1: Basic Structures

*Few things are harder to put up with than the annoyance of a good example.*

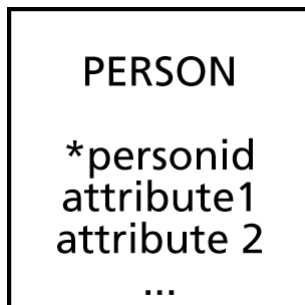
Mark Twain, *Pudd'nhead Wilson*, 1894

Every data model is composed of the same basic structures. This is a major advantage because you can focus on a small part of a full data model without being concerned about the rest of it. As a result, translation to a relational database is very easy because you systematically translate each basic structure. This section describes each of the basic structures and shows how they are mapped to a relational database. Because the mapping is shown as a diagram and SQL CREATE statements, you might use this section frequently when first learning data modeling.

### One entity

#### No relationships

The unrelated entity was introduced in Chapter 3. This is simply a flat file, and the mapping is very simple. Although it is unlikely that you will have a data model with a single entity, it is covered for completeness.



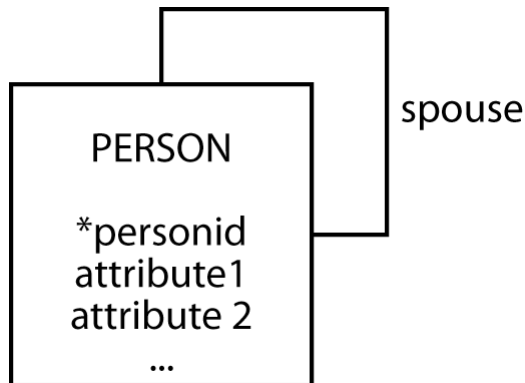
```

CREATE TABLE person (
  personid      INTEGER,
  attribute1    ... ,
  attribute2    ... ,
  ...
  PRIMARY KEY(personid));
  
```

#### A 1:1 recursive relationship

A recursive one-to-one (1:1) relationship is used to describe situations like current marriage. In most countries, a person can legally have zero or one current spouse. The relationship should be labeled to avoid misunderstandings. Mapping to the relational database requires that the identifier of the one end of the relationship becomes a foreign key. It does not matter which one you select. Notice that when **personid** is used as a foreign key, it must be given a different column name—in this case **partner**—because two

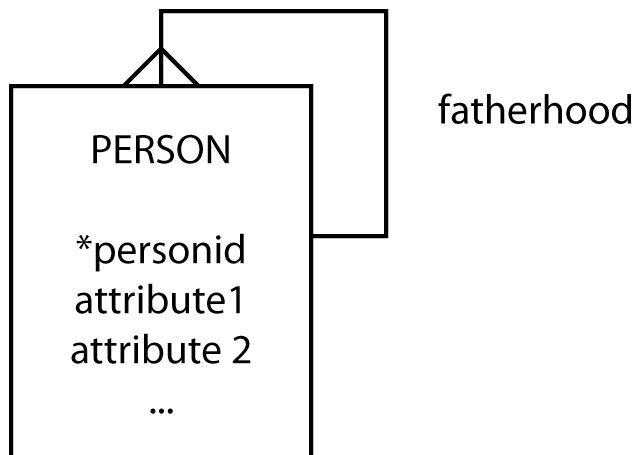
columns in the same table cannot have the same name. The foreign key constraint is not defined, because this constraint cannot refer to the table being created.



```
CREATE TABLE person (
  personid      INTEGER,
  attribute1    ... ,
  attribute2    ... ,
  ...
  partner       INTEGER,
  PRIMARY KEY(personid));
```

### A recursive 1:m relationship

A recursive one-to-many (1:m) relationship describes situations like fatherhood or motherhood. The following figure maps fatherhood. A father may have many biological children, but a child has only one biological father. The relationship is mapped like any other 1:m relationship. The identifier of the one end becomes a foreign key in the many end. Again, we must rename the identifier when it becomes a foreign key in the same table. Also, again the foreign key constraint is not defined because it cannot refer to the table being created.



```
CREATE TABLE person (
  personid      INTEGER,
  Attribute1    ... ,
  Attribute2    ... ,
  ...
  ...)
```

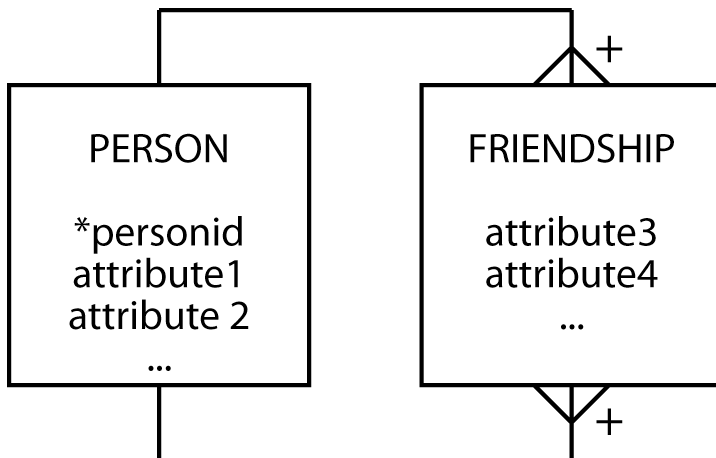
```
father    INTEGER,
PRIMARY KEY(personid));
```

It is possible to have more than one 1:m recursive relationship. For example, details of a mother-child relationship would be represented in the same manner and result in the data model having a second 1:m recursive relationship. The mapping to the relational model would result in an additional column to contain a foreign key *mother*, the *personid* of a person's mother.

### A recursive m:m relationship

A recursive many-to-many (m:m) relationship can describe a situation such as friendship. A person can have many friends and be a friend to many persons. As with m:m relationships between a pair of entities, we convert this relationship to two 1:m relationships and create an associative entity.

The resulting entity friendship has a composite primary key based on the identifier of person, which in effect means the two components are based on *personid*. To distinguish between them, the columns are called *personid1* and *personid2*, so you can think of friendship as a pair of *personids*. You will see the same pattern occurring with other m:m recursive relationships. Notice both person identifiers are independent foreign keys, because they are used to map the two 1:m relationships between person and friendship.



```
CREATE TABLE person (
  personid    INTEGER,
  Attribute1  ... ,
  attribute2  ... ,
  ...
  PRIMARY KEY(personid));
```

```
CREATE TABLE friendship (
  personid1   INTEGER,
  personid2   INTEGER,
  attribute3  ... ,
  attribute4  ... ,
  ...
  PRIMARY KEY(personid1,personid2),
  CONSTRAINT fk_friendship_person1
    FOREIGN KEY(personid1) REFERENCES person(personid),
  CONSTRAINT fk_friendship_person2
    FOREIGN KEY(personid2) REFERENCES person(personid));
```

A single entity can have multiple m:m recursive relationships. Relationships such as enmity (not enemysip) and siblinghood are m:m recursive on person. The approach to recording these relationships is the same as that outlined previously.

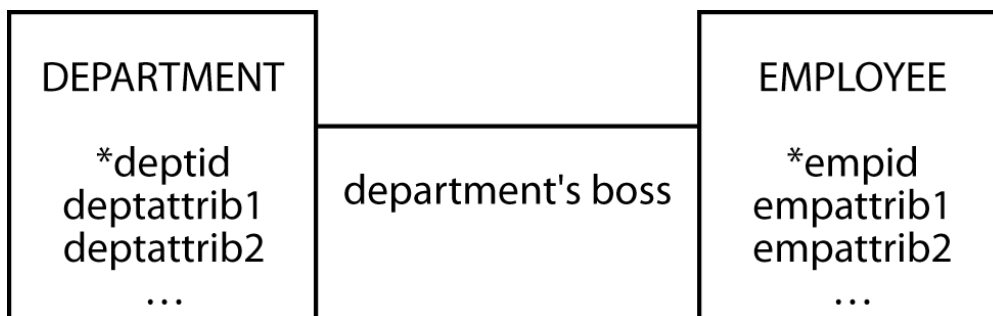
## Two entities

### No relationship

When there is no relationship between two entities, the client has decided there is no need to record a relationship between the two entities. When you are reading the data model with the client, be sure that you check whether this assumption is correct both now and for the foreseeable future. When there is no relationship between two entities, map them each as you would a single entity with no relationships.

### A 1:1 relationship

A 1:1 relationship sometimes occurs in parallel with a 1:m relationship between two entities. It signifies some instances of an entity that have an additional role. For example, a department has many employees (the 1:m relationship), and a department has one boss (the 1:1). The data model fragment shown in the following figure represents the 1:1 relationship.



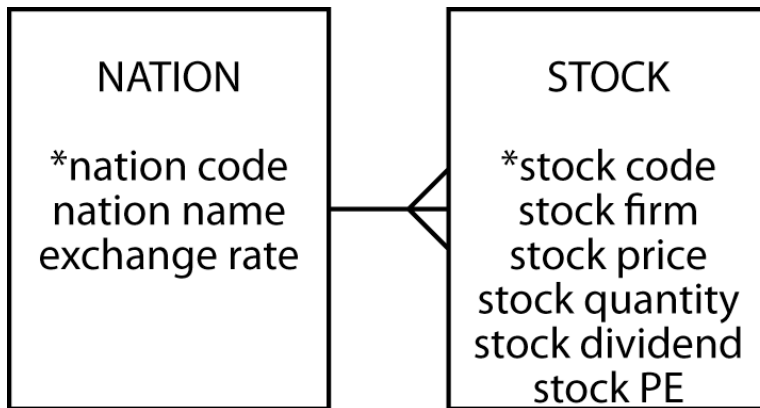
The guideline, as explained in Chapter 6, is to map the relationship to the relational model by placing the foreign key at the mandatory relationship side of the relationship. In this case, we place the foreign key in `department` because each department must have an employee who is the boss.

```
CREATE TABLE employee (  
    empid          INTEGER,  
    empattrib1     ... ,  
    empattrib2     ... ,  
    ...  
    PRIMARY KEY(empid));
```

```
CREATE TABLE department (  
    deptid         CHAR(3),  
    deptattrib1    ... ,  
    deptattrib2    ... ,  
    ...  
    bossid INTEGER,  
    PRIMARY KEY(deptid),  
    CONSTRAINT fk_department_employee  
        FOREIGN KEY(bossid) REFERENCES employee(empid));
```

### A 1:m relationship

The 1:m relationship is possibly the easiest to understand and map. The mapping to the relational model is very simple. The primary key of the “one” end becomes a foreign key in the “many” end.

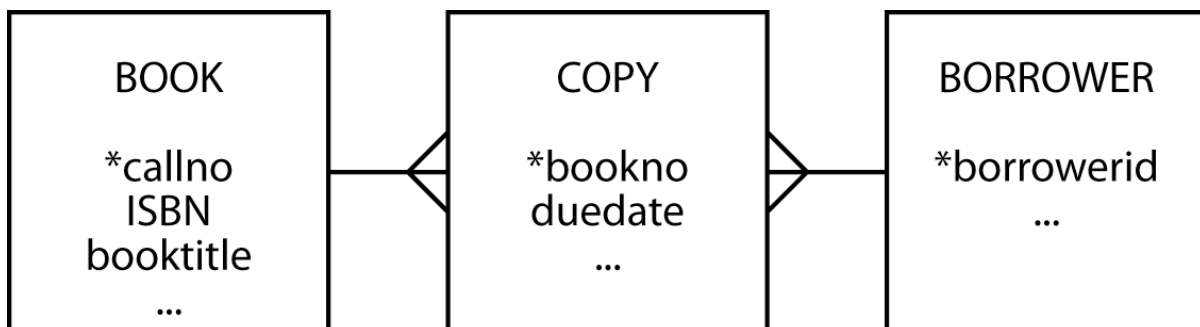


```
CREATE TABLE nation (  
  natcode      CHAR(3),  
  natname     VARCHAR(20),  
  exchrates   DECIMAL(9,5),  
  PRIMARY KEY(natcode));
```

```
CREATE TABLE stock (  
  stkcode     CHAR(3),  
  stkfirm     VARCHAR(20),  
  stkprice    DECIMAL(6,2),  
  stkqty      DECIMAL(8),  
  stkdiv      DECIMAL(5,2),  
  stkpe       DECIMAL(5),  
  natcode     CHAR(3),  
  PRIMARY KEY(stkcode),  
  CONSTRAINT fk_stock_nation  
    FOREIGN KEY(natcode) REFERENCES nation(natcode));
```

### An m:m relationship

An m:m relationship is transformed into two 1:m relationships. The mapping is then a twofold application of the 1:m rule.



The `book` and `borrower` tables must be created first because `copy` contains foreign key constraints that refer to `book` and `borrower`. The column `borrowerid` can be null because a book need not be borrowed; if it's sitting on the shelf, there is no borrower.

```
CREATE TABLE book (
  callno      VARCHAR(12),
  isbn        ... ,
  booktitle   ... ,
  ...
  PRIMARY KEY(callno));
```

```
CREATE TABLE borrower (
  borrowerid   INTEGER,
  ...
  PRIMARY KEY(borrowerid));
```

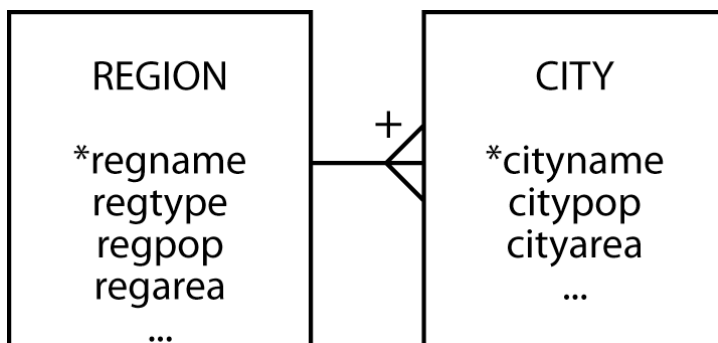
```
CREATE TABLE copy (
  bookno       INTEGER,
  duedate      DATE,
  ...
  callno       VARCHAR(12),
  borrowerid   INTEGER,
  PRIMARY KEY(bookno),
  CONSTRAINT fk_copy_book
    FOREIGN KEY(callno) REFERENCES book(callno),
  CONSTRAINT fk_copy_borrower
    FOREIGN KEY (borrowerid) REFERENCES borrower(borrowerid));
```

## Another entity's identifier as part of the identifier

Using one entity's identifier as part of another entity's identifier tends to cause the most problems for novice data modelers. (One entity's identifier is part of another identifier when there is a plus sign on an arc. The plus is almost always at the crow's foot end of a 1:m relationship.) Tables are formed by applying the following rule: The primary key of the table at the other end of the relationship becomes both a foreign key and part of the primary key in the table at the plus end. The application of this rule is shown for several common data model fragments.

### A weak or dependent entity

In the following figure, *regname* is part of the identifier (signified by the plus near the crow's foot) and a foreign key of *city* (because of the 1:m between *region* and *city*).



```
CREATE TABLE region (
  regname    VARCHAR(20),
  regtype    ...,
  regpop     ...,
  regarea    ...,
  ...
  PRIMARY KEY(regname));
```

```
CREATE TABLE city (
  cityname    VARCHAR(20),
  citypop     ... ,
  cityarea    ... ,
  ...
  regname     VARCHAR(20),
  PRIMARY KEY(cityname,regname),
  CONSTRAINT fk_city_region
  FOREIGN KEY(regname) REFERENCES region(regname));
```

### An associative entity

In the following figure, observe that cityname and firmname are both part of the primary key (signified by the plus near the crow's foot) and foreign keys (because of the two 1:m relationships) of store.



```
CREATE TABLE city (
  cityname    VARCHAR(20),
  ...
  PRIMARY KEY(cityname));
```

```
CREATE TABLE firm
  firmname     VARCHAR(15),
  firmstreet   ... ,
  firmzip      ... ,
  ...
  PRIMARY KEY(firmname));
```

```
CREATE TABLE store (
  storestreet  VARCHAR(30),
  storezip     ... ,
  ...
  cityname     VARCHAR(20),
  firmname     VARCHAR(15),
```

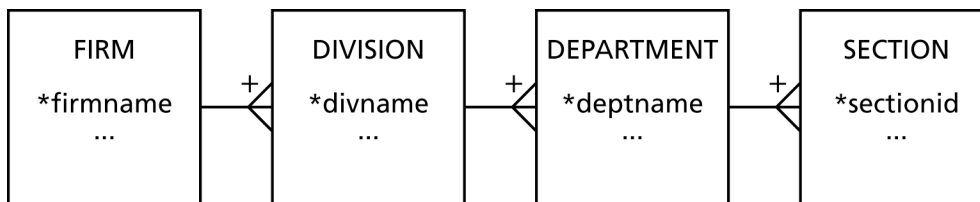
```

PRIMARY KEY(storestreet,cityname,firmname),
CONSTRAINT fk_store_city
    FOREIGN KEY(cityname) REFERENCES city(cityname),
CONSTRAINT fk_store_firm
    FOREIGN KEY(firmname) REFERENCES firm(firmname));

```

## A tree structure

The interesting feature of the following figure is the primary key. Notice that the primary key of a lower level of the tree is a composite of its partial identifier and the primary key of the immediate higher level. The primary key of department is a composite of deptname, divname, and firmname. Novice modelers often forget to make this translation.



```

CREATE TABLE firm (
    firmname    VARCHAR(15),
    ...
    PRIMARY KEY(firmname));

```

```

CREATE TABLE division (
    divname     VARCHAR(15),
    ...
    firmname    VARCHAR(15),
    PRIMARY KEY(firmname,divname),
    CONSTRAINT fk_division_firm
        FOREIGN KEY(firmname) REFERENCES firm(firmname));

```

```

CREATE TABLE department (
    deptname    VARCHAR(15),
    ...
    divname     VARCHAR(15),
    firmname    VARCHAR(15),
    PRIMARY KEY(firmname,divname,deptname),
    CONSTRAINT fk_department_division
        FOREIGN KEY (firmname,divname)
            REFERENCES division(firmname,divname));

```

```

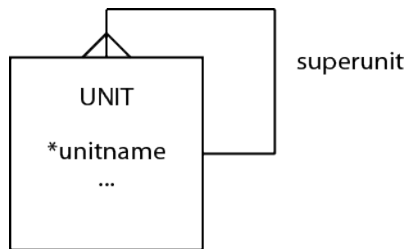
CREATE TABLE section (
    sectionid   VARCHAR(15),
    ...
    divname     VARCHAR(15),
    firmname    VARCHAR(15),
    deptname    VARCHAR(15),
    PRIMARY KEY(firmname,divname,deptname,sectionid),
    CONSTRAINT fk_department_department

```



```
FOREIGN KEY (firmname,divname,deptname)
REFERENCES department(firmname,divname,deptname));
```

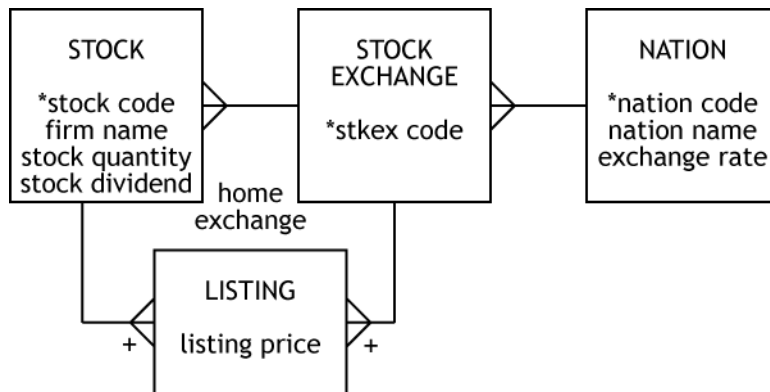
**Another approach to a tree structure** A more general approach to modeling a tree structure is to recognize that it is a series of 1:m recursive relationships. Thus, it can be modeled as follows. This model is identical in structure to that of recursive 1:m reviewed earlier in this chapter and converted to a table in the same manner. Notice that we label the relationship *superunit*, and this would be a good choice of name for the foreign key.



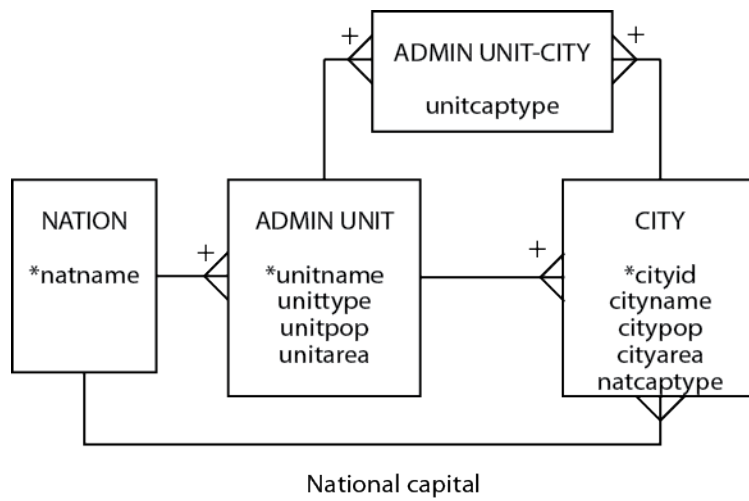
## Exercises

Write the SQL CREATE statements for the following data models.

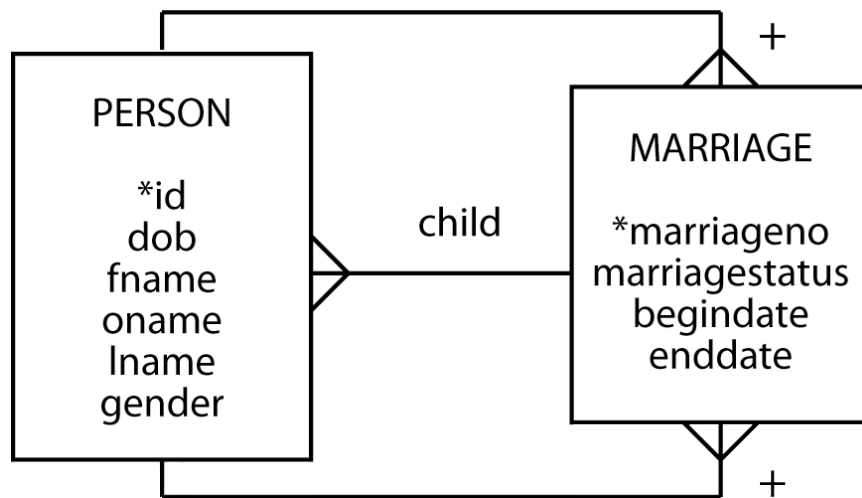
1.



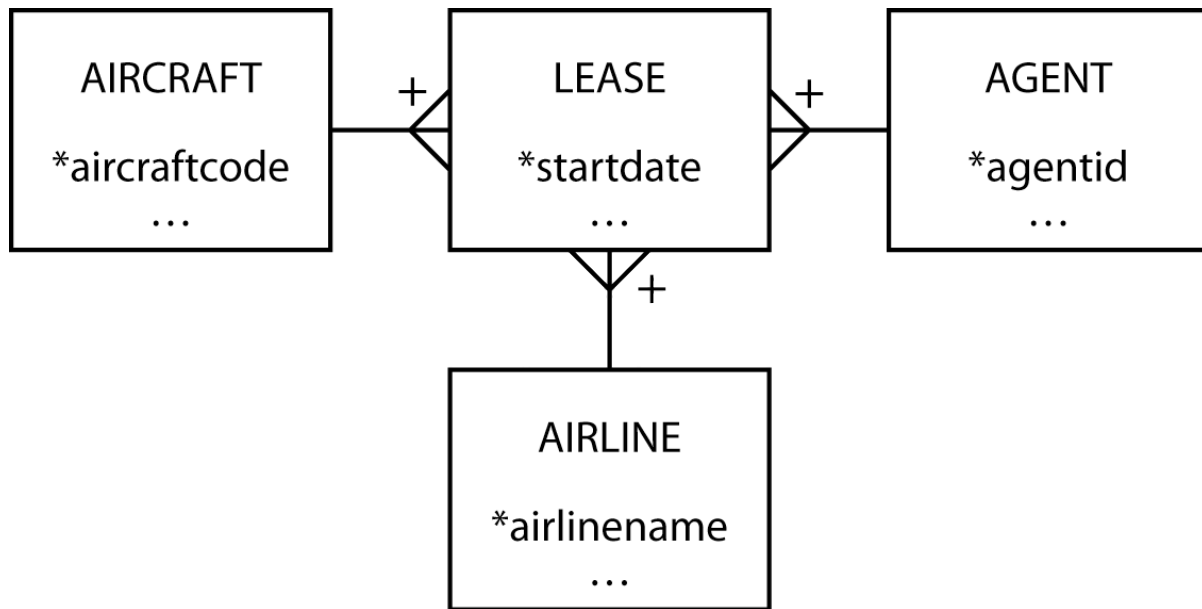
2.



3.



4.



5. Under what circumstances might you use choose a fixed tree over a 1:m recursive data model?

## 8 Normalization and Other Data Modeling Methods

*There are many paths to the top of the mountain, but the view is always the same.*  
Chinese Proverb

### Learning objectives

Students completing this chapter will

- understand the process of normalization;
- be able to distinguish between different normal forms;
- recognize that different data modeling approaches, while they differ in representation methods, are essentially identical.

### Introduction

There are often many ways to solve a problem, and different methods frequently produce essentially the same solution. When this is the case, the goal is to find the most efficient path. We believe that data modeling, as you have learned in the preceding chapters, is an efficient and easily learned approach to database design. There are, however, other paths to consider. One of these methods, normalization, was the initial approach to database design. It was developed as part of the theoretical foundation of the relational data model. It is useful to understand the key concepts of normalization because they advance your understanding of the relational model. Experience and research indicate, however, that normalization is more difficult to master than data modeling.

Data modeling first emerged as entity-relationship (E-R) modeling in a paper by Peter Pin-Shan Chen. He introduced two key database design concepts:

- Identify entities and the relationships between them.
- A graphical representation improves design communication.

Chen's core concepts spawned many species of data modeling. To give you an appreciation of the variation in data modeling approaches, we briefly review, later in this chapter, Chen's E-R approach and IDEF1X, an approach used by the Department of Defense.

## Normalization

Normalization is a method for increasing the quality of database design. It is also a theoretical base for defining the properties of relations. The theory gradually developed to create an understanding of the desirable properties of a relation. The goal of normalization is identical to that of data modeling—a high-fidelity design. The need for normalization seems to have arisen from the conversion of prior file systems into database format. Often, analysts started with the old file design and used normalization to design the new database. Now, designers are more likely to start with a clean slate and use data modeling.

Normal forms can be arrived at in several ways. The recommended approach is data modeling, as experience strongly indicates people find it is an easier path to high quality database design. If the principles of data modeling are followed faithfully, then the outcome should be a high-fidelity model and a normalized database. In other words, if you model data correctly, you create a normalized design. Nevertheless, modeling mistakes can occur, and normalization is a useful crosscheck for ensuring the soundness of a data model. Normalization also provides a theoretical underpinning to data modeling.

Normalization gradually converts a file design into normal form by the successive application of rules to move the design from first to fifth normal form. Before we look at these steps, it is useful to learn about functional dependency.

### Functional dependency

A functional dependency is a relationship between attributes in an entity. It simply means that one or more attributes determine the value of another. For example, given a stock's code, you can determine its current PE ratio. In other words, PE ratio is functionally dependent on stock code. In addition, stock name, stock price, stock quantity, and stock dividend are functionally dependent on stock code. The notation for indicating that stock code functionally determines stock name is

stock code  $\rightarrow$  stock name

An identifier functionally determines all the attributes in an entity. That is, if we know the value of stock code, then we can determine the value of stock name, stock price, and so on.

Formulae, such as  $\text{yield} = \text{stock dividend} / \text{stock price} * 100$ , are a form of functional dependency. In this case, we have

(stock dividend, stock price)  $\rightarrow$  yield

This is an example of **full functional dependency** because yield can be determined only from both attributes.

An attribute, or set of attributes, that fully functionally determines another attribute is called a **determinant**. Thus, stock code is a determinant because it fully functionally determines stock PE. An identifier, usually called a key when discussing normalization, is a determinant. Unlike a key, a determinant need not be unique. For example, a university could have a simple fee structure where undergraduate courses are \$5000 and graduate courses are \$7500. Thus, course type  $\rightarrow$  fee. Since there are many undergraduate and graduate courses, course type is not unique for all records.

There are situations where a given value determines multiple values. This **multidetermination** property is denoted as  $A \rightarrow \rightarrow B$  and reads "A multidetermines B." For instance, a department multidetermines a course.

If you know the department, you can determine the set of courses it offers. **Multivalued dependency** means that functional dependencies are multivalued.

Functional dependency is a property of a relation's data. We cannot determine functional dependency from the names of attributes or the current values. Sometimes, examination of a relation's data will indicate that a functional dependency does not exist, but it is by understanding the relationships between data elements we can determine functional dependency.

Functional dependency is a theoretical avenue for understanding relationships between attributes. If we have two attributes, say A and B, then three relations are possible, as shown in the following table.

*Functional dependencies of two attributes*

<i>Relationship</i>	<i>Functional dependency</i>	<i>Relationship</i>
They determine each other	$A \rightarrow B$ and $B \rightarrow A$	1:1
One determines the other	$A \rightarrow B$	1:m
They do not determine each other	$A \not\rightarrow B$ and $B \not\rightarrow A$	m:m

**One-to-one attribute relationship** Consider two attributes that determine each other ( $A \rightarrow B$  and  $B \rightarrow A$ ), for instance a country's code and its name. Using the example of Switzerland, there is a one-to-one (1:1) relationship between CH and Switzerland:  $CH \rightarrow \text{Switzerland}$  and  $\text{Switzerland} \rightarrow CH$ . When two attributes have a 1:1 relationship, they must occur together in at least one table in a database so that their equivalence is a recorded fact.

**One-to-many attribute relationship** Examine the situation where one attribute determines another (i.e.,  $A \rightarrow B$ ), but the reverse is not true (i.e.,  $A \not\rightarrow B$ ), as is the case with country name and its currency unit. If you know a country's name, you can determine its currency, but if you know the currency unit (e.g., the euro), you cannot always determine the country (e.g., both Italy and Portugal use the euro). As a result, if A and B occur in the same table, then A must be the key. In our example, country name would be the key, and currency unit would be a nonkey column.

**Many-to-many attribute relationship** The final case to investigate is when neither attribute determines the other (i.e.,  $A \not\rightarrow B$  and  $B \not\rightarrow A$ ). The relationship between country name and language is many-to-many (m:m). For example, Belgium has two languages (French and Flemish), and French is spoken in many countries. To record the m:m relationship between these attributes, a table containing both attributes as a composite key is required. This is essentially the associative entity created during data modeling, when there is an m:m relationship between entities.

As you can understand from the preceding discussion, functional dependency is an explicit form of presenting some of the ideas you gained implicitly in earlier chapters on data modeling. The next step is to delve into another theoretical concept underlying the relational model: normal forms.

## Normal forms

Normal forms describe a classification of relations. Initial work by Codd identified first (1NF), second (2NF), and third (3NF) normal forms. Later researchers added Boyce-Codd (BCNF), fourth (4NF), and fifth (5NF) normal forms. Normal forms are nested like a set of Russian dolls, with the innermost doll, 1NF, contained within all other normal forms. The hierarchy of normal forms is 5NF, 4NF, BCNF, 3NF, 2NF, and 1NF. Thus, 5NF is the outermost doll.

A new normal form, domain-key normal form (DK/NF), appeared in 1981. When a relation is in DK/NF, there are no modification anomalies. Conversely, any relation that is free of anomalies must be in DK/NF. The difficult part is discovering how to convert a relation to DK/NF.

## First normal form

*A relation is in first normal form if and only if all columns are single-valued.* In other words, 1NF states that all occurrences of a row must have the same number of columns. In data modeling terms, this means that an attribute must have a single value. An attribute that can have multiple values must be represented as a one-to-many (1:m) relationship. At a minimum, a data model will be in 1NF, because all attributes of an entity are required to be single-valued.

## Second normal form

Second normal form is violated when a nonkey column is dependent on only a component of the primary key. This can also be stated as *a relation is in second normal form if and only if it is in first normal form, and all nonkey columns are dependent on the key.*

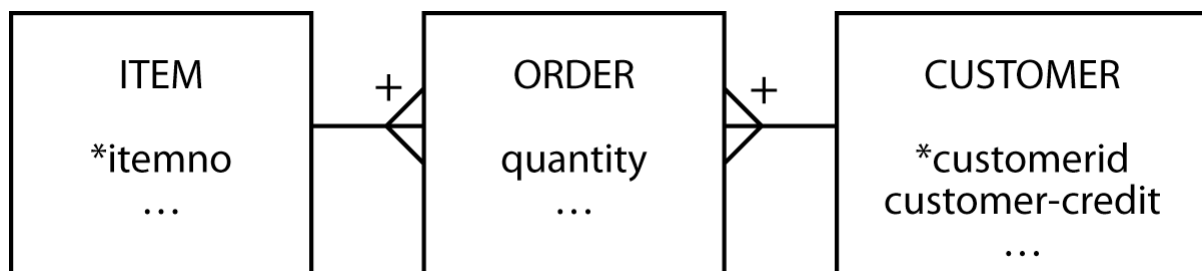
Consider the following table. The primary key of **order** is a composite of **itemno** and **customerid**. The problem is that **customer-credit** is a fact about **customerid** (part of the composite key) rather than the full key (**itemno+customerid**), or in other words, it is not fully functionally dependent on the primary key. An insert anomaly arises when you try to add a new customer to order. You cannot add a customer until that person places an order, because until then you have no value for item number, and part of the primary key will be null. Clearly, this is neither an acceptable business practice nor an acceptable data management procedure.

Analyzing this problem, you realize an item can be in many orders, and a customer can order many items—an m:m relationship. By drawing the data model in the following figure, you realize that *customer-credit* is an attribute of **customer**, and you get the correct relational mapping.

*Second normal form violation*

<u>itemno</u>	<u>customerid</u>	quantity	customer-credit
12	57	25	OK
34	679	3	POOR

*Resolving second normal form violation*



## Third normal form

Third normal form is violated when a nonkey column is a fact about another nonkey column. Alternatively, *a relation is in third normal form if and only if it is in second normal form and has no transitive dependencies.*

The problem in the following table is that **exchange rate** is a fact about **nation**, a nonkey field. In the language of functional dependency, **exchange rate** is not fully functionally dependent on **stockcode**, the primary key.

The functional dependencies are **stockcode** → **nation** → **exchange rate**. In other words, **exchange rate** is transitively dependent on **STOCK**, since **exchange rate** is dependent on **nation**, and **nation** is dependent

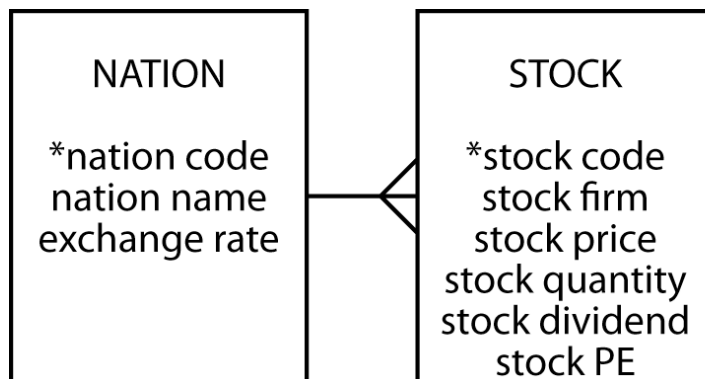
on **stockcode**. The fundamental problem becomes very apparent when you try to add a new nation to the **stock** table. Until you buy at least one stock for that nation, you cannot insert the nation, because you do not have a primary key. Similarly, if you delete MG from the **stock** table, details of the USA exchange rate are lost. There are modification anomalies.

When you think about the data relationships, you realize that a nation has many stocks and a stock belongs to only one nation. Now the data model and relational map can be created readily .

*Third normal form violation*

<u>stockcode</u>	nation	exchange rate
MG	USA	0.67
IR	AUS	0.46

*Resolving third normal form violation*



*Skill builder* You have been given a spreadsheet that contains details of invoices. The column headers for the spreadsheet are *date, invoice number, invoice amount, invoice tax, invoice total, cust number, custname, cust street, cust city, cust state, cust postal code, cust nation, product code, product price, product quantity, salesrep number, salesrep first name, salesrep last name, salesrep district, district name, and district size*. Normalize this spreadsheet so that it can be converted to a high-fidelity relational database.

## Boyce-Codd normal form

The original definition of 3NF did not cover a situation that, although rare, can occur. So Boyce-Codd normal form, a stronger version of 3NF, was developed. BCNF is necessary because 3NF does not cover the cases when

- A relation has multiple candidate keys.
- Those candidate keys are composite.
- The candidate keys overlap because they have at least one column in common.

Before considering an example, **candidate key** needs to be defined. Earlier we introduced the idea that an entity could have more than one potential unique identifier. These identifiers become candidate keys when the data model is mapped to a relational database. One of these candidates is selected as the primary key.

Consider the following case from a management consulting firm. A client can have many types of problems (e.g., finance, personnel), and the same problem type can be an issue for many clients. Consultants specialize

and advise on only one problem type, but several consultants can advise on one problem type. A consultant advises a client. Furthermore, for each problem type, the client is advised by only one consultant. If you did not use data modeling, you might be tempted to create the following table.

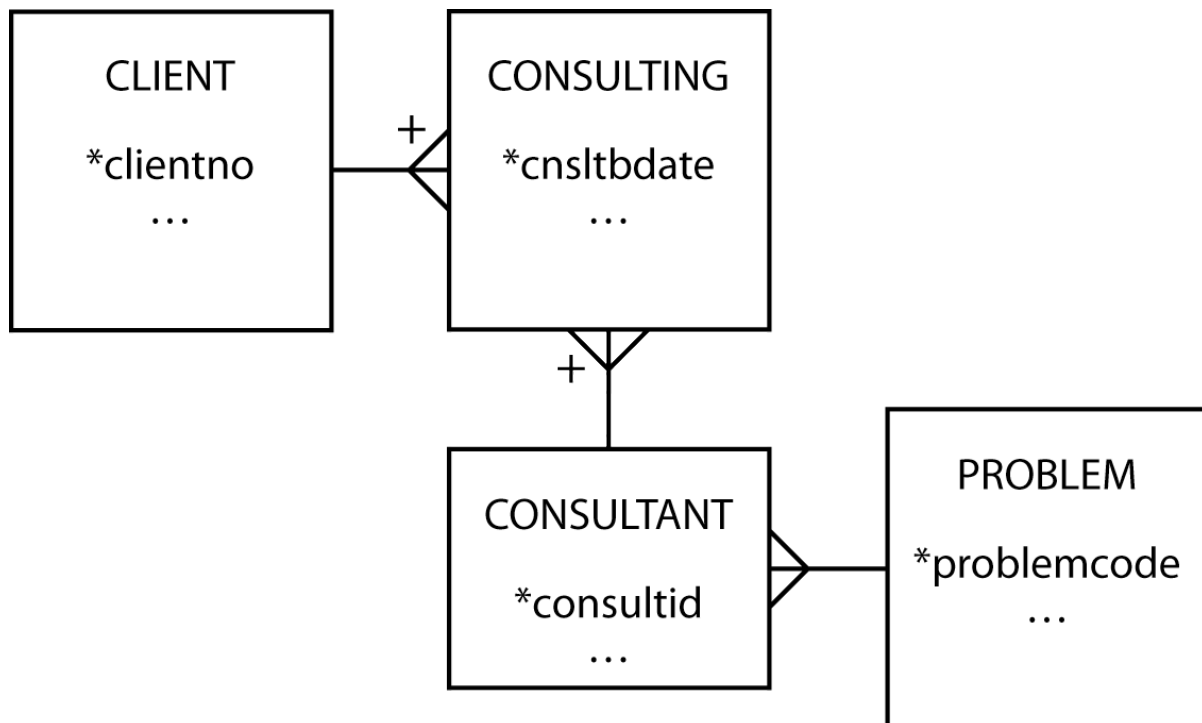
*Boyce-Codd normal form violation*

<u>client</u>	<u>proptype</u>	consultant
Alpha	Marketing	Gomez
Alpha	Production	Raginiski

The column `client` cannot be the primary key because a client can have several problem types; however, a client is advised by only one consultant for a specific problem type, so the composite key `client+proptype` determines `consultant`. Also, because a consultant handles only one type of problem, the composite key `client+consultant` determines `proptype`. So, both of these composites are candidate keys. Either one can be selected as the primary key, and in this case `client+proptype` was selected. Notice that all the previously stated conditions are satisfied—there are multiple, composite candidate keys that overlap. This means the table is 3NF, but not BCNF. This can be easily verified by considering what happens if the firm adds a new consultant. A new consultant cannot be added until there is a client—an insertion anomaly. The problem is that `consultant` is a determinant, `consultant→proptype`, but is not a candidate key. In terms of the phrasing used earlier, the problem is that part of the key column is a fact about a nonkey column. The precise definition is *a relation is in Boyce-Codd normal form if and only if every determinant is a candidate key*.

This problem is avoided by creating the correct data model and then mapping to a relational model.

*Resolving Boyce-Codd normal form violation*



## Fourth normal form

Fourth normal form requires that a row should not contain two or more independent multivalued facts about an entity. This requirement is more readily understood after investigating an example.



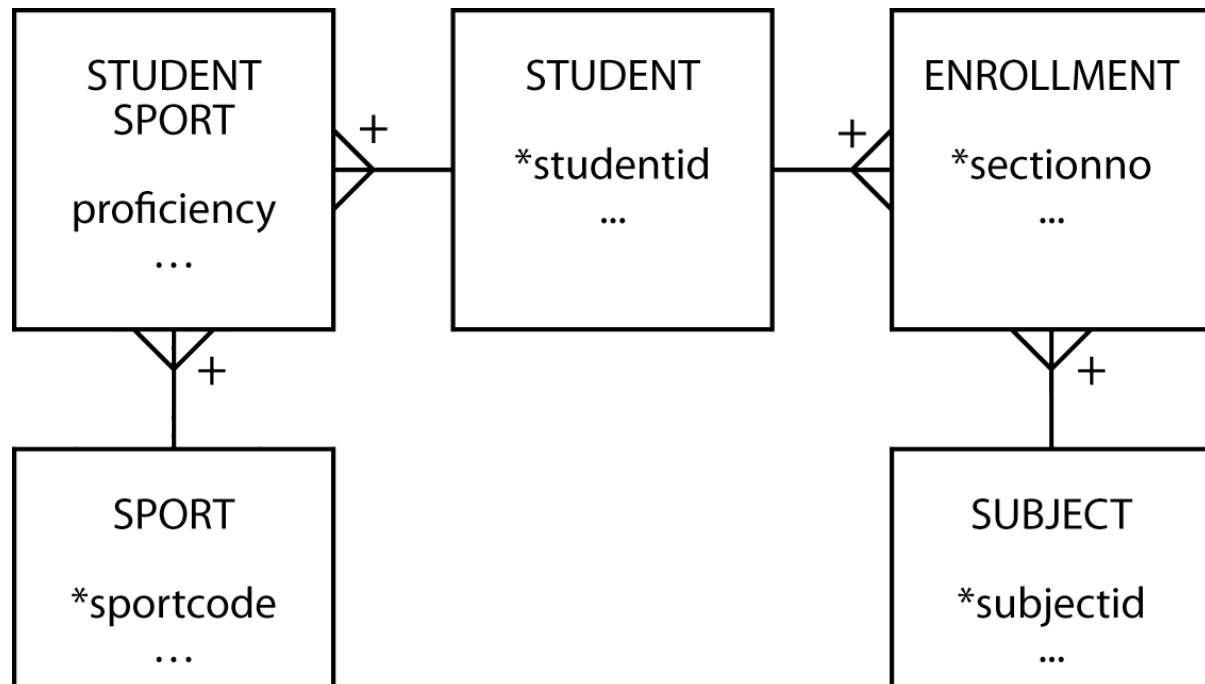
Consider students who play sports and study subjects. One way of representing this information is shown in the following table. Consider the consequence of trying to insert a student who did not play a sport. Sport would be null, and this is not permissible because part of the composite primary key would then be null—a violation of the entity integrity rule. You cannot add a new student until you know her sport and her subject. Modification anomalies are very apparent.

#### *Fourth normal form violation*

<u>studentid</u>	<u>sport</u>	<u>subject</u>	...
50	Football	English	...
50	Football	Music	...
50	Tennis	Botany	...
50	Karate	Botany	...

This table is not in 4NF because sport and subject are independent multivalued facts about a student. There is no relationship between sport and subject. There is an indirect connection because sport and subject are associated with a student. In other words, a student can play many sports, and the same sport can be played by many students—a many-to-many (m:m) relationship between student and sport. Similarly, there is an m:m relationship between student and subject. It makes no sense to store information about a student's sports and subjects in the same table because sport and subject are not related. The problem arises because sport and subject are multivalued dependencies of student. The solution is to convert multivalued dependencies to functional dependencies. More formally, *a relation is in fourth normal form if it is in Boyce-Codd normal form and all multivalued dependencies on the relation are functional dependencies*. A data model sorts out this problem, and note that the correct relational mapping requires five tables.

#### *Resolving fourth normal form violation*

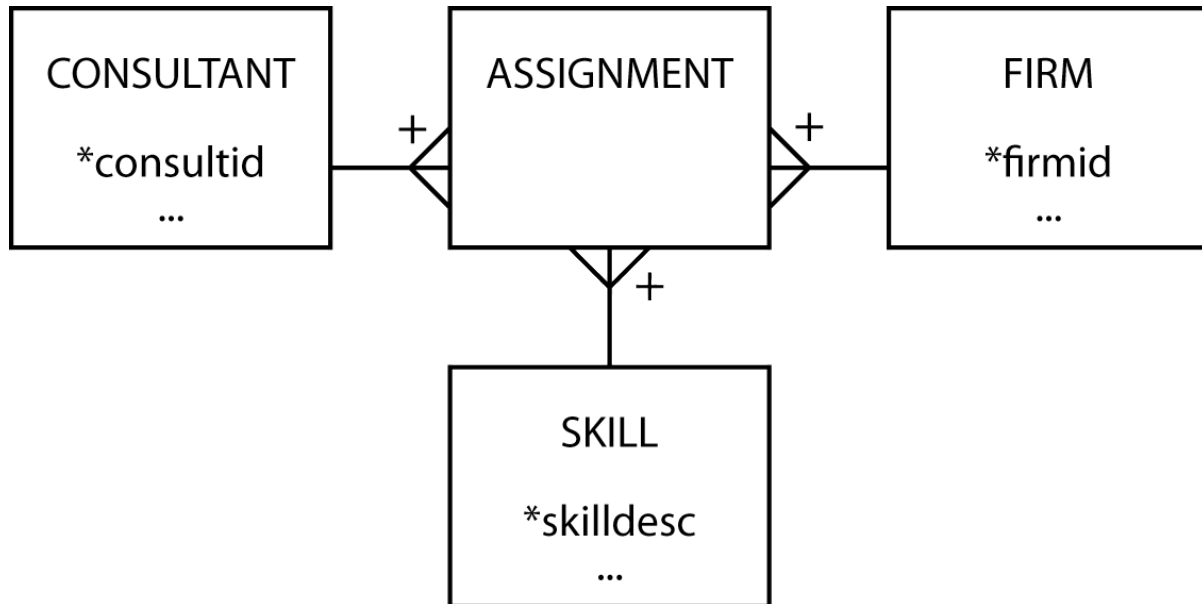


#### **Fifth normal form**

Fifth normal form deals with the case where a table can be reconstructed from other tables. The reconstruction approach is preferred, because it means less redundancy and fewer maintenance problems.

A consultants, firms, and skills problem is used to illustrate the concept of 5NF. The problem is that consultants provide skills to one or more firms and firms can use many consultants; a consultant has many skills and a skill can be used by many firms; and a firm can have a need for many skills and the same skill can be required by many firms. The data model for this problem has the ménage-à-trois structure which was introduced in Chapter 7.

*The CONSULTANT-FIRM-SKILL data model without a rule*



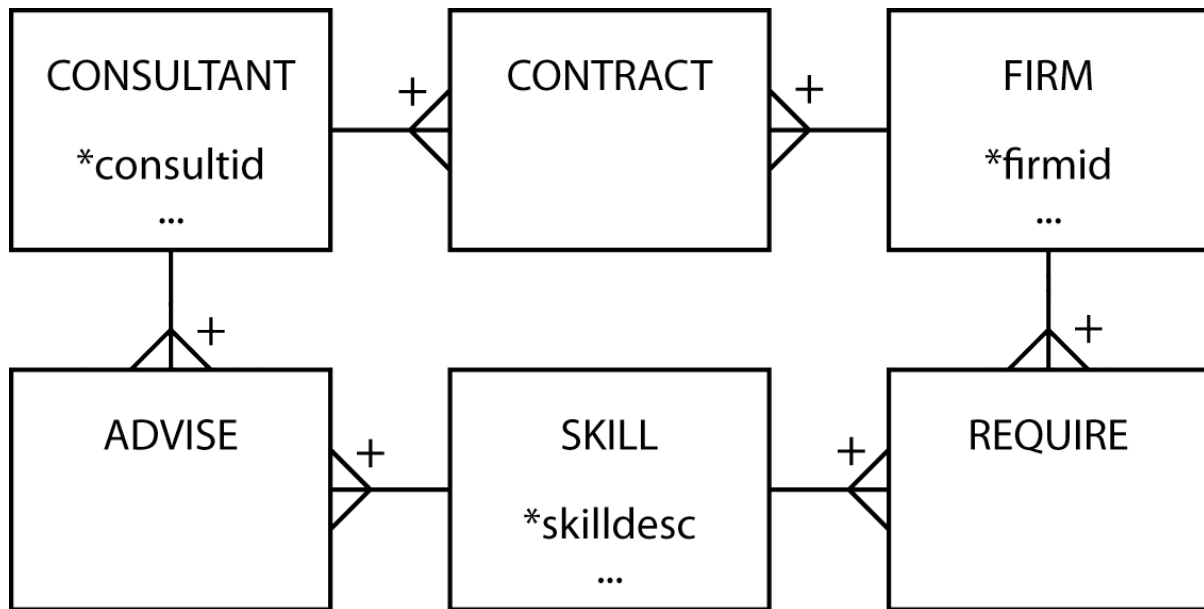
The relational mapping of the three-way relationships results in four tables, with the associative entity ASSIGNMENT mapping shown in the following table.

*Relation table assignment*

<u>consultid</u>	<u>firmid</u>	<u>skilldesc</u>
Tan	IBM	Database
Tan	Apple	Data comm

The table is in 5NF because the combination of all three columns is required to identify which consultants supply which firms with which skills. For example, we see that Tan advises IBM on database and Apple on data communications. The data in this table cannot be reconstructed from other tables.

The preceding table is not in 5NF *if there is a rule of the following form*: If a consultant has a certain skill (e.g., database) and has a contract with a firm that requires that skill (e.g., IBM), then the consultant advises that firm on that skill (i.e., he advises IBM on database). Notice that this rule means we can infer a relationship, and we no longer need the combination of the three columns. As a result, we break the single three-entity m:m relationship into three two-entity m:m relationships. Revised data model after the introduction of the rule



Further understanding of 5NF is gained by examining the relational tables resulting from these three associative entities. Notice the names given to each of the associative entities. The table **contract** records data about the firms a consultant advises; **advise** describes the skills a consultant has; and **require** stores data about the types of skills each firm requires. To help understand this problem, examine the three tables below.

*Relational table contract*

<u>consultid</u>	<u>firmid</u>
Gonzales	Apple
Gonzales	IBM
Gonzales	NEC
Tan	IBM
Tan	NEC
Wood	Apple

*Relational table advise*

<u>consultid</u>	<u>skilldesc</u>
Gonzales	Database
Gonzales	Data comm
Gonzales	Groupware
Tan	Database
Tan	Data comm
Wood	Data comm

*Relational table require*

<u>firmid</u>	<u>skilldesc</u>
IBM	Data comm
IBM	Groupware

<u>firmid</u>	<u>skilldesc</u>
NEC	Data comm
NEC	Database
NEC	Groupware
Apple	Data comm

Consider joining **contract** and **advise**, the result of which we call **could advise** because it lists skills the consultant could provide if the firm required them. For example, Tan has skills in database and data communications, and Tan has a contract with IBM. If IBM requires database skills, Tan can handle it. We need to look at require to determine whether IBM requires advice on database; it does not.

*Relational table could advise*

<u>consultid</u>	<u>firmid</u>	<u>skilldesc</u>
Gonzales	Apple	Database
Gonzales	Apple	Data comm
Gonzales	Apple	Groupware
Gonzales	IBM	Database
Gonzales	IBM	Data comm
Gonzales	IBM	Groupware
Gonzales	NEC	Database
Gonzales	NEC	Data comm
Gonzales	NEC	Groupware
Tan	IBM	Database
Tan	IBM	Data comm
Tan	NEC	Database
Tan	NEC	Data comm
Wood	Apple	Data comm

*Relational table can advise*

The join of **could advise** with **require** gives details of a firm's skill needs that a consultant can provide. The table **can advise** is constructed by directly joining **contract**, **advise**, and **require**. Because we can construct **can advise** from three other tables, the data are in 5NF.

<u>consultid</u>	<u>firmid</u>	<u>skilldesc</u>
Gonzales	IBM	Data comm
Gonzales	IBM	Groupware
Gonzales	NEC	Database
Gonzales	NEC	Data comm
Gonzales	NEC	Groupware
Tan	IBM	Data comm
Tan	NEC	Database
Tan	NEC	Data comm
Wood	Apple	Data comm

Since data are stored in three separate tables, updating is easier. Consider the case where IBM requires database skills. We only need to add one row to require to record this fact. In the case of can advise, we would have to add two rows, one for Gonzales and one for Tan.

Now we can give 5NF a more precise definition: *A relation is in fifth normal form if and only if every join dependency of the relation is a consequence of the candidate keys of the relation.*

Up to this point, data modeling has enabled you to easily avoid normalization problems. Fifth normal form introduces a complication, however. How can you tell when to use a single three-way associative entity or three two-way associative entities? If there is a constraint or rule that is applied to the relationships between entities, consider the possibility of three two-way associative entities. Question the client carefully whenever you find a ménage-à-trois. Check to see that there are no rules or special conditions.

### Domain-key normal form

The definition of DK/NF builds on three terms: key, constraint, and domain. You already know a key is a unique identifier. A **constraint** is a rule governing attribute values. It must be sufficiently well-defined so that its truth can be readily evaluated. Referential integrity constraints, functional dependencies, and data validation rules are examples of constraints. A **domain** is a set of all values of the same data type. With the help of these terms, the concept of DK/NF is easily stated. *A relation is in domain-key normal form if and only if every constraint on the relation is a logical consequence of the domain constraints and the key constraints that apply to the relation.*

Note that DK/NF does not involve ideas of dependency; it just relies on the concepts of key, constraint, and domain. The problem with DK/NF is that while it is conceptually simple, no algorithm has been developed for converting relations to DK/NF. Hence, database designers must rely on their skills to create relations that are in DK/NF.

### Conclusion

Normalization provides designers with a theoretical basis for understanding what they are doing when modeling data. It also alerts them to be aware of problems that they might not ordinarily detect when modeling. For example, 5NF cautions designers to investigate situations carefully (i.e., look for special rules), when their data model contains a ménage-à-trois.

### Other data modeling methods

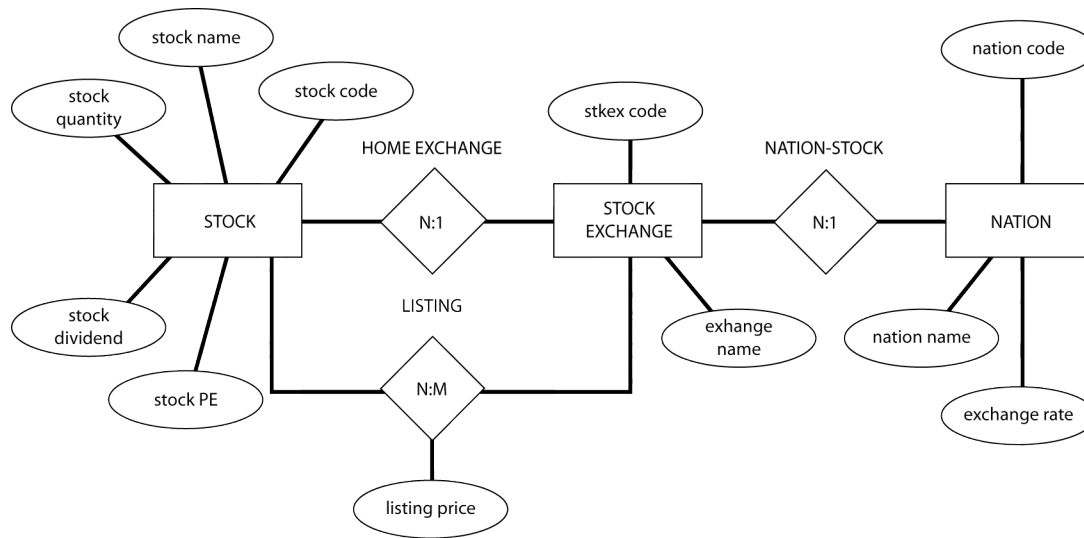
As mentioned previously, there are many species of data modeling. Here we consider two methods.

#### The E-R model

One of the most widely known data modeling methods is the E-R model developed by Chen. There is no standard for the E-R model, and it has been extended and modified in a number of ways.

As the following figure shows, an E-R model looks like the data models with which you are now familiar. Entities are shown by rectangles, and relationships are depicted by diamonds within which the cardinality of the relationship is indicated (e.g., there is a 1:m relationship between NATION and STOCK EXCHANGE). Cardinality can be 1:1, 1:m (conventionally shown as 1:N in E-R diagrams), and m:m (shown as M:N). Recursive relationships are also readily modeled. One important difference is that an m:m relationship is not shown as an associative entity; thus the database designer must convert this relationship to a table.

*An E-R diagram*



Attributes are shown as ellipses connected to the entity or relationship to which they belong (e.g., *stock name* is an attribute of STOCK). Relationships are named, and the name is shown above or below the relationship symbol. For example, LISTING is the name of the m:m relationship between STOCK and STOCK EXCHANGE. The method of recording attributes consumes space, particularly for large data models. It is more efficient to record attributes within an entity rectangle, as in the data modeling method you have learned.

### Skill builder

Ken is a tour guide for trips that can last several weeks. He has requested a database that will enable him to keep track of the people who have been on the various tours he has led. He wants to be able to record his comments about some of his clients to remind him of extra attention he might need to pay to them on a future trip (e.g., Bill tends to get lost, Daisy has problems getting along with others). Some times people travel with a partner, and Ken wants to record this in his database.

Design a database using MySQL Workbench. Experiment with Object Notation and Relationship Notation (options of the Model tab) to get a feel for different forms of representing data models.

## Conclusion

The underlying concepts of different data modeling methods are very similar. They all have the same goal: improving the quality of database design. They share common concepts, such as entity and relationship mapping. We have adopted an approach that is simple and effective. The method can be learned rapidly, novice users can create high-quality models, and it is similar to the representation style of MySQL Workbench. Also, if you have learned one data modeling method, you can quickly adapt to the nuances of other methods.

## Summary

Normalization gradually converts a file design into normal form by successive application of rules to move the design from first to fifth normal form. Functional dependency means that one or more attributes determine the value of another. An attribute, or set of attributes, that fully functionally determines another attribute is called a *determinant*. First normal form states that all occurrences of a row must have the same number of columns. Second normal form is violated when a nonkey column is a fact about a component of the prime key. Third normal form is violated when a nonkey column is a fact about another nonkey column. Boyce-Codd normal form is a stronger version of third normal form. Fourth normal form requires that a

row should not contain two or more independent multivalued facts about an entity. Fifth normal form deals with the case where a table can be reconstructed from data in other tables. Fifth normal form arises when there are special rules or conditions. A high-fidelity data model will be of high normal form.

One of the most widely known methods of data modeling is the E-R model. The basic concepts of most data modeling methods are very similar. All aim to improve database design.

## Key terms and concepts

---

---

Alternate key	Functional dependency
Boyce-Codd normal form (BCNF)	Generalization hierarchy
Constraint	Inversion entry
Data model	Many-to-many attribute relationship
Determinant	Multidetermination
Domain	Multivalued dependency
Domain-key normal form (DK/NF)	Normalization
Entity	One-to-many attribute relationship
Entity-relationship (E-R) model	One-to-one attribute relationship
Fifth normal form (5NF)	Relationship
First normal form (1NF)	Second normal form (2NF)
Fourth normal form (4NF)	Third normal form (3NF)
Full functional dependency	

---

## References and additional readings

Chen, P. 1976. The entity-relationship model—toward a unified view of data. *ACM Transactions on Database Systems* 1 (1):9–36.

Kent, W. 1983. A simple guide to five normal forms in relational database theory. *Communications of the ACM* 26 (2): 120-125.

## Exercises

### Short answers

1. What is normalization, and what is its goal?
2. How does DK/NF differ from earlier normal forms?
3. How does E-R differ from the data modeling method of this text?

### Normalization and data modeling

Using normalization or E-R create data models from the following narratives, which are sometimes intentionally incomplete. You will have to make some assumptions. Make certain you state these assumptions alongside your data model. Define the identifier(s) and attributes of each entity.

1. The president of a book wholesaler has told you that she wants information about publishers, authors, and books.

2. A university has many subject areas (e.g., MIS, Romance languages). Professors teach in only one subject area, but the same subject area can have many professors. Professors can teach many different courses in their subject area. An offering of a course (e.g., Data Management 457, French 101) is taught by only one professor at a particular time.
3. Kids'n'Vans retails minivans for a number of manufacturers. Each manufacturer offers several models of its minivan (e.g., SE, LE, GT). Each model comes with a standard set of equipment (e.g., the Acme SE comes with wheels, seats, and an engine). Minivans can have a variety of additional equipment or accessories (radio, air conditioning, automatic transmission, airbag, etc.), but not all accessories are available for all minivans (e.g., not all manufacturers offer a driver's side airbag). Some sets of accessories are sold as packages (e.g., the luxury package might include stereo, six speakers, cocktail bar, and twin overhead foxtails).
4. Steve operates a cinema chain and has given you the following information:  
 "I have many cinemas. Each cinema can have multiple theaters. Movies are shown throughout the day starting at 11 a.m. and finishing at 1 a.m. Each movie is given a two-hour time slot. We never show a movie in more than one theater at a time, but we do shift movies among theaters because seating capacity varies. I am interested in knowing how many people, classified by adults and children, attended each showing of a movie. I vary ticket prices by movie and time slot. For instance, *Lassie Get Lost* is 50 cents for everyone at 11 a.m. but is 75 cents at 11 p.m."
5. A telephone company offers a 10 percent discount to any customer who phones another person who is also a customer of the company. To be eligible for the discount, the pairing of the two phone numbers must be registered with the telephone company. Furthermore, for billing purposes, the company records both phone numbers, start time, end time, and date of call.

## 9 The Relational Model and Relational Algebra

*Nothing is so practical as a good theory.*

K. Lewin, 1945

### Learning objectives

Students completing this chapter will

- know the structures of the relational model;
- understand relational algebra commands;
- be able to determine whether a DBMS is completely relational.

### Background

The relational model, developed as a result of recognized shortcomings of hierarchical and network DBMSs, was introduced by Codd in 1970. As the major developer, Codd believed that a sound theoretical model would solve most practical problems that could potentially arise.

In another article, Codd expounds the case for adopting the relational over earlier database models. There is a threefold thrust to his argument. *First*, earlier models force the programmer to code at a low level of structural detail. As a result, application programs are more complex and take longer to write and debug. *Second*, no commands are provided for processing multiple records at one time. Earlier models do not provide the set-processing capability of the relational model. The set-processing feature means that queries can be more concisely expressed. *Third*, the relational model, through a query language such as structured query



language (SQL), recognizes the clients' need to make ad hoc queries. The relational model and SQL permits an IS department to respond rapidly to unanticipated requests. It can also mean that analysts can write their own queries. Thus, Codd's assertion that the relational model is a practical tool for increasing the productivity of IS departments is well founded.

The productivity increase arises from three of the objectives that drove Codd's research. The *first* was to provide a clearly delineated boundary between the logical and physical aspects of database management. Programming should be divorced from considerations of the physical representation of data. Codd labels this the **data independence objective**. The *second* objective was to create a simple model that is readily understood by a wide range of analysts and programmers. This **communicability objective** promotes effective and efficient communication between clients and IS personnel. The *third* objective was to increase processing capabilities from record-at-a-time to multiple-records-at-a-time—the **set-processing objective**. Achievement of these objectives means fewer lines of code are required to write an application program, and there is less ambiguity in client-analyst communication.

The relational model has three major components:

- Data structures
- Integrity rules
- Operators used to retrieve, derive, or modify data

## Data structures

Like most theories, the relational model is based on some key structures or concepts. We need to understand these in order to understand the theory.

### Domain

A domain is a set of values all of the same data type. For example, the domain of nation name is the set of all possible nation names. The domain of all stock prices is the set of all currency values in, say, the range \$0 to \$10,000,000. You can think of a domain as all the legal values of an attribute.

In specifying a domain, you need to think about the smallest unit of data for an attribute defined on that domain. In Chapter 8, we discussed how a candidate attribute should be examined to see whether it should be segmented (e.g., we divide name into first name, other name, and last name, and maybe more). While it is unlikely that name will be a domain, it is likely that there will be a domain for first name, last name, and so on. Thus, a domain contains values that are in their *atomic* state; they cannot be decomposed further.

The practical value of a domain is to define what comparisons are permissible. Only attributes drawn from the same domain should be compared; otherwise it is a bit like comparing bananas and strawberries. For example, it makes no sense to compare a stock's PE ratio to its price. They do not measure the same thing; they belong to different domains. Although the domain concept is useful, it is rarely supported by relational model implementations.

### Relations

A relation is a table of  $n$  columns (or *attributes*) and  $m$  rows (or *tuples*). Each column has a unique name, and all the values in a column are drawn from the same domain. Each row of the relation is uniquely identified. The order of columns and rows is immaterial.

The **cardinality** of a relation is its number of rows. The **degree** of a relation is the number of columns. For example, the relation nation (see the following figure) is of degree 3 and has a cardinality of 4. Because the cardinality of a relation changes every time a row is added or deleted, you can expect cardinality to change

frequently. The degree changes if a column is added to a relation, but in terms of relational theory, it is considered to be a new relation. So, only a relation's cardinality changes.

A relational database with tables **nation** and **stock**

		<u>natcode</u>	natname	exchrte		
	AUS		Australia	0.46		
	IND		India	0.0228		
	UK		United Kingdom	1		
	USA		United States	0.67	→	

	<u>stkcode</u>	stkfirm	stkprice	stkqty	stkdiv	stkpe	natcode	
FC	Freedonia	Copper	27.5	10,529	1.84	16	UK	
PT	Patagonian	Tea	55.25	12,635	2.5	10	UK	
AR	Abyssinian	Ruby	31.82	22,010	1.32	13	UK	
SLG	Sri Lankan	Gold	50.37	32,868	2.68	16	UK	
ILZ	Indian Lead &	Zinc	37.75	6,390	3	12	UK	
BE	Burmese	Elephant	0.07	154,713	0.01	3	UK	
BS	Bolivian	Sheep	12.75	231,678	1.78	11	UK	
NG	Nigerian	Geese	35	12,323	1.68	10	UK	
CS	Canadian	Sugar	52.78	4,716	2.5	15	UK	
ROF	Royal Ostrich	Farms	33.75	1,234,923	3	6	UK	
MG	Minnesota	Gold	53.87	816,122	1	25	USA	←
GP	Georgia	Peach	2.35	387,333	0.2	5	USA	←
NE	Naremben	Emu	12.34	45,619	1	8	AUS	
QD	Queensland	Diamond	6.73	89,251	0.5	7	AUS	
IR	Indooroopilly	Ruby	15.92	56,147	0.5	20	AUS	
BD	Bombay	Duck	25.55	167,382	1	12	IN	

## Relational database

A relational database is a collection of relations or tables. The distinguishing feature of the relational model, when compared to the hierarchical and network models, is that there are no explicit linkages between tables. Tables are linked by common columns drawn on the same domain; thus, the portfolio database (see the preceding tables) consists of tables **stock** and **nation**. The 1:m relationship between the two tables is represented by the column **natcode** that is common to both tables. Note that the two columns need not have the same name, but they must be drawn on the same domain so that comparison is possible. In the case of an m:m relationship, while a new table must be created to represent the relationship, the principle of linking tables through common columns on the same domain remains in force.

## Primary key

A relation's primary key is its unique identifier; for example, the primary key of **nation** is **natcode**. As you already know, a primary key can be a composite of several columns. The primary key guarantees that each row of a relation can be uniquely addressed.

## Candidate key

In some situations, there may be several attributes, known as candidate keys, that are potential primary keys. Column **natcode** is unique in the **nation** relation, for example. We also can be fairly certain that two nations will not have the same name. Therefore, **nation** has multiple candidate keys: **natcode** and **natname**.

## Alternate key

When there are multiple candidate keys, one is chosen as the primary key, and the remainder are known as alternate keys. In this case, we selected **natcode** as the primary key, and **natname** is an alternate key.

## Foreign key

The foreign key is an important concept of the relational model. It is the way relationships are represented and can be thought of as the glue that holds a set of tables together to form a relational database. A foreign key is an attribute (possibly composite) of a relation that is also a primary key of a relation. The foreign key and primary key may be in different relations or the same relation, but both keys must be drawn from the same domain.

## Integrity rules

The integrity section of the relational model consists of two rules. The **entity integrity rule** ensures that each instance of an entity described by a relation is identifiable in some way. Its implementation means that each row in a relation can be uniquely distinguished. The rule is

*No component of the primary key of a relation may be null.*

Null in this case means that the component cannot be undefined or unknown; it must have a value. Notice that the rule says “component of a primary key.” This means every part of the primary key must be known. If a part cannot be defined, it implies that the particular entity it describes cannot be defined. In practical terms, it means you cannot add a nation to **nation** unless you also define a value for **natcode**.

The definition of a foreign key implies that there is a corresponding primary key. The **referential integrity rule** ensures that this is the case. It states

*A database must not contain any unmatched foreign key values.*

Simply, this rule means that you cannot define a foreign key without first defining its matching primary key. In practical terms, it would mean you could not add a Canadian stock to the **stock** relation without first creating a row in **nation** for Canada.

Notice that the concepts of foreign key and referential integrity are intertwined. There is not much sense in having foreign keys without having the referential integrity rule. Permitting a foreign key without a corresponding primary key means the relationship cannot be determined. Note that the referential integrity rule does not imply a foreign key cannot be null. There can be circumstances where a relationship does not exist for a particular instance, in which case the foreign key is null.

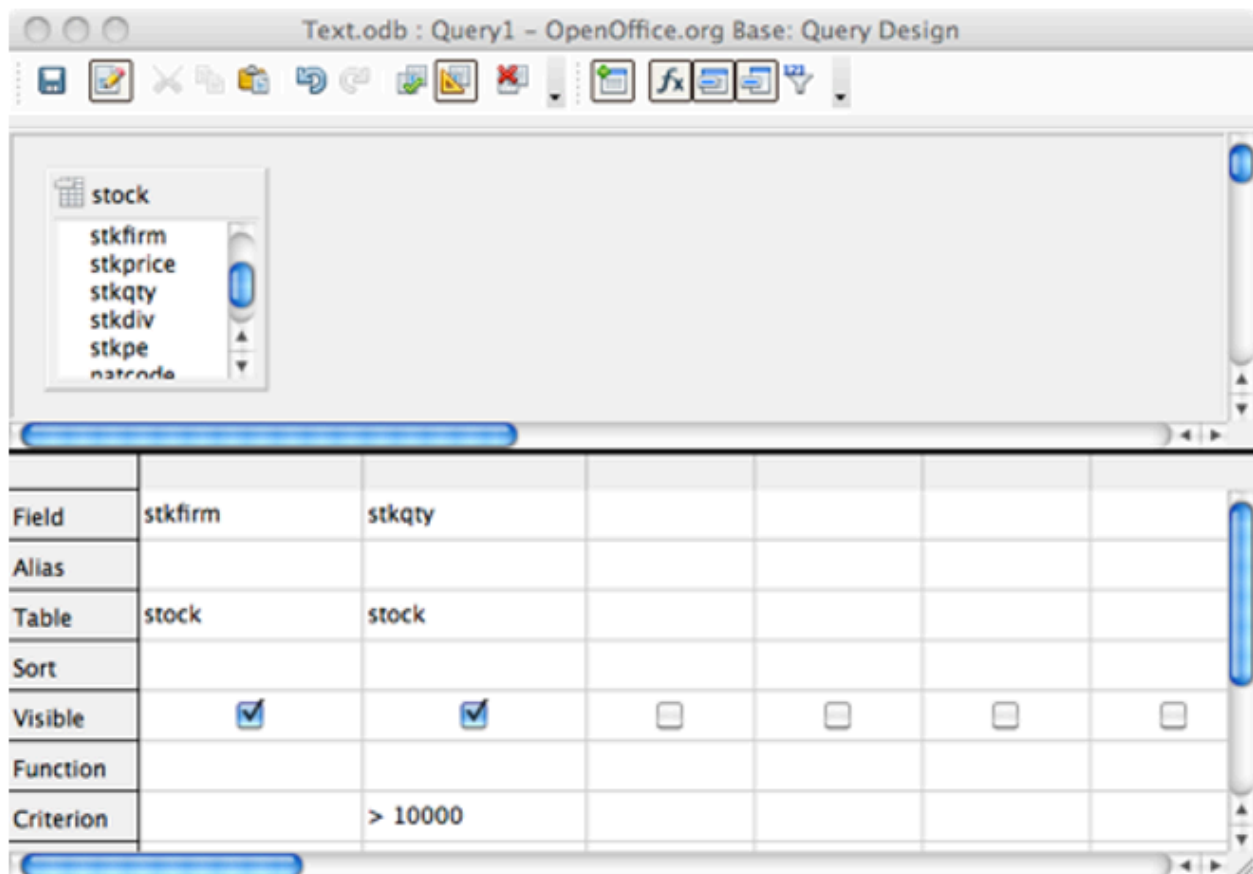
## Manipulation languages

There are four approaches to manipulating relational tables, and you are already familiar with SQL. Query-by-example (QBE), which describes a general class of graphical interfaces to a relational database to make it easier to write queries, is also commonly used. Less widely used manipulation languages are relational algebra and relational calculus. These languages are briefly discussed, with some attention given to relational algebra in the next section and SQL in the following chapter.

**Query-by-example (QBE)** is not a standard, and each vendor implements the idea differently. The general principle is to enable those with limited SQL skills to interrogate a relational database. Thus, the analyst might select tables and columns using drag and drop or by clicking a button. The following screen shot shows the QBE interface to LibreOffice, which is very similar to that of MS Access. It shows selection of the **stock** table and two of its columns, as well as a criterion for **stkqty**. QBE commands are translated to SQL prior to execution, and many systems allow you to view the generated SQL. This can be handy. You

can use QBE to generate a portion of the SQL for a complex query and then edit the generated code to fine-tune the query.

*Query-by-example with LibreOffice*



**Relational algebra** has a set of operations similar to traditional algebra (e.g., add and multiply) for manipulating tables. Although relational algebra can be used to resolve queries, it is seldom employed, because it requires you to specify both what you want and how to get it. That is, you have to specify the operations on each table. This makes relational algebra more difficult to use than SQL, where the focus is on specifying what is wanted.

**Relational calculus** overcomes some of the shortcomings of relational algebra by concentrating on what is required. In other words, there is less need to specify how the query will operate. Relational calculus is classified as a nonprocedural language, because you do not have to be overly concerned with the procedures by which a result is determined.

Unfortunately, relational calculus can be difficult to learn, and as a result, language designers developed **SQL** and **QBE**, which are nonprocedural and more readily mastered. You have already gained some skills in SQL. Although QBE is generally easier to use than SQL, IS professionals need to master SQL for two reasons. *First*, SQL is frequently embedded in other programming languages, a feature not available with QBE. *Second*, it is very difficult, or impossible, to express some queries in QBE (e.g., divide). Because SQL is important, it is the focus of the next chapter.

## Relational algebra

The relational model includes a set of operations, known as relational algebra, for manipulating relations. Relational algebra is a standard for judging data retrieval languages. If a retrieval language, such as SQL, can be used to express every relational algebra operator, it is said to be **relationally complete**.

## Relational algebra operators

Operator	Description
Restrict	Creates a new table from specified rows of an existing table
Project	Creates a new table from specified columns of an existing table
Product	Creates a new table from all the possible combinations of rows of two existing tables
Union	Creates a new table containing rows appearing in one or both tables of two existing tables
Intersection	Creates a new table containing rows appearing in both tables of two existing tables
Difference	Creates a new table containing rows appearing in one table but not in the other of two existing tables
Join	Creates a new table containing all possible combinations of rows of two existing tables satisfying the join condition
Divide	Creates a new table containing $x_i$ such that the pair $(x_i, y_i)$ exists in the first table for every $y_i$ in the second table

There are eight relational algebra operations that can be used with either one or two relations to create a new relation. The assignment operator ( $:=$ ) indicates the name of the new relation. The relational algebra statement to create relation A, the union of relations B and C, is expressed as

**A := B UNION C**

Before we begin our discussion of each of the eight operators, note that the first two, restrict and project, operate on a single relation and the rest require two relations.

### Restrict

Restrict extracts specified rows from a single relation. As the shaded rows in the following figure depict, restrict takes a horizontal slice through a relation.

*Relational operation restrict—a horizontal slice*

W	X	Y	Z

The relational algebra command to create the new relation using restrict is

```
tablename WHERE column1 theta column2
```

or

```
tablename WHERE column1 theta literal
```

where theta can be =, <>, >, >=, <, or <=.

For example, the following relational algebra command creates a new table from stock containing all rows with a nation code of 'US':

```
usstock := stock WHERE natcode = 'US'
```

## Project

Project extracts specified columns from a table. As the following figure shows, project takes a vertical slice through a table.

*Relational operator project—a vertical slice*

W	X	Y	Z

The relational algebra command to create the new relation using project is

```
tablename [columnname, ...]
```

So, in order to create a new table from nation that contains the nation's name and its exchange rate, for example, you would use this relational algebra command:

```
rates := nation\[natname, exchrates\]
```

## Product

Product creates a new relation from all possible combinations of rows in two other relations. It is sometimes called TIMES or MULTIPLY. The relational command to create the product of two tables is

```
tablename1 TIMES tablename2
```

The operation of product is illustrated in the following figure, which shows the result of A TIMES B.

*Relational operator product*

	A		B	
V	W	X	Y	Z
v <sub>1</sub>	w <sub>1</sub>	x <sub>1</sub>	y <sub>1</sub>	z <sub>1</sub>
v <sub>2</sub>	w <sub>2</sub>	x <sub>2</sub>	y <sub>2</sub>	z <sub>2</sub>
v <sub>3</sub>	w <sub>3</sub>			

A TIMES B

V	W	X	Y	Z
v <sub>1</sub>	w <sub>1</sub>	x <sub>1</sub>	y <sub>1</sub>	z <sub>1</sub>
v <sub>1</sub>	w <sub>1</sub>	x <sub>2</sub>	y <sub>2</sub>	z <sub>2</sub>
v <sub>2</sub>	w <sub>2</sub>	x <sub>1</sub>	y <sub>1</sub>	z <sub>1</sub>
v <sub>2</sub>	w <sub>2</sub>	x <sub>2</sub>	y <sub>2</sub>	z <sub>2</sub>
v <sub>3</sub>	w <sub>3</sub>	x <sub>1</sub>	y <sub>1</sub>	z <sub>1</sub>
v <sub>3</sub>	w <sub>3</sub>	x <sub>2</sub>	y <sub>2</sub>	z <sub>2</sub>

## Union

The union of two relations is a new relation containing all rows appearing in one or both relations. The two relations must be **union compatible**, which means they have the same column names, in the same order, and drawn on the same domains. Duplicate rows are automatically eliminated—they must be, or the relational model is no longer satisfied. The relational command to create the union of two tables is

```
tablename1 UNION tablename2
```

Union is illustrated in the following figure. Notice that corresponding columns in tables A and B have the same names. While the sum of the number of rows in relations A and B is five, the union contains four rows, because one row (x<sub>2</sub>, y<sub>2</sub>) is common to both relations.

*Relational operator union*

A		B	
X	Y	X	Y
x <sub>1</sub>	y <sub>1</sub>	x <sub>2</sub>	y <sub>2</sub>
x <sub>2</sub>	y <sub>2</sub>	x <sub>4</sub>	y <sub>4</sub>
x <sub>3</sub>	y <sub>3</sub>		

A UNION B

X	Y
x <sub>1</sub>	y <sub>1</sub>
x <sub>2</sub>	y <sub>2</sub>
x <sub>3</sub>	y <sub>3</sub>
x <sub>4</sub>	y <sub>4</sub>

## Intersect

The intersection of two relations is a new relation containing all rows appearing in both relations. The two relations must be union compatible. The relational command to create the intersection of two tables is

```
tablename1 INTERSECT tablename2
```

The result of A INTERSECT B is one row, because only one row ( $x_2, y_2$ ) is common to both relations A and B.

*Relational operator intersect*

A		B	
X	Y	X	Y
$x_1$	$y_1$	$x_2$	$y_2$
$x_2$	$y_2$	$x_4$	$y_4$
$x_3$	$y_3$		

A INTERSECT B

X	Y
$x_2$	$y_2$

## Difference

The difference between two relations is a new relation containing all rows appearing in the first relation but not in the second. The two relations must be **union compatible**. The relational command to create the difference between two tables is

```
tablename1 MINUS tablename2
```

The result of A MINUS B is two rows (see the following figure). Both of these rows are in relation A but not in relation B. The row containing ( $x_1, y_2$ ) appears in both A and B and thus is not in A MINUS B.

*Relational operator difference*

A		B	
X	Y	X	Y
$x_1$	$y_1$	$x_2$	$y_2$
$x_2$	$y_2$	$x_4$	$y_4$
$x_3$	$y_3$		

A MINUS B

X	Y
$x_1$	$y_1$
$x_3$	$y_3$



## Join

Join creates a new relation from two relations for all combinations of rows satisfying the join condition. The general format of join is

```
tablename1 JOIN tablename2 WHERE tablename1.columnname1 theta
      tablename2.columnname2
```

where theta can be =, <>, >, >=, <, or <=.

The following figure illustrates A JOIN B WHERE W = Z, which is an equijoin because theta is an equals sign. Tables A and B are matched when values in columns W and Z in each relation are equal. The matching columns should be drawn from the same domain. You can also think of join as a product followed by restrict on the resulting relation. So the join can be written

```
(A TIMES B) WHERE W theta Z
```

*Relational operator join*

	A		B	
V	W	X	Y	Z
v <sub>1</sub>	wz <sub>1</sub>	x <sub>1</sub>	y <sub>1</sub>	wz <sub>1</sub>
v <sub>2</sub>	wz <sub>2</sub>	x <sub>2</sub>	y <sub>2</sub>	wz <sub>2</sub>
v <sub>3</sub>	wz <sub>3</sub>			

A EQUIJOIN B

V	W	X	Y	Z
v <sub>1</sub>	wz <sub>1</sub>	x <sub>1</sub>	y <sub>1</sub>	wz <sub>1</sub>
v <sub>3</sub>	wz <sub>3</sub>	x <sub>2</sub>	y <sub>2</sub>	wz <sub>3</sub>

## Divide

Divide is the hardest relational operator to understand. Divide requires that A and B have a set of attributes, in this case Y, that are common to both relations. Conceptually, A divided by B asks the question, “Is there a value in the X column of A (e.g., x<sub>1</sub>) that has a value in the Y column of A for every value of y in the Y column of B?” Look first at B, where the Y column has values y<sub>1</sub> and y<sub>2</sub>. When you examine the X column of A, you find there are rows (x<sub>1</sub>, y<sub>1</sub>) and (x<sub>1</sub>, y<sub>2</sub>). That is, for x<sub>1</sub>, there is a value in the Y column of A for every value of y in the Y column of B. Thus, the result of the division is a new relation with a single row and column containing the value x<sub>1</sub>.

*Relational operator divide*

A		B
X	Y	Y
x <sub>1</sub>	y <sub>1</sub>	y <sub>1</sub>
x <sub>1</sub>	y <sub>3</sub>	y <sub>2</sub>
x <sub>1</sub>	y <sub>2</sub>	
x <sub>2</sub>	y <sub>1</sub>	
x <sub>3</sub>	y <sub>3</sub>	

A DIVIDE B

$$\frac{\overline{X}}{\overline{x_1}}$$

## Querying with relational algebra

A few queries will give you a taste of how you might use relational algebra. To assist in understanding these queries, the answer is expressed, side-by-side, in both relational algebra (on the left) and SQL (on the right).

*List all data in share.*

A very simple report of all columns in the table **share**.

	Relational	SQL
share		SELECT * FROM share

*Report a firm's name and price-earnings ratio.*

A projection of two columns from **share**.

	Relational	SQL
share [shrfirm, shrpe]		SELECT shrfirm, shrpe FROM share

*Get all shares with a price-earnings ratio less than 12.*

A restriction of **share** on column **shrpe**.

	Relational	SQL
share WHERE shrpe < 12		SELECT * FROM share WHERE shrpe < 12

*List details of firms where the share holding is at least 100,000.*

A restriction and projection combined. Notice that the restriction is expressed within parentheses.

	Relational	SQL
(share WHERE shrqty >= 100000) [shrfirm, shrprice, shrqty, shrdiv]		SELECT shrfirm, shrprice, shrqty, shrdiv FROM share

*Find all shares where the PE is 12 or higher and the share holding is less than 10,000.*

Intersect is used with two restrictions to identify shares satisfying both conditions.

	Relational	SQL
(share WHERE shrpe >= 12) INTERSECT		SELECT * FROM share WHERE shrpe >= 12 AND

	Relational	SQL
(share WHERE shrqty < 10000)	shrqty < 10000	

*Skill builder* Write relational algebra statements for the following queries:

Report a firm's name and dividend.

Find the names of firms where the holding is greater than 10,000.

## A primitive set of relational operators

The full set of eight relational operators is not required. As you have already seen, JOIN can be defined in terms of product and restrict. Intersection and divide can also be defined in terms of other commands. Indeed, only five operators are required: restrict, project, product, union, and difference. These five are known as *primitives* because these are the minimal set of relational operators. None of the primitive operators can be defined in terms of the other operators. The following table illustrates how each primitive can be expressed as an SQL command, which implies that SQL is relationally complete.

*Comparison of relational algebra primitive operators and SQL*

Operator   **	Relational   S algebra	QL***
Restrict	A WHERE condition	SELECT * FROM A WHERE condition
Project	A [X]	SELECT X FROM A
Product	A TIMES B	SELECT * from A, B
Union	A UNION B	SELECT * FROM A UNION SELECT * FROM B
Difference	A MINUS B	SELECT * FROM A WHERE NOT EXISTS (SELECT * FROM B WHERE A.X = B.X AND A.Y = B.Y AND ...*)

- Essentially where all columns of A are equal to all columns of B

## A fully relational database

The three components of a relational database system are structures (domains and relations), integrity rules (primary and foreign keys), and a manipulation language (relational algebra). A **fully relational database** system provides complete support for each of these components. Many commercial systems support SQL but do not provide support for domains. Such systems are not fully relational but are relationally complete.

In 1985, Codd established the 12 commandments of relational database systems (see the following table). In addition to providing some insights into Codd's thinking about the management of data, these rules can also be used to judge how well a database fits the relational model. The major impetus for these rules was uncertainty in the marketplace about the meaning of "relational DBMS." Codd's rules are a checklist for establishing the completeness of a relational DBMS. They also provide a short summary of the major features of the relational DBMS.

*Codd's rules for a relational DBMS*

---

## Rules

---

The information rule  
The guaranteed access rule  
Systematic treatment of null values  
Active online catalog of the relational model  
The comprehensive data sublanguage rule  
The view updating rule  
High-level insert, update, and delete  
Physical data independence  
Logical data independence  
Integrity independence  
Distribution independence  
The nonsubversion rule

---

**The information rule** There is only one logical representation of data in a database. All data must appear to be stored as values in a table.

**The guaranteed access rule** Every value in a database must be addressable by specifying its table name, column name, and the primary key of the row in which it is stored.

**Systematic treatment of null values** There must be a distinct representation for unknown or inappropriate data. This must be unique and independent of data type. The DBMS should handle null data in a systematic way. For example, a zero or a blank cannot be used to represent a null. This is one of the more troublesome areas because null can have several meanings (e.g., missing or inappropriate).

**Active online catalog of the relational model** There should be an online catalog that describes the relational model. Authorized users should be able to access this catalog using the DBMS's query language (e.g., SQL).

**The comprehensive data sublanguage rule** There must be a relational language that supports data definition, data manipulation, security and integrity constraints, and transaction processing operations. Furthermore, this language must support both interactive querying and application programming and be expressible in text format. SQL fits these requirements.

**The view updating rule** The DBMS must be able to update any view that is theoretically updatable.

**High-level insert, update, and delete** The system must support set-at-a-time operations. For example, multiple rows must be updatable with a single command.

**Physical data independence** Changes to storage representation or access methods will not affect application programs. For example, application programs should remain unimpaired even if a database is moved to a different storage device or an index is created for a table.

**Logical data independence** Information-preserving changes to base tables will not affect application programs. For instance, no applications should be affected when a new table is added to a database.

**Integrity independence** Integrity constraints should be part of a database's definition rather than embedded within application programs. It must be possible to change these constraints without affecting any existing application programs.

**Distribution independence** Introduction of a distributed DBMS or redistribution of existing distributed data should have no impact on existing applications.

**The nonsubversion rule** It must not be possible to use a record-at-a-time interface to subvert security or integrity constraints. You should not be able, for example, to write a Java program with embedded SQL commands to bypass security features.

Codd issued an additional higher-level rule, rule 0, which states that a relational DBMS must be able to manage databases entirely through its relational capacities. In other words, a DBMS is either totally relational or it is not relational.

## Summary

The relational model developed as a result of recognized shortcomings of hierarchical and network DBMSs. Codd created a strong theoretical base for the relational model. Three objectives drove relational database research: data independence, communicability, and set processing. The relational model has domain structures, integrity rules, and operators used to retrieve, derive, or modify data. A domain is a set of values all of the same data type. The practical value of a domain is to define what comparisons are permissible. A relation is a table of  $n$  columns and  $m$  rows. The cardinality of a relation is its number of rows. The degree of a relation is the number of columns.

A relational database is a collection of relations. The distinguishing feature of the relational model is that there are no explicit linkages between tables. A relation's primary key is its unique identifier. When there are multiple candidates for the primary key, one is chosen as the primary key and the remainder are known as alternate keys. The foreign key is the way relationships are represented and can be thought of as the glue that binds a set of tables together to form a relational database. The purpose of the entity integrity rule is to ensure that each entity described by a relation is identifiable in some way. The referential integrity rule ensures that you cannot define a foreign key without first defining its matching primary key.

The relational model includes a set of operations, known as relational algebra, for manipulating relations. There are eight operations that can be used with either one or two relations to create a new relation. These operations are restrict, project, product, union, intersect, difference, join, and divide. Only five operators, known as primitives, are required to define all eight relational operations. An SQL statement can be translated to relational algebra and vice versa. If a retrieval language can be used to express every relational algebra operator, it is said to be relationally complete. A fully relational database system provides complete support for domains, integrity rules, and a manipulation language. Codd set forth 12 rules that can be used to judge how well a database fits the relational model. He also added rule 0—a DBMS is either totally relational or it is not relational.

## Key terms and concepts

---

Alternate key	Join
Candidate key	Logical data independence
Cardinality	Nonsubversion rule
Catalog	Null
Communicability objective	Operators

---

Data independence	Physical data independence
Data structures	Primary key
Data sublanguage rule	Product
Degree	Project
Difference	Query-by-example (QBE)
Distribution independence	Referential integrity
Divide	Relational algebra
Domain	Relational calculus
Entity integrity	Set processing objective
Foreign key	Relational database
Fully relational database	Relational model
Guaranteed access rule	Relationally complete
Information rule	Relations
Integrity independence	Restrict
Integrity rules	Union
Intersect	View updating rule

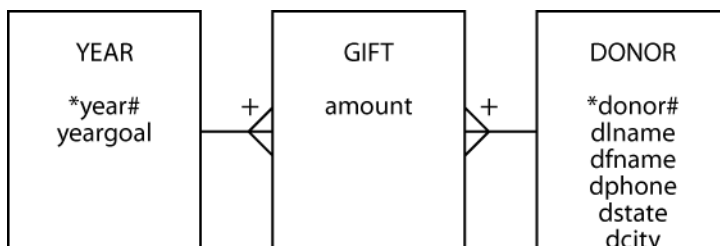
---

## References and additional readings

Codd, E. F. 1982. Relational database: A practical foundation for productivity. *Communications of the ACM* 25 (2):109–117.

## Exercises

1. What reasons does Codd give for adopting the relational model?
2. What are the major components of the relational model?
3. What is a domain? Why is it useful?
4. What are the meanings of cardinality and degree?
5. What is a simple relational database?
6. What is the difference between a primary key and an alternate key?
7. Why do we need an entity integrity rule and a referential integrity rule?
8. What is the meaning of “union compatible”? Which relational operations require union compatibility?
9. What is meant by the term “a primitive set of operations”?
10. What is a fully relational database?
11. How well does OpenOffice’s database satisfy Codd’s rules.
12. Use relational algebra with the given data model to solve the following queries:



- a. List all donors.
- b. List the first and last names of all donors.
- c. List the phone numbers of donors Hays and Jeffs.
- d. List the amount given by each donor for each year.
- e. List the donors who have made a donation every year.
- f. List the names of donors who live in Georgia or North Carolina.
- g. List the names of donors whose last name is Watson and who live in Athens, GA.

## Section 3 Advanced Data Management

*Advancement only comes with habitually doing more than you are asked.*

Gary Ryan Blair.<sup>18</sup>

The wiring of the world has given us ubiquitous networks and broadened the scope of issues that data management must now embrace. In everyday life, you might use a variety of networks (e.g., the Internet, 4G, WiFi, and bluetooth) to gain access to information wherever you might be and whatever time it is. As a result, data managers need to be concerned with both the **spatial and temporal** dimensions of data. In a highly connected world, massive amounts of data are exchanged every minute between computers to enable a high level of global integration in economic and social activity. **XML** has emerged as the foundation for data exchange across many industries. It is a core technology for global economic development. On the social side, every day people generate millions of messages, photos, and videos that fuel services such as Twitter, Flickr, and YouTube. Many organizations are interested in analyzing these data streams to learn about social trends, customers' opinions, and entrepreneurial opportunities. Organizations need skills in collecting, processing, and interpreting the myriad data flows that intersect with their everyday business. **Organizational or business intelligence** is the general term for describing an enterprise's efforts to collect, store, process, and interpret data from internal and external sources. It is the first stage of data-driven decision making. Once data have been captured and stored in an organizational repository, there are several techniques that can be applied.

In a world awash with data, **visualization** has become increasingly important for enabling executives to make sense of the business environment, to identify problems, and highlight potential new directions. **Text mining** is a popular tool for trying to make sense of data streams emanating from tweets and blogs. The many new sources of data and their high growth rate have made it more difficult to support real time analysis of the torrents of data that might contain valuable insights for an organization's managers. Fortunately, **Hadoop distributed file system (HDFS)** and **cluster computing** methods are a breakthrough in storing and processing data that enable faster processing at lower cost. **Dashboards** are widely used for presenting key information. Furthermore, the open source statistics and graphical package, **R**, provides a common foundation for handling text mining, data visualization, HDFS, and cluster computing. It has become another component of the data manager's toolkit.

The section covers the following topics.

- Spatial and temporal data management
- XML
- Organizational intelligence
- Introduction to R
- Data visualization

---

<sup>18</sup><http://www.garyryanblair.com>

- Text mining
- Cluster Computing
- Dashboards

## 10 XML: Managing Data Exchange

*Words can have no single fixed meaning. Like wayward electrons, they can spin away from their initial orbit and enter a wider magnetic field. No one owns them or has a proprietary right to dictate how they will be used.*

David Lehman, End of the Word, 1991

### Learning objectives

Students completing this chapter will be able to

- define the purpose of XML;
- create an XML schema;
- code data in XML format;
- create an XML stylesheet;
- discuss data management options for XML documents.

### Introduction

There are four central problems in data management: capture, storage, retrieval, and exchange. The focus for most of this book has been on storage (i.e., data modeling) and retrieval (i.e., SQL). Now it is time to consider capture and exchange. Capture has always been an important issue, and the guiding principle is to capture data once in the cheapest possible manner.

### SGML

The Standard Generalized Markup Language (SGML) was designed to reduce the cost and increase the efficiency of document management. Its child, XML, has essentially replaced SGML. For example, the second edition of the *Oxford English Dictionary* was specified in SGML, and the third edition is stored in XML format.<sup>19</sup>

A markup language embeds information about a document in the text. In the following table, the markup tags indicate that the text contains CD liner notes. Note also that the titles and identifiers of the mentioned CDs are explicitly identified.

*Markup language*

```
<cdliner>This uniquely creative collaboration between Miles Davis and Gil Evans has already resulted in
```

<sup>19</sup>Cowlshaw, M. F. (1987). Lexx—a programmable structured editor. *IBM Journal of Research and Development*, 31(1), 73-80.



SGML is an International Standard (ISO 8879) that defines the structure of documents. It is a vendor-independent language that supports cross-system portability and publication for all media. Developed in 1986 to manage software documentation, SGML was widely accepted as the markup language for a number of information-intensive industries. As a metalanguage, SGML is the mother of both HTML and XML.

SGML illustrates four major advantages a markup language provides for data management:

- **Reuse:** Information can be created once and reused over and over. By storing critical documents in markup format, firms do not need to duplicate efforts when there are changes to documents. For example, a firm might store all its legal contracts in SGML.
- **Flexibility:** SGML documents can be published in any medium for a wide variety of audiences. Because SGML is content-oriented, presentation decisions are delayed until the output format is known. Thus, the same content could be printed, presented on the Web in HTML, or written to a DVD as a PDF.
- **Revision:** SGML enhances control over revision and enables version control. When stored in an SGML database, original data are archived alongside any changes. That means you know exactly what the original document contained and what changes were made.
- **Format independence:** SGML files are stored as text and can be read by many programs on all operating systems. Thus, it preserves textual information independent of how and when it is presented. SGML protects a firm's investment in documentation for the long term. Because it is now possible to display documentation using multiple media (e.g., Web and iPad), firms have become sensitized to the need to store documents in a single, independent manner that can then be converted for display by a particular medium.

SGML's power is derived from its recording of both text and the meaning of that text. A short section of SGML demonstrates clearly the features and strength of SGML. The tags surrounding a chunk of text describe its meaning and thus support presentation and retrieval. For example, the pair of tags `<title>` and `</title>` surrounding "XML: Managing Data Exchange" indicates that it is the chapter title.

*SGML code*

```
<chapter>
<no>18</no>
<title>XML: Managing Data Exchange</title>
<section>
<quote><emph type = '2'>Words can have no single fixed meaning. Like wayward electrons, they can spin a
</quote>
</section>
</chapter>
```

Taking this piece of SGML, it is possible, using an appropriate stylesheet, to create a print version where the title of the chapter is displayed in Times, 16 point, bold, or a HTML version where the title is displayed in red, Georgia, 14 point, italics. Furthermore, the database in which this text is stored can be searched for any chapters that contain "Exchange" in their title.

Now, consider the case where the text is stored as HTML. How do you, with complete certainty, identify the chapter title? Do you extract all text contained by `<h1>` and `</h1>` tags? You will then retrieve "18" as a possible chapter title. What happens if there is other text displayed using `<h1>` and `</h1>` tags? The problem with HTML is that it defines presentation and has very little meaning. A similar problem exists for documents prepared with a word processor.

*HTML code*

```

<html>
<body>
<h1><b>18 </b></h1>
<h1><b>XML: Managing Data Exchange</b></h1>
<p><i>Words can have no single fixed meaning. Like wayward electrons, they can spin away from their ini
</body>
</html>

```

By using embedded tags to record meaning, SGML makes a document platform-independent and greatly improves the effectiveness of searching. Despite its many advantages, there are some features of SGML that make implementation difficult and also limit the ability to create tools for information management and exchange. As a result, XML, a derivative of SGML, was developed.

## XML

Extensible Markup Language (XML), a language designed to make information self-describing, retains the core ideas of SGML. You can think of XML as SGML for electronic and mobile commerce. Since the definition of XML was completed in early 1998 by the World Wide Web Consortium (W3C), the standard has spread rapidly because it solves a critical data management problem. XML is a metalanguage—a language to generate languages.

Despite having the same parent, there are major differences between XML and HTML.

*XML vs. HTML*

XML	HTML
Structured text	Formatted text
User-definable structure (extensible)	Predefined formats (not extensible)
Context-sensitive retrieval	Limited retrieval
Greater hypertext linking	Limited hypertext linking

HTML, an electronic-publishing language, describes how a Web browser should display text and images on a computer screen. It tells the browser nothing about the meaning of the data. For example, the browser does not know whether a piece of text represents a price, a product code, or a delivery date. Humans infer meaning from the context (e.g., August 8, 2012, is recognized as a date). Given the explosive growth of the Web, HTML clearly works well enough for exchanging data between computers and humans. It does not, however, work for exchanging data between computers, because computers are not smart enough to deduce meaning from context.

Successful data exchange requires that the meaning of the exchanged data be readily determined by a computer. The XML solution is to embed tags in a file to describe the data (e.g., insert tags into an order to indicate attributes such as price, size, quantity, and color). A browser, or program for that matter, can then recognize this document as a customer order. Consequently, it can do far more than just display the price. For example, it can convert all prices to another currency. More importantly, the data can be exchanged between computers and understood by the receiving system.

XML consists of rules (that anyone can follow to create a markup language (e.g., a markup language for financial data such as XBRL). Hence, the “eXtensible” in the XML name, indicating that the language can be easily extended to include new tags. In contrast, HTML is not extensible and its set of tags is fixed, which is one of the major reasons why HTML is easy to learn. The XML rules ensure that a type of computer program known as a parser can process any extension or addition of new tags.

*XML rules*

- Elements must have both an opening and a closing tag.
- Elements must follow a strict hierarchy with only one root element.
- Elements must not overlap other elements.
- Element names must obey XML naming conventions.
- XML is case sensitive.

Consider the credit card company that wants to send you your latest statement via the Internet so that you can load it into your financial management program. Since this is a common problem for credit card companies and financial software authors, these industry groups have combined to create Open Financial Exchange (OFX),<sup>20</sup> a language for the exchange of financial data across the Internet.

XML has a small number of rules. Tags always come in pairs, as in HTML. A pair of tags surrounds each piece of data (e.g., `<price>89.12</price>`) to indicate its meaning, whereas in HTML, they indicate how the data are presented. Tag pairs can be nested inside one another to multiple levels, which effectively creates a tree or hierarchical structure. Because XML uses Unicode (see the discussion in Chapter 11), it enables the exchange of information not only between different computer systems, but also across language boundaries.

The differences between HTML and XML are captured in the following examples for each markup language. Note that in the following table, HTML incorporates formatting instructions (i.e., the course code is bold), whereas XML describes the meaning of the data.

#### *Comparison of HTML and XML coding*

HTML	XML
<code>&lt;p&gt;&lt;b&gt;MIST7600&lt;/b&gt;</code>	<code>&lt;course&gt;</code>
<code>Data Management&lt;br&gt;</code>	<code>&lt;code&gt;MIST7600&lt;/code&gt;</code>
<code>3 credit hours&lt;/p&gt;</code>	<code>&lt;title&gt;Data Management&lt;/title&gt;</code>
<code>&lt;/course&gt;</code>	<code>&lt;credit&gt;3&lt;/credit&gt;</code>

XML enables a shift of processing from the server to the browser. At present, most processing has to be done by the server because that is where knowledge about the data is stored. The browser knows nothing about the data and therefore can only present but not process. However, when XML is implemented, the browser can take on processing that previously had to be handled by the server.

Imagine that you are selecting a shirt from a mail-order catalog. The merchant's Web server sends you data on 20 shirts (100 Kbytes of text and images) with prices in U.S. dollars. If you want to see the prices in euros, the calculation will be done by the server, and the full details for the 20 shirts retransmitted (i.e., another 100 Kbytes are sent from the server to the browser). However, once XML is in place, all that needs to be sent from the server to the browser is the conversion rate of U.S. dollars to euros and a program to compute the conversion at the browser end. In most cases, less data will be transmitted between a server and browser when XML is in place. Consequently, widespread adoption of XML will reduce network traffic.

#### *Execution of HTML and XML code*

HTML	XML
Retrieve shirt data with prices in USD.	Retrieve shirt data with prices in USD.
Retrieve shirt data with prices in EUR.	Retrieve conversion rate of USD to EUR.
	Retrieve Java program to convert currencies.
	Compute prices in EUR.

<sup>20</sup><http://www.ofx.net>

XML can also make searching more efficient and effective. At present, search engines look for matching text strings, and consequently return many links that are completely irrelevant. For instance, if you are searching for details on the Nomad speaker system, and specify “nomad” as the sought text string, you will get links to many items that are of no interest (e.g., The Fabulous Nomads Surf Band). Searching will be more precise when you can specify that you are looking for a product name that includes the text “nomad.” The search engine can then confine its attention to text contained within the tags `<productname>` and `</productname>`, assuming these tags are the XML standard for representing product names.

The major expected gains from the introduction of XML are

- Store once and format many ways—Data stored in XML format can be extracted and reformatted for multiple presentation styles (e.g., printed report, DVD).
- Hardware and software independence—One format is valid for all systems. Capture once and exchange many times—Data are captured as close to the source as possible and never again (i.e., no rekeying).
- Accelerated targeted searching—Searches are more precise and faster because they use XML tags.
- Less network congestion—The processing load shifts from the server to the browser.

## XML language design

XML lets developers design application-specific vocabularies. To create a new language, designers must agree on three things:

- The allowable tags
- The rules for nesting tagged elements
- Which tagged elements can be processed

The first two, the language’s vocabulary and structure, are typically defined in an XML schema. Developers use the XML schema to understand the meaning of tags so they can write software to process an XML file.

XML tags describe meaning, independent of the display medium. An XML stylesheet, another set of rules, defines how an XML file is automatically formatted for various devices. This set of rules is called an Extensible Stylesheet Language (XSL). Stylesheets allow data to be rendered in a variety of ways, such as Braille or audio for visually impaired people.

## XML schema

An XML schema (or just schema for brevity) is an XML file associated with an XML document that informs an application how to interpret markup tags and valid formats for tags. The advantage of a schema is that it leads to standardization. Consistently named and defined tags create conformity and support organizational efficiency. They avoid the confusion and loss of time when the meaning of data is not clear. Also, when validation information is built into a schema, some errors are detected before data are exchanged.

XML does not require the creation of a schema. If a document is well formed, XML will interpret it correctly. A well-formed document follows XML syntax and has tags that are correctly nested.

A schema is a very strict specification, and any errors will be detected when parsing. A schema defines:

- The names and contents of all elements that are permissible in a certain document
- The structure of the document

- How often an element may appear
- The order in which the elements must appear
- The type of data the element can contain

**DOM** The **Document Object Model** (DOM) is the model underlying XML. It is based on a tree (i.e., it directly supports one-to-one and one-to-many, but not many-to-many relationships). A document is modeled as a hierarchical collection of nodes that have parent/child relationships. The node is the primary object and can be of different types (such as document, element, attribute, text). Each document has a single document node, which has no parent, and zero or more children that are element nodes. It is a good practice to create a visual model of the XML document and then convert this to a schema, which is XML's formal representation of the DOM.

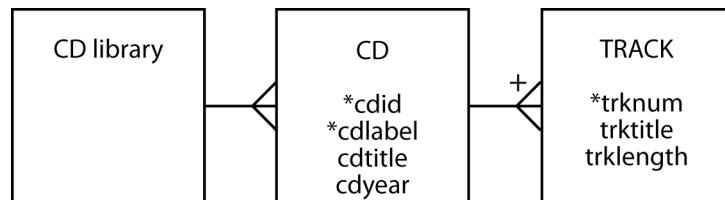
At this point, an example is the best way to demonstrate XML, schema, and DOM concepts. We will use the familiar CD problem that was introduced in Chapter 3. In keeping with the style of this text, we define a minimal amount of XML to get you started, and then more features are added once you have mastered the basics.

### CD library case

The CD library case gradually develops, over several chapters, a data model for recording details of a CD collection, culminating in the model at the end of Chapter 6. Unfortunately, we cannot quickly convert this final model to an XML document model, because a DOM is based on a tree model. Thus, we must start afresh.

The model, in this case, is based on the observation that a CD library has many CDs, and a CD has many tracks.

*CD library tree data model*



A model is then mapped into a schema using the following procedure.

- Each entity becomes a complex element type.
- Each data model attribute becomes a simple element type.
- The one-to-many (1:m) relationship is recorded as a sequence.

The schema for the CD library follows. For convenience of exposition, the source code lines have been numbered, but these numbers are not part of a schema.<sup>21</sup>

*Schema for CD library (cdlib.xsd)*

<sup>21</sup>You should use an XML editor for creating XML files. I have not found a good open source product. Among the commercial products, my preference is for Oxygen, which you can get for a 30 days trial.

```

<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd='http://www.w3.org/2001/XMLSchema'>
  <!--CD library-->
    <xsd:element name="cdlibrary">
      <xsd:complexType>
        <xsd:sequence>
          <xsd:element name="cd" type="cdType" minOccurs="1"
            maxOccurs="unbounded"/>
        </xsd:sequence>
      </xsd:complexType>
    </xsd:element>
  <!--CD-->
    <xsd:complexType name="cdType">
      <xsd:sequence>
        <xsd:element name="cdid" type="xsd:string"/>
        <xsd:element name="cdlabel" type="xsd:string"/>
        <xsd:element name="cdtitle" type="xsd:string"/>
        <xsd:element name="cdyear" type="xsd:integer"/>
        <xsd:element name="track" type="trackType" minOccurs="1"
          maxOccurs="unbounded"/>
      </xsd:sequence>
    </xsd:complexType>
  <!--Track-->
    <xsd:complexType name="trackType">
      <xsd:sequence>
        <xsd:element name="trknum" type="xsd:integer"/>
        <xsd:element name="trktitle" type="xsd:string"/>
        <xsd:element name="trklen" type="xsd:time"/>
      </xsd:sequence>
    </xsd:complexType>
</xsd:schema>

```

There are several things to observe about the schema.

- All XML documents begin with an XML declaration {1}.<sup>22</sup> The encoding attribute (i.e., encoding="UTF-8") specifies what form of Unicode is used (in this case the 8-bit form).
- The XSD Schema namespace<sup>23</sup> is declared {2}.
- Comments are placed inside the tag pair <!-- and --> {3}.
  - The CD library is defined {4–10} as a complex element type, which essentially means that it can have embedded elements, which are a sequence of CDs in this case.
  - A sequence is a series of child elements embedded in a parent, as illustrated by a CD library containing a sequence of CDs {7}, and a CD containing elements of CD identifier, label, and so forth {15–20}. The order of a sequence must be maintained by any XML document based on the schema.
  - A sequence can have a specified range of elements. In this case, there must be at least one CD (minOccurs="1") but there is no upper limit (maxOccurs="unbounded") on how many CDs there can be in the library {7}.

<sup>22</sup>In this chapter, numbers in {} refer to line numbers in the corresponding XML code.

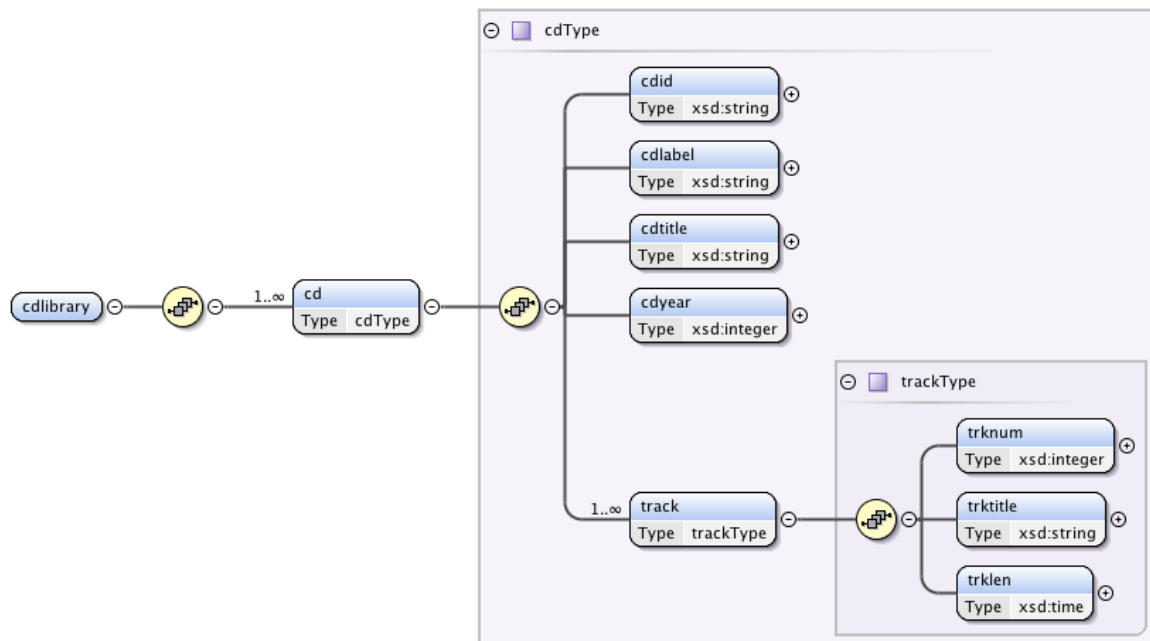
<sup>23</sup>A namespace is a collection of valid names of attributes, types, and elements for a schema. Think of a namespace as a dictionary.

- An element that has a child (e.g., cdlibrary, which is at the 1 end of a 1:m) or possesses attributes (e.g., track) is termed a complex element type.
- A CD is represented by a complex element type {13–20}, and has the name cdType {13}.
- The element cd is defined by specifying the name of the complex type (i.e., cdType) containing its specification {7}.
- A track is represented by a complex type because it contains elements of track number, title, and length {24–30}. The name of this complex type is trackType {24}.
- Notice the reference within the definition of cd to the complex type trackType, used to specify the element track {19}.
- Simple types (e.g., cdid and cdyear) do not contain any elements, and thus the type of data they store must be defined. Thus, cdid is a text string and cdyear is an integer.

The purpose of a schema is to define the contents and structure of an XML file. It is also used to verify that an XML file has a valid structure and that all elements in the XML file are defined in the schema.

If you use an editor, you can possibly create a visual view of the schema.

*A visual depiction of a schema as created by Oxygen*



Some common data types are shown in the following table. The meaning is obvious in most cases for those familiar with SQL, except for uriReference. A Uniform Resource Identifier (URI) is a generalization of the URL concept.

*Some common data types*

Data type
string
boolean
anyURI
decimal
float
integer
time

Data type
date

We can now use the recently defined CDlibrary schema to describe a small CD library containing the CD information given in the following table.

*Data for a small CD library*

The XML for describing the CD library follows. There are several things to observe:

- All XML documents begin with an XML declaration.
- The declaration immediately following the XML declaration identifies the root element of the document (i.e., cdlibrary) and the schema (i.e., cdlib.xsd).
- Details of a CD are enclosed by the tags <cd> and </cd>.
- Details of a track are enclosed by the tags <track> and </track>.

*XML for describing a CD (cdlib.xml)*

```
<?xml version="1.0" encoding="UTF-8"?>
<cdlibrary xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="cdlib.xsd">
  <cd>
    <cdid>A2 1325</cdid>
    <cdlabel>Atlantic</cdlabel>
    <cdtitle>Pyramid</cdtitle>
    <cdyear>1960</cdyear>
    <track>
      <trknum>1</trknum>
      <trktitle>Vendome</trktitle>
      <trklen>00:02:30</trklen>
    </track>
    <track>
      <trknum>2</trknum>
      <trktitle>Pyramid</trktitle>
      <trklen>00:10:46</trklen>
    </track>
  </cd>
  <cd>
    <cdid>D136705</cdid>
    <cdlabel>Verve</cdlabel>
    <cdtitle>Ella Fitzgerald</cdtitle>
    <cdyear>2000</cdyear>
    <track>
      <trknum>1</trknum>
      <trktitle>A tisket, a tasket</trktitle>
      <trklen>00:02:37</trklen>
    </track>
    <track>
      <trknum>2</trknum>
      <trktitle>Vote for Mr. Rhythm</trktitle>
      <trklen>00:02:25</trklen>
    </track>
  </cd>
</cdlibrary>
```



```

        <trknum>3</trknum>
        <trktitle>Betcha nickel</trktitle>
        <trklen>00:02:52</trklen>
    </track>
</cd>
</cdlibrary>

```

As you now realize, the definition of an XML document is relatively straightforward. It is a bit tedious with all the typing of tags to surround each data element. Fortunately, there are XML editors that relieve this tedium.

### Skill builder

1. Use the Firefox browser<sup>24</sup> to access this book's Web site, link to the Support > XML section, and click on `customerpayments.xml`. You will see how this browser displays XML. Investigate what happens when you click on the '-' and '+' signs next to some entries.
2. Again, using Firefox, save the displayed XML code (Save Page As ...) as `customerpayments.xml`, and open it in a text editor.
3. Now, add details of the customer and payment data displayed in the following table to the beginning of the XML file. Open the saved file with Firefox, and verify your work.

*Customer and payment data*

AA Souvenirs		
Yallingup		
Australia		
Check	Amount	Date
QP45901	9387.45	2005-03-16
AG9984	3718.67	2005-07-24

**XSL** As you now know from the prior exercise, the browser display of XML is not particularly useful. What is missing is a stylesheet that tells the browser how to display an XML file. The eXtensible Stylesheet Language (XSL) is used for defining the rendering of an XML file. An XSL document defines the rules for presenting an XML document's data. XSL is an application of XML, and an XSL file is also an XML file.

The power of XSL is demonstrated by applying the stylesheet that follows to the preceding XML.

*Result of applying a stylesheet to CD library data*

<sup>24</sup>Overall, Firefox seems to do the best job of displaying xml files.

## Complete List of Songs

Pyramid, Atlantic, 1960.5 [A2 1325]

1	Vendome	00:02:30
2	Pyramid	00:10:46

Ella Fitzgerald, Verve, 2000 [D136705]

1	A tisket, a tasket	00:02:37
2	Vote for Mr. Rhythm	00:02:25
3	Betcha nickel	00:02:52

*Stylesheet for displaying an XML file of CD data (cdlib.xsl)*

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output encoding="UTF-8" indent="yes" method="html" />
  <xsl:template match="/">
    <html>
      <head>
        <title> Complete List of Songs </title>
      </head>
      <body>
        <h1> Complete List of Songs </h1>
        <xsl:apply-templates select="cdlibrary" />
      </body>
    </html>
  </xsl:template>
  <xsl:template match="cdlibrary">
    <xsl:for-each select="cd">
      <br/>
      <font color="maroon">
        <xsl:value-of select="cdtitle" />
        ,
        <xsl:value-of select="cdlabel" />
        ,
        <xsl:value-of select="cdyear" />
        [
        <xsl:value-of select="cdid" />
        ] </font>
      <br/>
      <table>
        <xsl:for-each select="track">
          <tr>
            <td align="left">
              <xsl:value-of select="trknum" />
            </td>
            <td>
              <xsl:value-of select="trktitle" />
            </td>
            <td align="center">
```

```

                <xsl:value-of select="trklen" />
            </td>
        </tr>
    </xsl:for-each>
</table>
<br/>
</xsl:for-each>
</xsl:template>
</xsl:stylesheet>

```

To use a stylesheet with an XML file, you must add a line of code to point to the stylesheet file. In this case, you add the following:

```
<?xml-stylesheet type="text/xsl" href="cdlib.xsl" media="screen"?>
```

as the second line of `cdlib.xml` (i.e., it appears before `<cdlibrary ... >`). The added line of code points to `cdlib.xsl` as the stylesheet. This means that when the browser loads `cdlib.xml`, it uses the contents of `cdlib.xsl` to determine how to render the contents of `cdlib.xml`.

We now need to examine the contents of `cdlib.xsl` so that you can learn some basics of creating XSL commands. You will soon notice that all XSL commands are preceded by `xsl`:

- Tell the browser it is processing an XML file {1}
- Specify that the file is a stylesheet {2}
- Specify a template, which identifies which elements should be processed and how they are processed. The match attribute {4} indicates the template applies to the source node. Process the template {11} defined in the file {15–45}. A stylesheet can specify multiple templates to produce different reports from the same XML input.
- Specify a template to be applied when the XSL processor encounters the `<cdlibrary>` node {15}.
- Create an outer loop for processing each CD {16–44}.
- Define the values to be reported for each CD (i.e., title, label, year, and id) {19, 21, 23, 25}. The respective XSL commands select the values. For example, `<xsl:value-of select="cdtitle" />` specifies selection of `cdtitle`.
- Create an inner loop for processing the tracks on a particular CD {29–41}.
- Present the track data in tabular form using HTML table commands interspersed with XSL {28–42}.

### Skill builder

1. Use the Firefox browser to access this book's Web site, navigate to the XML page, and download `cdlib.xml` and `cdlib.xsl` to a directory or folder on your machine. Use Save Page As ... for downloading.
2. Using a text editor, change the saved copy of `cdlib.xml` by inserting the following as the second line:  

```
<?xml-stylesheet type="text/xsl" href="cdlib.xsl" media="screen"?>
```
3. Save the edited file in the same directory or folder as `cdlib.xsl`. Open the saved XML file with Firefox.

## Converting XML

There are occasions when there is a need to convert an XML file:

- **Transformation:** conversion from one XML vocabulary to another (e.g., between financial languages FPML and finML)
- **Manipulation:** reordering, filtering, or sorting parts of a document
- **Rendering in another language:** rendering the XML file using another format

You have already seen how XSL can be used to transform XML for rendering as HTML. The original XSL has been split into three languages:

- XSLT for transformation and manipulation
- XSLT for rendition
- XPath for accessing the structure of an XML file

For a data management course, this is as far as you need to go with learning about XSL. Just remember that you have only touched the surface. To become proficient in XML, you will need an entire course on the topic.

## XPath for navigating an XML document

XPath is a navigation language for an XML document. It defines how to select nodes or sets of nodes in a document. The first step to understanding XPath is to know about the different types of nodes. In the following XML document, the document node is `<cdlibrary> {1}`, `<trktitle>Vendome</trktitle> {9}` is an example of an element node, and `<track length="00:02:30"> {7}` is an instance of an attribute node.

*An XML document*

```
<cdlibrary>
  <cd>
    <cdid>A2 1325</cdid>
    <cdlabel>Atlantic</cdlabel>
    <cdtitle>Pyramid</cdtitle>
    <cdyear>1960</cdyear>
    <track length="00:02:30">
      <trknum>1</trknum>
      <trktitle>Vendome</trktitle>
    </track>
    <track length="00:10:46">
      <trknum>2</trknum>
      <trktitle>Pyramid</trktitle>
    </track>
  </cd>
</cdlibrary>
```

## A family of nodes

Each element and attribute has one **parent node**. In the preceding XML document, `cd` is the parent of `cdid`, `cdlabel`, `cdyear`, and `track`. Element nodes may have zero or more **children nodes**. Thus `cdid`, `cdlabel`, `cdyear`, and `track` are the children of `cd`. **Ancestor nodes** are the parent, parent's parent, and so forth of a node. For example, `cd` and `cdlibrary` are ancestors of `cdtitle`. Similarly, we have **descendant nodes**, which are the children, children's children, and so on of a node. The descendants of `cd` include `cdid`, `cdtitle`, `track`, `trknum`, and `trktitle`. Finally, we have **sibling nodes**, which are nodes that share a parent. In the sample document, `cdid`, `cdlabel`, `cdyear`, and `track` are siblings.

## Navigation examples

The examples in the following table give you an idea of how you can use XPath to extract data from an XML document. Our preference is to answer such queries using SQL, but if your data are in XML format, then XPath provides a means of interrogating the file.

### *XPath examples*

Example	Result
/cdlibrary/cd[1]	Selects the first CD
//trktitle	Selects all the titles of all tracks
/cdlibrary/cd[last()-1]	Selects the second last CD
/cdlibrary/cd[last()]/track[last()]/trklen T	he length of the last track on the last CD
/cdlibrary/cd[cdyear=1960]	Selects all CDs released in 1960
/cdlibrary/cd[cdyear>1950]/cdtitle	Titles of CDs released after 1950

## XQuery for querying an XML document

XQuery is a query language for XML, and thus it plays a similar role that SQL plays for a relational database. It is used for finding and extracting elements and attributes of an XML document. It builds on XPath's method for specifying navigation through the elements of an XML document. As well as being used for querying, XQuery can also be used for transforming XML to XHTML.

The first step is to specify the location of the XML file. In this case, we will use the cdlib.xml file, which is stored on the web site for this book. Using XPath notation it is relatively straightforward to list some values in an XML file

*List the titles of CDs.*

```
doc("http://www.richardtwatson.com/xml/cdlib.xml")/cdlibrary/cd/cdtitle
<?xml version="1.0" encoding="UTF-8"?>
<cdtitle>Pyramid</cdtitle>
<cdtitle>Ella Fitzgerald</cdtitle>
```

You can also use an XPath expression to select particular information.

*List the titles of CDs released under the Verve label*

```
doc("http://www.richardtwatson.com/xml/cdlib.xml")/cdlibrary/cd[cdlabel='Verve']/cdtitle
<?xml version="1.0" encoding="UTF-8"?>
<cdtitle>Ella Fitzgerald</cdtitle>
```

XQuery commands can also be written using an SQL like structure, which is called FLWOR, an easy way to remember 'For, Let, Where, Order by, Return.' Here is an example.

*List the titles of tracks longer than 5 minutes*

```
for $x in doc("http://www.richardtwatson.com/xml/cdlib.xml")/cdlibrary/cd/track
where $x/'track length' > "00:05:00"
order by $x/'trktitle'
return $x
```

```
<?xml version="1.0" encoding="UTF-8"?>
<track>
  <trknum>2</trknum>
  <trktitle>Pyramid</trktitle>
  <trklen>00:10:46</trklen>
</track>
```

If you want to just report the data without the tags, use `return data($x)`.

## XML and databases

XML is more than a document-processing technology. It is also a powerful tool for data management. For database developers, XML can be used to facilitate middle-tier data integration and schemas. Most of the major DBMS producers have XML-centric extensions for their product lines.

Many XML documents are stored for the long term, because they are an important repository of organizational memory. A data exchange language, XML is a means of moving data between databases, which means a need for tools for exporting and importing XML.

XML documents can be stored in the same format as you would store a word processing or HTML file: You just place them in an appropriately named folder. File systems, however, have limitations that become particularly apparent when a large number of files need to be stored, as in the corporate setting.

What is needed is a DBMS for storing, retrieving, and manipulating XML documents. Such a DBMS should:

- Be *MISSING CONTENT*

Two possible solutions for XML document management are a relational database management system (RDBMS) or an XML database.

## RDBMS

An XML document can be stored within an RDBMS. Storing an intact XML document as a CLOB is a sensible strategy if the XML document contains static content that will only be updated by replacing the entire document. Examples include written text such as articles, advertisements, books, or legal contracts. These document-centric files (e.g., articles and legal contracts) are retrieved and updated in their entirety.

For more dynamic data-centric XML files (e.g., orders, price lists, airline schedules), the RDBMS must be extended to support the structure of the data so that portions of the document (e.g., an element such as the price for a product) can be retrieved and updated.

## XML database

A second approach is to build a special-purpose XML database. Tamino<sup>25</sup> is an example of such an approach.

## MySQL and XML

MySQL has functions for storing, searching, and maintaining XML documents. Any XML file can be stored as a document, and each XML fragment is stored as a character string. Creation of a table is straightforward:

```
CREATE TABLE cdlib (  
  docid INT AUTO_INCREMENT,  
  doc VARCHAR(10000),  
  PRIMARY KEY(docid));
```

Insertion of an XML fragment follows the familiar pattern:

---

<sup>25</sup>[www.softwareag.com/Corporate/products/wm/tamino/default.asp](http://www.softwareag.com/Corporate/products/wm/tamino/default.asp)

```
INSERT INTO cdlib (doc) VALUES
('<cd>
  <cdid>A2 1325</cdid>
  <cdlabel>Atlantic</cdlabel>
  <cdtitle>Pyramid</cdtitle>
  <cdyear>1960</cdyear>
  <track length="00:02:30">
    <trknum>1</trknum>
    <trktitle>Vendome</trktitle>
  </track>
  <track length="00:10:46">
    <trknum>2</trknum>
    <trktitle>Pyramid</trktitle>
  </track>
</cd>');
```

## Querying XML

ExtractValue is the MySQL function for retrieving data from a node. It requires specification of the XML fragment to be retrieved and the XPath expression to locate the required node.

*Report the title of the first CD in the library*

```
SELECT ExtractValue(doc, '/cd/cdtitle[1]') FROM cdlib;
```

ExtractValue
Pyramid

*Report the title of the first track on the first CD.*

```
SELECT ExtractValue(doc, '//cd[1]/track[1]/trktitle') FROM cdlib;
```

ExtractValue
Vendome

## Updating XML

UpdateXML replaces a single fragment of XML with another fragment. It requires specification of the XML fragment to be replaced and the XPath expression to locate the required node.

*Change the title for the CD with identifier A2 1325 to Stonehenge*

```
UPDATE cdlib SET doc = UpdateXML(doc, '//cd[cdid="A2 1325"]/cdtitle', '<cdtitle>Stonehenge</cdtitle>');
```

You can repeat the previous query to find the title of the first CD to verify the update.

## Generating XML

A set of user defined functions (UDF)<sup>26</sup> has been developed to convert the results of an SQL query into XML. The `xql_element` function is used to define the name and value of the XML element to be reported. Here is an example.

*List in XML format the name of all stocks with a PE ratio greater than 14.*

```
SELECT xql_element ('firm',shrfirm) FROM share WHERE shrpe > 14
```

*Query output*

```
<firm>Canadian Sugar</firm>
<firm>Freedonia Copper</firm>
<firm>Sri Lankan Gold</firm>
```

The preceding is just a brief example of what can be done. Read the documentation on UDF to learn how to handle more elaborate transformations.

## Conclusion

XML has two main roles. The first is to facilitate the exchange of data between organizations and within those organizations that do not have integrated systems. Its second purpose is to support exchange between servers.

Mastery of XML is well beyond the scope of a single chapter. Indeed, it is a book-length topic, and hundreds of books have been written on XML. It is important to remember that the prime goal of XML is to support data interchange. If you would like to continue learning about XML, then consider the open content textbook ([en.wikibooks.org/wiki/XML](http://en.wikibooks.org/wiki/XML)), which was created by students and is under continual revision. You might want to contribute to this book.

## Summary

Electronic data exchange became more important with the introduction of the Internet. SGML, a precursor of XML, defines the structure of documents. SGML's value derives from its reusability, flexibility, support for revision, and format independence. XML, a derivative of SGML, is designed to support electronic commerce and overcome some of the shortcomings of SGML. XML supports data exchange by making information self-describing. It is a metalanguage because it is a language for generating other languages (e.g., finML). It provides substantial gains for the management and distribution of data. The XML language consists of an XML schema, document object model (DOM), and XSL. A schema defines the structure of a document and how an application should interpret XML markup tags. The DOM is a tree-based data model of an XML document. XSL is used to specify a stylesheet for displaying an XML document. XML documents can be stored in either a RDBMS or XML database.

## Key terms and concepts

Document object model (DOM)  
Document type definition (DTD)  
Electronic data interchange (EDI)

Markup language  
Occurrence indicators  
Standard generalized markup language (SGML)

---

<sup>26</sup>The lib\mysqludf\xql library is housed at <http://www.mysqludf.org>.) It might not be installed for the version of MySQL you are using.



## References and additional readings

Watson, R. T., and others. 2004. *XML: managing data exchange*:  
[http://en.wikibooks.org/wiki/XML\\_-\\_Managing\\_Data\\_Exchange](http://en.wikibooks.org/wiki/XML_-_Managing_Data_Exchange).

## Exercises

1. A business has a telephone directory that records the first and last name, telephone number, and e-mail address of everyone working in the firm. Departments are the main organizing unit of the firm, so the telephone directory is typically displayed in department order, and shows for each department the contact phone and fax numbers and e-mail address.
  - a. Create a hierarchical data model for this problem.
  - b. Define the schema.
  - c. Create an XML file containing some directory data.
  - d. Create an XSL file for a stylesheet and apply the transformation to the XML file.
2. Create a schema for your university or college's course bulletin.
3. Create a schema for a credit card statement.
4. Create a schema for a bus timetable.
5. Using the portion of ClassicModels that has been converted to XML,<sup>27</sup> answer the following questions using XPath.
  - a. List all customers.
  - b. Who is the last customer in the file?
  - c. Select all customers in Sweden.
  - d. List the payments of more than USD 100,000.
  - e. Select the first payments by Toys4GrownUps.com.
  - f. What was the payment date for check DP677013?
  - g. Who paid with check DP677013?
  - h. What payments were received on 2003-12-04?
  - i. Who made payments on 2003-12-04?
  - j. List the numbers of all checks from customers in Denmark.
6. Using the portion of ClassicModels that has been converted to XML, answer the following questions using XQuery.
  - a. List all customers.
  - b. Who is the last customer in the file?
  - c. Select all customers in Sweden sorted by customer name.
  - d. List the payments of more than USD 100,000.
  - e. Select the first payments by Toys4GrownUps.com.
  - f. What was the payment date for check DP677013?
  - g. Who paid with check DP677013?
  - h. What payments were received on 2003-12-04?
  - i. Who made payments on 2003-12-04?
  - j. List the numbers of all checks from customers in Denmark.

---

<sup>27</sup><http://www.richardtwatson.com/xml/customerpayments.xml>

## 11 Organizational Intelligence

*There are three kinds of intelligence: One kind understands things for itself, the other appreciates what others can understand, the third understands neither for itself nor through others. This first kind is excellent, the second good, and the third kind useless.*

Machiavelli, *The Prince*, 1513

### Learning objectives

Students completing this chapter will be able to

- understand the principles of organizational intelligence;
- decide whether to use verification or discovery for a given problem;
- select the appropriate data analysis technique(s) for a given situation;

### Introduction

Too many companies are *data rich* but *information poor*. They collect vast amounts of data with their transaction processing systems, but they fail to turn these data into the necessary information to support managerial decision making. Many organizations make limited use of their data because they are scattered across many systems rather than centralized in one readily accessible, integrated data store. Technologies exist to enable organizations to create vast repositories of data that can be then analyzed to inform decision making and enhance operational performance.

**Organizational intelligence** is the outcome of an organization's efforts to collect, store, process, and interpret data from internal and external sources. The conclusions or clues gleaned from an organization's data stores enable it to identify problems or opportunities, which is the first stage of decision making.

Organizational intelligence technology is in transition. In this chapter, we deal with the older version, which is still in place in many organizations. In the latter chapter on cluster computing, we cover the newer approach that some firms have already adopted. It is likely that a mix of the two sets of technologies will exist in parallel for some time.

### An organizational intelligence system

Transaction processing systems (TPSs) are a core component of organizational memory and thus an important source of data. Along with relevant external information, the various TPSs are the bedrock of an organizational intelligence system. They provide the raw facts that an organization can use to learn about itself, its competitors, and the environment. A TPS can generate huge volumes of data. In the United States, a telephone company may generate millions of records per day detailing the telephone calls it has handled. The hundreds of million credit cards on issue in the world generate billions of transactions per year. A popular Web site can have a hundred million hits per day. TPSs are creating a massive torrent of data that potentially reveals to an organization a great deal about its business and its customers.

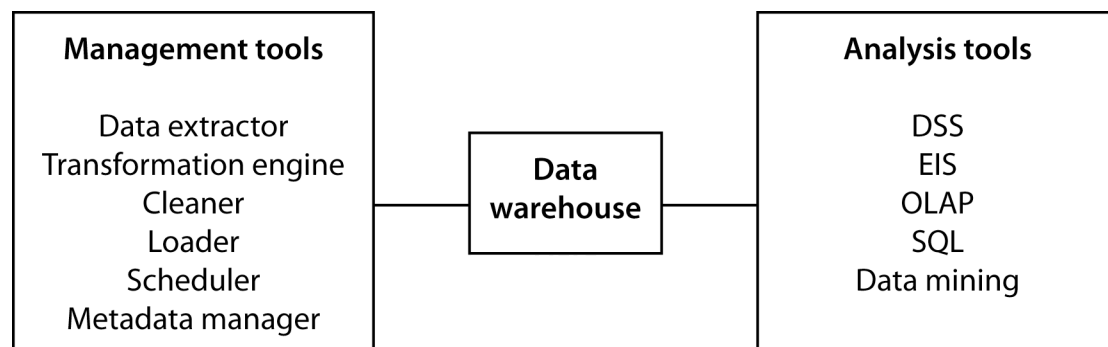
Unfortunately, many organizations are unable to exploit, either effectively or efficiently, the massive amount of data generated by TPSs. Data are typically scattered across a variety of systems, in different database technologies, in different operating systems, and in different locations. The fundamental problem is that organizational memory is highly fragmented. Consequently, organizations need a technology that can accumulate a considerable proportion of organizational memory into one readily accessible system. Making these data available to decision makers is crucial to improving organizational performance, providing first-class customer service, increasing revenues, cutting costs, and preparing for the future. For many organizations,

their memory is a major untapped resource—an *underused intelligence system containing undetected key facts about customers*. To take advantage of the mass of available raw data, an organization first needs to organize these data into one logical collection and then use software to sift through this collection to extract meaning.

The **data warehouse**, a subject-oriented, integrated, time-variant, and nonvolatile set of data that supports decision making, has emerged as the key device for harnessing organizational memory. *Subject* databases are designed around the essential entities of a business (e.g., customer) rather than applications (e.g., auto insurance). *Integrated* implies consistency in naming conventions, keys, relationships, encoding, and translation (e.g., gender is always coded as m or f in all relevant fields). *Time-variant* means that data are organized by various time periods (e.g., by months). Because a data warehouse is updated with a bulk upload, rather than as transactions occur, it contains *nonvolatile* data.

Data warehouses are enormous collections of data, often measured in terabytes, compiled by mass marketers, retailers, and service companies from the transactions of their millions of customers. Associated with a data warehouse are data management aids (e.g., data extraction), analysis tools (e.g., OLAP), and applications (e.g., executive information system).

*The data warehouse*



## The data warehouse

### Creating and maintaining the data warehouse

A data warehouse is a snapshot of an organization at a particular time. In order to create this snapshot, data must be extracted from existing systems, transformed, cleaned, and loaded into the data warehouse. In addition, regular snapshots must be taken to maintain the usefulness of the warehouse.

**Extraction** Data from the operational systems, stored in operational data stores (ODS), are the raw material of a data warehouse. Unfortunately, it is not simply a case of pulling data out of ODSs and loading them into the warehouse. Operational systems were often written many years ago at different times. There was no plan to merge these data into a single system. Each application is independent or shares little data with others. The same data may exist in different systems with different names and in different formats. The extraction of data from many different systems is time-consuming and complex. Furthermore, extraction is not a one-time process. Data must be extracted from operational systems on an ongoing basis so that analysts can work with current data.

**Transformation** Transformation is part of the data extraction process. In the warehouse, data must be standardized and follow consistent coding systems. There are several types of transformation:

- **Encoding:** Non-numeric attributes must be converted to a common coding system. Gender may be coded, for instance, in a variety of ways (e.g., m/f, 1/0, or M/F) in different systems. The extraction program must transform data from each application to a single coding system (e.g., m/f).

- **Unit of measure:** Distance, volume, and weight can be recorded in varying units in different systems (e.g., centimeters or inches) and must be converted to a common system.
- **Field:** The same attribute may have different names in different applications (e.g., sales-date, sdate, or saledate), and a standard name must be defined.
- **Date:** Dates are stored in a variety of ways. In Europe the standard for date is *dd/mm/yy*, in the U.S. it is *mm/dd/yy*, whereas the ISO standard is *yyyy-mm-dd*.

**Cleaning** Unfortunately, some of the data collected from applications may be *dirty*—they contain errors, inconsistencies, or redundancies. There are a variety of reasons why data may need cleaning:

- The same record is stored by several departments. For instance, both Human Resources and Production have an employee record. Duplicate records must be deleted.
- Multiple records for a company exist because of an acquisition. For example, the record for Sun Microsystems should be removed because it was acquired by Oracle.
- Multiple entries for the same entity exist because there are no corporate data entry standards. For example, FedEx and Federal Express both appear in different records for the same company.
- Data entry fields are misused. For example, an address line field is used to record a second phone number.

Data cleaning starts with determining the dirtiness of the data. An analysis of a sample should indicate the extent of the problem and whether commercial data-cleaning tools are required. Data cleaning is unlikely to be a one-time process. All data added to the data warehouse should be validated in order to maintain the integrity of the warehouse. Cleaning can be performed using specialized software or custom-written code.

**Loading** Data that have been extracted, transformed, and cleaned can be loaded into the warehouse. There are three types of data loads:

- **Archival:** Historical data (e.g., sales for the period 2005–2012) that is loaded once. Many organizations may elect not to load these data because of their low value relative to the cost of loading.
- **Current:** Data from current operational systems.
- **Ongoing:** Continual revision of the warehouse as operational data are generated. Managing the ongoing loading of data is the largest challenge for warehouse management. This loading is done either by completely reloading the data warehouse or by just updating it with the changes.

**Scheduling** Refreshing the warehouse, which can take many hours, must be scheduled as part of a data center’s regular operations. Because a data warehouse supports medium- to long-term decision making, it is unlikely that it would need to be refreshed more frequently than daily. For shorter decisions, operational systems are available. Some firms may decide to schedule less frequently after comparing the cost of each load with the cost of using data that are a few days old.

**Metadata** A data dictionary is a reference repository containing *metadata* (i.e., *data about data*). It includes a description of each data type, its format, coding standards (e.g., volume in liters), and the meaning of the field. For the data warehouse setting, a data dictionary is likely to include details of which operational system created the data, transformations of the data, and the frequency of extracts. Analysts need access to metadata so that they can plan their analyses and learn about the contents of the data warehouse. If a data dictionary does not exist, it should be established and maintained as part of ensuring the integrity of the data warehouse.

Data warehouse technology

Selecting an appropriate data warehouse system is critical to support significant data mining or online analytical processing. Data analysis often requires intensive processing of large volumes of data, and large main memories are necessary for good performance. In addition, the system should be scalable so that as the demand for data analysis grows, the system can be readily upgraded.

In recent years, there has been a shift to Hadoop, which is covered in the next chapter, as the foundation for a data warehouse. It offers speed and cost advantages over the technology that had predominated for some years.

Exploiting data stores

Two approaches to analyzing a data store (i.e., a database or data warehouse) are data mining and online analytical processing (OLAP). Before discussing each of these approaches, it is helpful to recognize the fundamentally different approaches that can be taken to exploiting a data store.

Verification and discovery

The **verification** approach to data analysis is driven by a hypothesis or conjecture about some relationship (e.g., customers with incomes in the range of \$50,000–75,000 are more likely to buy minivans). The analyst then formulates a query to process the data to test the hypothesis. The resulting report will either support or disconfirm the theory. If the theory is disconfirmed, the analyst may continue to propose and test hypotheses until a target customer group of likely prospects for minivans is identified. Then, the minivan firm may market directly to this group because the likelihood of converting them to customers is higher than mass marketing to everyone. The verification approach is highly dependent on a persistent analyst eventually finding a useful relationship (i.e., who buys minivans?) by testing many hypotheses. OLAP, DSS, EIS, and SQL-based querying systems support the verification approach.

Data mining uses the **discovery** approach. It sifts through the data in search of frequently occurring patterns and trends to report generalizations about the data. Data mining tools operate with minimal guidance from the client. Data mining tools are designed to yield useful facts about business relationships efficiently from a large data store. The advantage of discovery is that it may uncover important relationships that no amount of conjecturing would have revealed and tested.

A useful analogy for thinking about the difference between verification and discovery is the difference between conventional and open-pit gold mining. A conventional mine is worked by digging shafts and tunnels with the intention of intersecting the richest gold vein. Verification is like conventional mining—some parts of the gold deposit may never be examined. The company drills where it believes there will be gold. In open-pit mining, everything is excavated and processed. Discovery is similar to open-pit mining—everything is examined. Both verification and discovery are useful; it is not a case of selecting one or the other. Indeed, analysts should use both methods to gain as many insights as possible from the data.

Comparison of verification and discovery

Verification      Discovery	
What is the average sale for in-store and catalog customers?	What is the
What is the average high school GPA of students who graduate from college compared to those who do not?	What are th

OLAP

Edgar F. Codd, the father of the relational model, and colleagues (including, notably, Sharon B. Codd, his wife) proclaimed in 1993 that RDBMSs were never intended to provide powerful functions for data

synthesis, analysis, and consolidation. This was the role of spreadsheets and special-purpose applications. They argued that analysts need data analysis tools that complement RDBMS technology, and they put forward the concept of **online analytical processing (OLAP)**: the analysis of business operations with the intention of making timely and accurate analysis-based decisions.

Instead of rows and columns, OLAP tools provide multidimensional views of data, as well as some other differences. OLAP means fast and flexible access to large volumes of derived data whose underlying inputs may be changing continuously.

#### *Comparison of TPS and OLAP applications*

<i><b>TPS</b></i>	<i><b>OLAP</b></i>
Optimized for transaction volume	Optimized for data analysis
Process a few records at a time	Process summarized data
Real-time update as transactions occur	Batch update (e.g., daily)
Based on tables	Based on hypercubes
Raw data	Aggregated data
SQL is widely used	MDX becoming a standard

For instance, an OLAP tool enables an analyst to view how many widgets were shipped to each region by each quarter in 2012. If shipments to a particular region are below budget, the analyst can find out which customers in that region are ordering less than expected. The analyst may even go as far as examining the data for a particular quarter or shipment. As this example demonstrates, the idea of OLAP is to give analysts the power to view data in a variety of ways at different levels. In the process of investigating data anomalies, the analyst may discover new relationships. The operations supported by the typical OLAP tool include

- Calculations and modeling across dimensions, through hierarchies, or across members
- Trend analysis over sequential time periods
- Slicing subsets for on-screen viewing
- Drill-down to deeper levels of consolidation
- Drill-through to underlying detail data
- Rotation to new dimensional comparisons in the viewing area

An OLAP system should give fast, flexible, shared access to analytical information. Rapid access and calculation are required if analysts are to make ad hoc queries and follow a trail of analysis. Such quick-fire analysis requires computational speed and fast access to data. It also requires powerful analytic capabilities to aggregate and order data (e.g., summarizing sales by region, ordered from most to least profitable). Flexibility is another desired feature. Data should be viewable from a variety of dimensions, and a range of analyses should be supported.

## **MDDB**

OLAP is typically used with an MDDB, a data management system in which data are represented by a multidimensional structure. The MDDB approach is to mirror and extend some of the features found in spreadsheets by moving beyond two dimensions. These tools are built directly into the MDDB to increase the speed with which data can be retrieved and manipulated. These additional processing abilities, however, come at a cost. The dimensions of analysis must be identified prior to building the database. In addition,

MDDBs have size limitations that RDBMSs do not have and, in general, are an order of magnitude smaller than a RDBMS.

MDDB technology is optimized for analysis, whereas relational technology is optimized for the high transaction volumes of a TPS. For example, SQL queries to create summaries of product sales by region, region sales by product, and so on, could involve retrieving many of the records in a marketing database and could take hours of processing. A MDDB could handle these queries in a few seconds. TPS applications tend to process a few records at a time (e.g., processing a customer order may entail one update to the customer record, two or three updates to inventory, and the creation of an order record). In contrast, OLAP applications usually deal with summarized data.

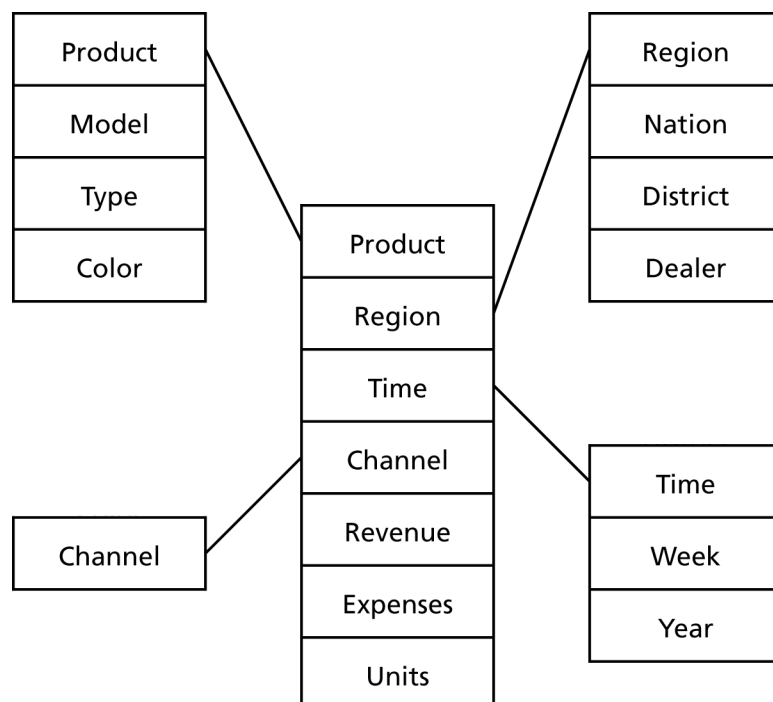
Fortunately, RDBMS vendors have standardized on SQL, and this provides a commonality that allows analysts to transfer considerable expertise from one relational system to another. Similarly, MDX, originally developed by Microsoft to support multidimensional querying of an SQL server, has been implemented by a number of vendors for interrogating an MDDB. More details are provided later in this chapter.

The current limit of MDDB technology is approximately 10 dimensions, which can be millions to trillions of data points.

**ROLAP** An alternative to a physical MDDB is a *relational OLAP* (or ROLAP), in which case a multidimensional model is imposed on a relational model. As we discussed earlier, this is also known as a logical MDDB. Not surprisingly, a system designed to support OLAP should be superior to trying to retrofit relational technology to a task for which it was not specifically designed.

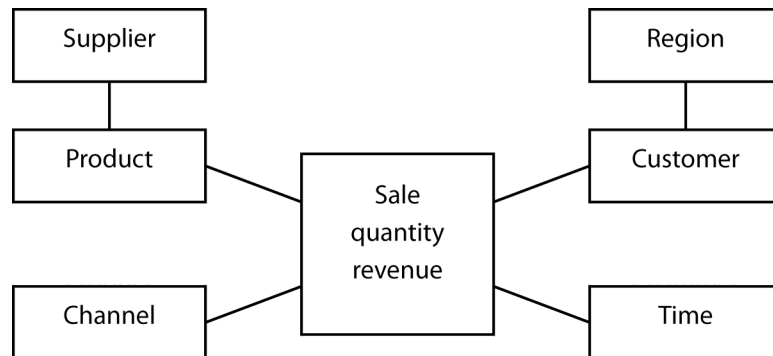
The **star schema** is used by some MDDBs to represent multidimensional data within a relational structure. The center of the star is a table storing multidimensional *facts* derived from other tables. Linked to this central table are the *dimensions* (e.g., region) using the familiar primary-key/foreign-key approach of the relational model. The following figure depicts a star schema for an international automotive company. The advantage of the star model is that it makes use of a RDBMS, a mature technology capable of handling massive data stores and having extensive data management features (e.g., backup and recovery). However, if the fact table is very large, which is often the case, performance may be slow. A typical query is a join between the fact table and some of the dimensional tables.

*A star schema*



A **snowflake schema**, more complex than a star schema, resembles a snowflake. Dimensional data are grouped into multiple tables instead of one large table. Space is saved at the expense of query performance because more joins must be executed. Unless you have good reasons, you should opt for a star over a snowflake schema.

*A snowflake schema*



*Rotation, drill-down, and drill-through*

MDDB technology supports **rotation** of data objects (e.g., changing the view of the data from “by year” to “by region” as shown in the following figure) and **drill-down** (e.g., reporting the details for each nation in a selected region as shown in the Drill Down figure), which is also possible with a relational system. Drill-down can slice through several layers of summary data to get to finer levels of detail. The Japanese data, for instance, could be dissected by region (e.g., Tokyo), and if the analyst wants to go further, Tokyo could be analyzed by store. In some systems, an analyst can **drill through** the summarized data to examine the source data within the organizational data store from which the MDDB summary data were extracted.

*Rotation*

Year	Data	Region			Grand total
		Asia	Europe	North America	
2010	Sum of hardware	97	23	198	318
	Sum of software	83	41	425	549
2011	Sum of hardware	115	28	224	367
	Sum of software	78	65	410	553
2012	Sum of hardware	102	25	259	386
	Sum of software	55	73	497	625
Total sum of hardware	314	76	681	1,071	
Total sum of software	216	179	1,322	1717	

Region	Data	Year			Grand total
		2010	2011	2012	
Asia	Sum of hardware	97	115	102	314
	Sum of software	83	78	55	216
Europe	Sum of hardware	23	28	25	76
	Sum of software	41	65	73	179
North America	Sum of hardware	198	224	259	681
	Sum of software	425	410	497	1,332
Total sum of hardware	318	367	386	1,071	
Total sum of software	549	553	625	1,727	

*Drill-down*

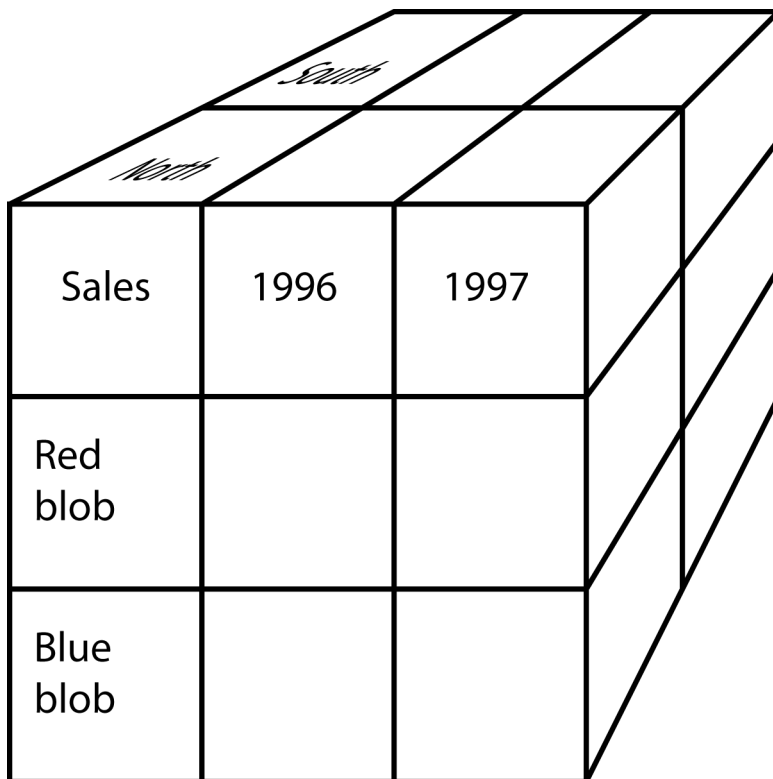


	Region	Sales variance		Nation	Sales variance
Africa	105%		—>	China	123%
Asia	57%			Japan	52%
Europe	122%			India	87%
North America	97%			Singapore	95%
Pacific	85%				
South America	163%				

## The hypercube

From the analyst's perspective, a fundamental difference between MDDB and RDBMS is the representation of data. As you know from data modeling, the relational model is based on tables, and analysts must think in terms of tables when they manipulate and view data. The relational world is two-dimensional. In contrast, the **hypercube** is the fundamental representational unit of a MDDB. Analysts can move beyond two dimensions. To envisage this change, consider the difference between the two-dimensional blueprints of a house and a three-dimensional model. The additional dimension provides greater insight into the final form of the building.

*A hypercube*



Of course, on a screen or paper only two dimensions can be shown. This problem is typically overcome by selecting an attribute of one dimension (e.g., North region) and showing the other two dimensions (i.e., product sales by year). You can think of the third dimension (i.e., region in this case) as the page dimension—each page of the screen shows one region or slice of the cube.

	Page	Columns
Region: North		Sales

	Page	Columns		
		Red blob	Blue blob	Total
Rows Year	2011			
	2012			
	Total			

#### *A three-dimensional hypercube display*

A hypercube can have many dimensions. Consider the case of a furniture retailer who wants to capture six dimensions of data. Although it is extremely difficult to visualize a six-dimensional hypercube, it helps to think of each cell of the cube as representing a fact (e.g., the Atlanta store sold five Mt. Airy desks to a business in January).

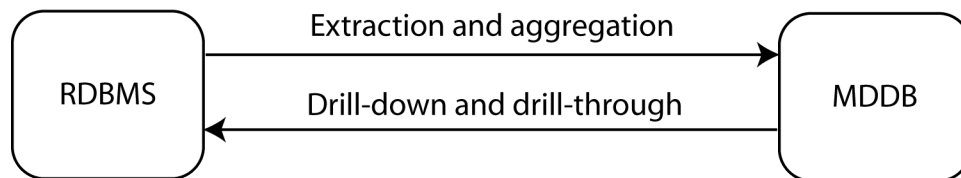
#### *A six-dimensional hypercube*

Dimension	Example
Brand	Mt. Airy
Store	Atlanta
Customer segment	Business
Product group	Desks
Period	January
Variable	Units sold

Similarly, a six-dimensional hypercube can be represented by combining dimensions (e.g., brand and store can be combined in the row dimension by showing stores within a brand).

A quick inspection of the table of the comparison of TPS and OLAP applications reveals that relational and multidimensional database technologies are designed for very different circumstances. Thus, the two technologies should be considered as complementary, not competing, technologies. Appropriate data can be periodically extracted from an RDBMS, aggregated, and loaded into an MDDB. Ideally, this process is automated so that the MDDB is continuously updated. Because analysts sometimes want to drill down to low-level aggregations and even drill through to raw data, there must be a connection from the MDDB to the RDBMS to facilitate access to data stored in the relational system.

#### *The relationship between RDBMS and MDDB*



### **Designing a multidimensional database**

The multidimensional model, based on the hypercube, requires a different design methodology from the relational model. At this stage, there is no commonly used approach, such as the entity-relationship principle of the relational model. However, the method proposed by Thomsen (1997) deserves consideration.

The starting point is to identify what must be tracked (e.g., sales for a retailer or revenue per passenger mile for a transportation firm). A collection of tracking variables is called a **variable dimension**.

The next step is to consider what types of analyses will be performed on the variable dimension. In a sales system, these may include sales by store by month, comparison of this month's sales with last month's

for each product, and sales by class of customer. These types of analyses cannot be conducted unless the instances of each variable have an identifying tag. In this case, each sale must be tagged with time, store, product, and customer type. Each set of identifying factors is an **identifier dimension**. As a cross-check for identifying either type of dimension, use these six basic prompts.

*Basic prompts for determining dimensions*

	Prompt	Example	Source
When?	June 5, 2013 10:27am		Transaction data
Where?	Paris		
What?	Tent		
How?	Catalog		
Who?	Young adult woman		Face recognition or credit card issuer
Why?	Camping trip to Bolivia		Social media
Outcome?	Revenue of €624.00		Transaction data

Most of the data related to the prompts can be extracted from transactional data. Face recognition software could be used to estimate the age and gender of the buyer in a traditional retail establishment. If the buyer uses a credit card, then such data, with greater precision, could be obtained from the bank issuing the credit card. In the case of why, the motivation for the purchase, the retailer can mine social exchanges made by the customer. Of course, this requires the retailer to be able to uniquely identify the buyer, through a store account or credit card, and use this identification to mine social media.

Variables and identifiers are the key concepts of MDDB design. The difference between the two is illustrated in the following table. Observe that time, an identifier, follows a regular pattern, whereas sales do not. Identifiers are typically known in advance and remain constant (e.g., store name and customer type), while variables change. It is this difference that readily distinguishes between variables and identifiers. Unfortunately, when this is not the case, there is no objective method of discriminating between the two. As a result, some dimensions can be used as both identifiers and variables.

*A sales table*

Identifier	Variable
time (hour)	sales (dollars)
10:00	523
11:00	789
12:00	1,256
13:00	4,128
14:00	2,634

There can be a situation when your intuitive notion of an identifier and variable is not initially correct. Consider a Web site that is counting the number of hits on a particular page. In this case, the identifier is hit and time is the variable because the time of each hit is recorded.

*A hit table*

Identifier	Variable
hit	time (hh:mm:ss)
1	9:34:45
2	9:34:57
3	9:36:12
4	9:41:56

The next design step is to consider the form of the dimensions. You will recall from statistics that there are three types of variables (dimensions in MDDb language): nominal, ordinal, and continuous. A nominal variable is an unordered category (e.g., region), an ordinal variable is an ordered category (e.g., age group), and a continuous variable has a numeric value (e.g., passenger miles). A hypercube is typically a combination of several types of dimensions. For instance, the identifier dimensions could be product and store (both nominal), and the variable dimensions could be sales and customers. A dimension's type comes into play when analyzing relationships between identifiers and variables, which are known as independent and dependent variables in statistics. The most powerful forms of analysis are available when both dimensions are continuous. Furthermore, it is always possible to recode a continuous variable into ordinal categories. As a result, wherever feasible, data should be collected as a continuous dimension.

*Relationship of dimension type to possible analyses*

		Identifier dimension	
Variable dimension	Continuous	Continuous	Nominal or ordinal
		Regression and curve fitting	Analysis of variance
		Sales over time	Sales by store
	Nominal or ordinal	Logistic regression <i>Customer response (yes or no) to the level of advertising</i>	Contingency table analysis Number of sales by region

This brief introduction to multidimensionality modeling has demonstrated the importance of distinguishing between types of dimensions and considering how the form of a dimension (e.g., nominal or continuous) will affect the choice of analysis tools. Because multidimensional modeling is a relatively new concept, you can expect design concepts to evolve. If you become involved in designing an MDDb, then be sure to review carefully current design concepts. In addition, it would be wise to build some prototype systems, preferably with different vendor implementations of the multidimensional concept, to enable analysts to test the usefulness of your design.

**Skill builder** A national cinema chain has commissioned you to design a multidimensional database for its marketing department. What identifier and variable dimensions would you select?

## Data mining

Data mining is the search for relationships and global patterns that exist in large databases but are hidden in the vast amounts of data. In data mining, an analyst combines knowledge of the data with advanced machine learning technologies to discover nuggets of knowledge hidden in the data. Data mining software can find meaningful relationships that might take years to find with conventional techniques. The software is designed to sift through large collections of data and, by using statistical and artificial intelligence techniques, identify hidden relationships. The mined data typically include electronic point-of-sale records, inventory, customer transactions, and customer records with matching demographics, usually obtained from an external source. Data mining does not require the presence of a data warehouse. An organization can mine data from its operational files or independent databases. However, data mining independent files will not uncover relationships that exist between data in different files. Data mining will usually be easier and more effective when the organization accumulates as much data as possible in a single data store, such as a data warehouse. Recent advances in processing speeds and lower storage costs have made large-scale mining of corporate data a reality.

Database marketing, a common application of data mining, is also one of the best examples of the effective use of the technology. Database marketers use data mining to develop, test, implement, measure, and

modify tailored marketing programs. The intention is to use data to maintain a lifelong relationship with a customer. The database marketer wants to anticipate and fulfill the customer's needs as they emerge. For example, recognizing that a customer buys a new car every three or four years and with each purchase gets an increasingly more luxurious car, the car dealer contacts the customer during the third year of the life of the current car with a special offer on its latest luxury model.

## Data mining uses

There are many applications of data mining:

- Predicting the probability of default for consumer loan applications. Data mining can help lenders reduce loan losses substantially by improving their ability to predict bad loans.
- Reducing fabrication flaws in VLSI chips. Data mining systems can sift through vast quantities of data collected during the semiconductor fabrication process to identify conditions that cause yield problems.
- Predicting audience share for television programs. A market-share prediction system allows television programming executives to arrange show schedules to maximize market share and increase advertising revenues.
- Predicting the probability that a cancer patient will respond to radiation therapy. By more accurately predicting the effectiveness of expensive medical procedures, health care costs can be reduced without affecting quality of care.
- Predicting the probability that an offshore oil well is going to produce oil. An offshore oil well may cost \$30 million. Data mining technology can increase the probability that this investment will be profitable.
- Identifying quasars from trillions of bytes of satellite data. This was one of the earliest applications of data mining systems, because the technology was first applied in the scientific community.

## Data mining functions

Based on the functions they perform, five types of data mining functions exist:

**Associations** An association function identifies affinities existing among the collection of items in a given set of records. These relationships can be expressed by rules such as “72 percent of all the records that contain items A, B, and C also contain items D and E.” Knowing that 85 percent of customers who buy a certain brand of wine also buy a certain type of pasta can help supermarkets improve use of shelf space and promotional offers. Discovering that fathers, on the way home on Friday, often grab a six-pack of beer after buying some diapers, enabled a supermarket to improve sales by placing beer specials next to diapers.

**Sequential patterns** Sequential pattern mining functions identify frequently occurring sequences from given records. For example, these functions can be used to detect the set of customers associated with certain frequent buying patterns. Data mining might discover, for example, that 32 percent of female customers within six months of ordering a red jacket also buy a gray skirt. A retailer with knowledge of this sequential pattern can then offer the red-jacket buyer a coupon or other enticement to attract the prospective gray-skirt buyer.