CHAPTER 6

RELATIONAL DATABASES AND SQL

LEARNING OBJECTIVES

AFTER READING THIS CHAPTER, YOU SHOULD BE ABLE TO:

- UNDERSTAND THE TECHNIQUES USED TO MODEL COMPLEX ACCOUNTING PHENOMENA IN AN ENTITY-RELATIONSHIP (E-R) DIAGRAM.
- DEVELOP E-R DIAGRAMS THAT MODEL EFFECTIVE ACCOUNTING DATABASE STRUCTURES USING THE RESOURCES, EVENTS, AGENTS (REA) APPROACH.
- RECOGNIZE THE COMPONENTS OF RELATIONAL TABLES AND THE KEYS TO EFFECTIVE RELATIONAL DATABASE DESIGN.
- UNDERSTAND THE USE OF SQL COMMANDS TO CREATE RELATIONAL TABLES DURING IMPLEMENTATION OF THE MODEL.
- MANIPULATE RELATIONAL TABLES TO EXTRACT THE NECESSARY DATA DURING DECISION MAKING.

When you complete this course and ultimately finish your formal education, you will likely embark on a new career. When your new boss approaches you and asks you to pick out your new office furniture, so it can be ordered, consider the accounting entry that is made when the purchase order is issued—actually, under traditional double-entry accounting, *no* entry is made! The debits and credits do not begin flowing until title is passed (usually when the furniture is received). Considering that one of the high-risk areas for fraud today is the purchasing function of an organization, there is a strong need to monitor the purchasing process. Potential purchasing fraud includes using unapproved (and possibly related) vendors, purchasing goods for personal use, and vendors giving kickbacks to buyers. If an information system only captures debits and credits, there may be no way to trace what happens, which makes financial management, as well as preventing and detecting fraud, extremely difficult.

The REA approach to developing models of accounting databases addresses part of this issue by ensuring that a system captures all data related to business events (such as issuing a purchase order), not just the debit/credit transactions. Capturing relevant data, combined with a method of extracting and analyzing that data, such as SQL, helps managers manage. It also reduces fraud-risk, and if fraud does occur, it makes the work of a forensic accountant much more productive.

Synopsis

In this chapter, we describe the REA (Resources, Events, Agents) approach for developing models of accounting databases and using those models to build relational databases. We also describe SQL, a database query language used to construct and manipulate relational databases.

You will learn to develop more complex entity-relationship (E-R) diagrams, integrate the REA concepts with E-R diagrams, as well as create and manipulate relational databases with SQL. These advanced database techniques provide the foundation for understanding how ERP systems are constructed and how they function in a business environment. Advanced database skills will aid you in designing effective accounting information systems, finding the data you need to perform accounting tasks, and creating reports that provide your data in an easy-to-use format. At the conclusion of your study of this chapter you should understand how enterprise databases are constructed and used in modern organizations.

Introduction

In earlier chapters, you learned that organizations need well-designed information and DBMSs to effectively support decision making. You learned how to create data models using both bottom-up and top-down approaches and learned about alternative types of DBMSs.

In this chapter, our goal is to increase your knowledge of DBMSs, data modeling, DBMS implementation, and query languages. We begin with a more in-depth discussion of *entity-relationship (E-R) modeling* and the REA approach to designing data models for accounting systems. This discussion is followed by an exploration of the key components and concepts underlying relational DBMSs and how to build a working set of relational tables from an *E-R model*. We then examine the core commands in SQL for purposes of creating and manipulating relational databases. Finally, we discuss security- and ethics-related issues surrounding DBMSs.

We have discussed the importance of data to organizations in the information age, as well as a shift in the focus of organizations from using information for operational control toward the use of information in decision-support applications. Information systems today must effectively collect, organize, and integrate the data necessary to support decision making. The successful accountant of tomorrow will manage the creation of *data warehouses* and effectively use tools such as *data mining* to gather information that will help managers make better decisions.

REA Modeling

In Chapter 5, you learned some basic E-R modeling concepts, including *entities* and *attributes*. In this section, you will gain a deeper understanding of how database designers use the REA model to identify entities and attributes for accounting applications. The development of the REA approach is discussed in Technology Summary 6.1.

Entities and Attributes

Extending the concepts from Chapter 5, an entity in an accounting system can be classified as a resource, event, agent, or location about which data are collected. You learned that resources could include merchandise inventory, equipment, and cash.

TECHNOLOGY SUMMARY 6.1

THE REA MODEL

In 1982, accounting researcher William McCarthy published a paper[a] that outlined the REA approach, which was a new way of thinking about accounting information. At that time, computerized accounting systems were primarily designed to perform the same tasks that accountants performed in manual accounting systems. For example, the computer running a sales system would be programmed to maintain one file that stored sales records and another file that stored information about payments received. Periodically, the computer would run both files and update a third file that included the accounts receivable records.

The designers of these old accounting systems modeled the things that accountants did in manual systems instead of modeling the business events themselves. In other words, the computerized accounting systems were models of the existing accountant's model of business events (double-entry bookkeeping) instead of being modeled after business events.

McCarthy became interested in the E-R modeling techniques you learned about in Chapter 5. E-R modeling was still relatively new and was still a theoretical curiosity; computers powerful enough to implement relational databases for large companies did not exist in the late 1970s. McCarthy first wrote about the potential of E-R modeling for accounting applications in 1979, hopeful that the technology would eventually become available.[b] He was attracted by the E-R model's capability to capture the meaning (or **semantics**) of business events more efficiently and effectively than the double-entry bookkeeping model could. Instead of modeling the accounting artifacts of journals and ledgers, the E-R model-based systems could focus on the elements of transactions and other business events. McCarthy classified business information into three categories: resources, events, and agents. These categories formed the basis for the REA model.

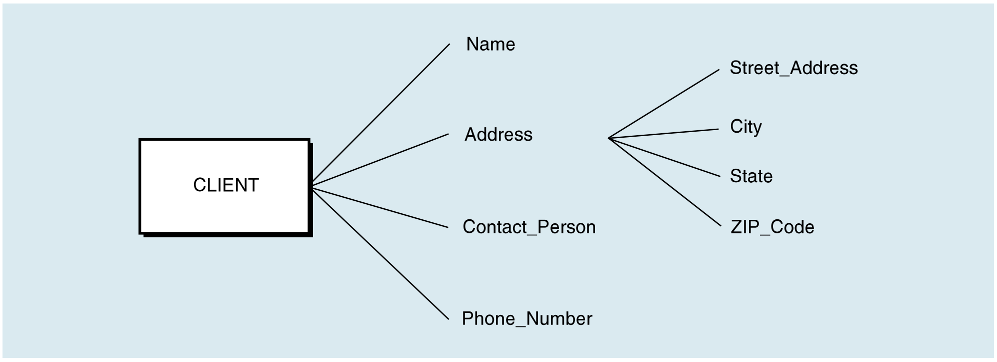
The REA model opened the door to a new world of accounting system design. Other accounting researchers worked with McCarthy to refine and improve the REA model.[c] As computers become more powerful and relational DBMS software became easier to use, the REA model was used increasingly to design accounting systems that could be integrated with other enterprise data. These integrated, enterprise-wide databases are used today to control and plan every aspect of a company's business processes.

[a] William E. McCarthy, ''The REA Accounting Model: A Generalized Framework for Accounting Systems in a Shared Data Environment,'' *The Accounting Review* 57(3) (July 1982): 554–578.
[b] William E. McCarthy, ''An Entity-Relationship View of Accounting Models,'' *The Accounting Review* 54(4) (October 1979): 667–686.
[c] Several publications discuss and extend the REA model, including Guido Geerts and William E. McCarthy, ''An Ontological Analysis of the Economic Primitives of the Extended-REA Enterprise Information Architecture,'' *International Journal of Accounting Information Systems* 3(1) (March 2002): 1–15; Cheryl Dunn and Severin Grabski, ''An Investigation of Localization as an Element of Cognitive Fit in Accounting Model Representations,'' *Decision Sciences* 32(1) (Winter 2001): 55–98; William E. McCarthy, ''The REA Modeling Approach to Teaching Accounting Information Systems,'' *Issues in Accounting Education* 18(4) (November 2003): 427–441; D.E. O'Leary, ''On the Relationship Between REA and SAP,'' *International Journal of Accounting Information Systems* 5(1) (May 2004): 65–81.

Events might include orders, sales, and purchases. Agents can be people such as customers and employees; agents also can be organizations, such as corporate vendors. Locations are physical objects or spaces at which events occur, resources are stored, or agents participate in events. An entity may also be described as anything in which we are interested that exists independently. An **instance** of an entity is one specific thing of the type defined by the entity. For example, the agent entity EMPLOYEE in a small company with three employees might have instances of Marge Evans, Roberto Garcia, and Arte Singh. In a relational database, the entity is represented as a table, and the three instances of the entity are represented as rows in that table.

To understand which entity we are capturing in our database and, likewise, to identify that unique entity when we retrieve the data, we need to describe the entity in detail. Data models describe entities by capturing their essential characteristics. The essential characteristics of an entity are its attributes. An **attribute** is an item of data that characterizes an entity or relationship. Figure 6.1 displays an attribute hierarchy for the

FIGURE 6.1 Attribute Hierarchy for the Entity CLIENT



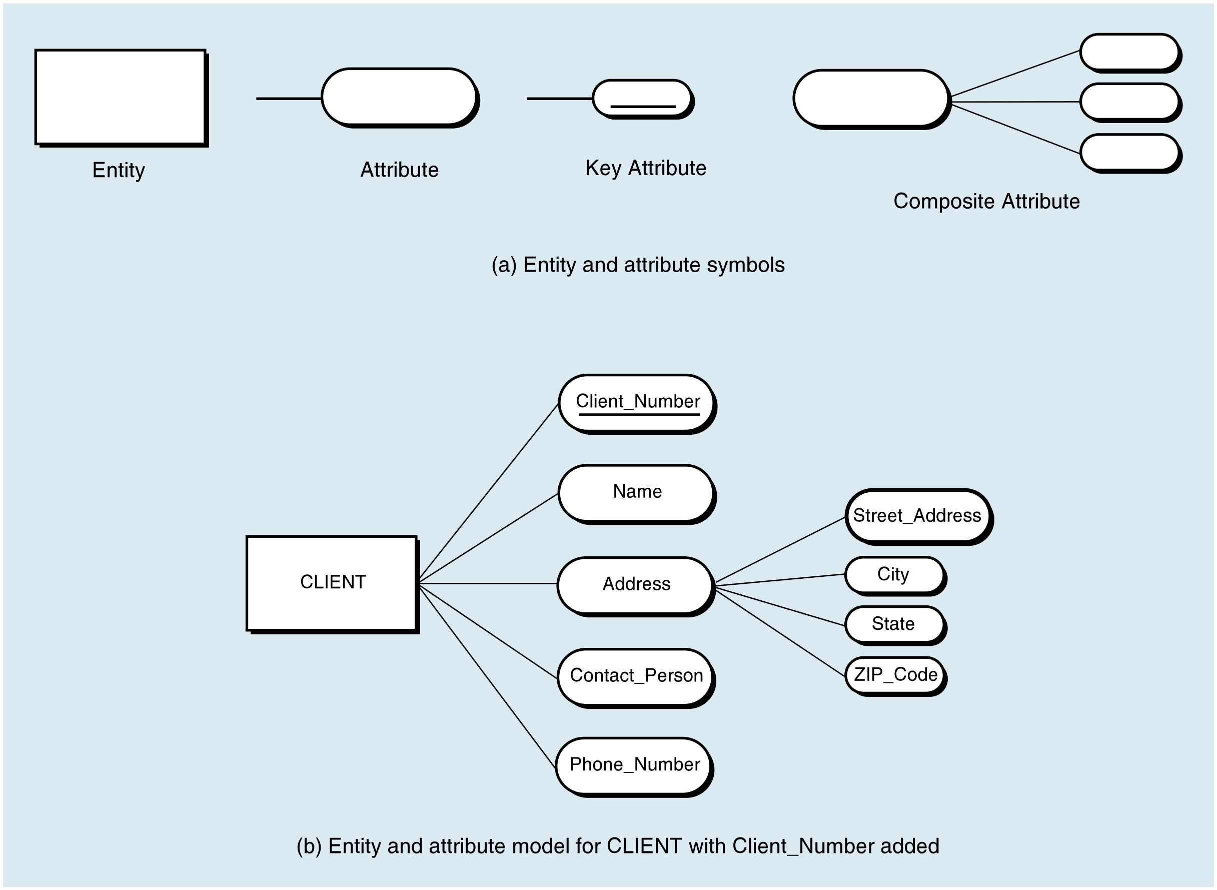
agent entity CLIENT. To fully describe a CLIENT, we need to record several attributes such as Name, Address, Contact_Person, and Phone_Number. Sometimes, attributes are a combination of parts that have unique meanings of their own. As you can see in Figure 6.1, the attribute Address can include several independent subattributes such as Street_Address, City, State, and ZIP_Code. Attributes that consist of multiple sub-attributes are referred to as **composite attributes**. The degree to which a database designer breaks down attributes is a matter of judgment. For example, the attribute Street_Address could be broken down into further subattributes such as Street_Number, Street_Name, Street_Type (road, avenue, lane, and so on), and Street_Directional_Suffix (N, S, E, NE, and so on). Although an important goal of attribute identification is to break the attributes down into small components, attributes do not need to be broken into the smallest possible units in every case.

Note that an important assumption lies behind our specification of the attributes for the agent entity CLIENT. We have assumed that a common set of attributes exists for each instance of CLIENT.[1] That is, every client has a name, address, contact person, and phone number. To design an effective data model, you must learn to identify the complete set of entities and the common attributes that fully describe each entity. The REA approach helps the designers of accounting databases identify a complete set of entities. It is important that the attributes allow the user of a database to uniquely identify each entity in the database.

To achieve the objective of uniquely identifying each entity to be stored in our database, it is necessary that one or more attributes be identified that will allow the user to access the entity that he or she is seeking. A **key attribute** is the attribute whose value is unique (i.e., different) for every entity that will ever appear in the database and is the most meaningful way of identifying each entity. This key attribute becomes the primary key (as discussed in Chapter 5). For our CLIENT agent entity, we might be tempted to use Name for the key attribute, but alphabetic-based attributes such as names are tricky because computers do not consistently distinguish between (or fail to distinguish between) uppercase and lowercase letters. Further, spellings and full names can be tricky in that one user might view the company name as "Arnold Consultants," whereas another user might use the full name, "Arnold Consultants, LLP." Similar problems can

1 Technically, CLIENT is an "entity type," however, most database developers use the term *entity* rather than "entity type." Entity type describes a collective group of entities (e.g., different clients). Entities of interest will usually fall into some category of similar-type entitie*s*. We will use this common terminology throughout the remainder of our discussion.

FIGURE 6.2 Symbols Used in E-R and REA Diagrams



arise with company names such as "The Final Authority." A well-intentioned data entry clerk might enter that name as "Final Authority, The" or simply "Final Authority" after concluding that the article "The" is not an important part of the company name. Most designers would use a numeric-valued or a nonnaming alphabetic attribute using one of the coding approaches you learned about in Chapter 5. For instance, an internally generated client number could be assigned to each instance in the CLIENT table. A numeric form using a *sequential coding* scheme might assign a number such as "12345." A nonnaming alphabetic form using *block coding* to categorize companies by the first letter of a company's name might assign an alphanumeric such as "A1234" for the client number.

Figure 6.2, part (a), shows one common set of symbols that are used to represent entities and attributes in E-R diagrams. In Figure 6.2, part (b), the rectangle is used to represent the CLIENT agent entity. To map the attributes of an entity, we add ovals connected to the entity (as shown in part a) for each attribute. Notice in part (b) that we have added an oval for each of the attributes shown in Figure 6.1 (pg. 179). For the composite attribute Address, we use the same oval connectors for each of the sub-attributes of the main attribute. Note that we have added a new attribute to the set of attributes shown in Figure 6.1—Client_Number. This attribute is the CLIENT entity's key attribute. The underline beneath the attribute name documents its selection as the key attribute.

Relationships

In Chapter 5, we defined relationships as associations between entities. As you have learned, a database includes multiple entities. However, to make the data stored in entities available to users who might want to reconstruct descriptions of various business events, the entities must be logically linked to represent the relationships that exist between them. The ease with which a user can extract related data from a database is heavily dependent on the quality of the database's logical design—that is, effective identification of the relationships between different entities. These relationships map and define the way in which data can be extracted from the database in the future. The mapping of the relationships between entities (i.e., development of the E-R diagram) provides a roadmap for getting from one piece of data in the database to another related piece of data—much as a road map in an atlas might show you how to drive from one city to another.

A three-step strategy is generally most effective in identifying all the relationships that should be included in a model. First, identify users' existing and desired information requirements to determine whether relationships in the data model can fulfill those requirements. Second, evaluate each of the entities in pairs to determine whether one entity in the pair provides a better description of an attribute contained in the other entity in the pair. Third, evaluate each entity to determine if there would be any need for two occurrences of the same entity type to be linked.

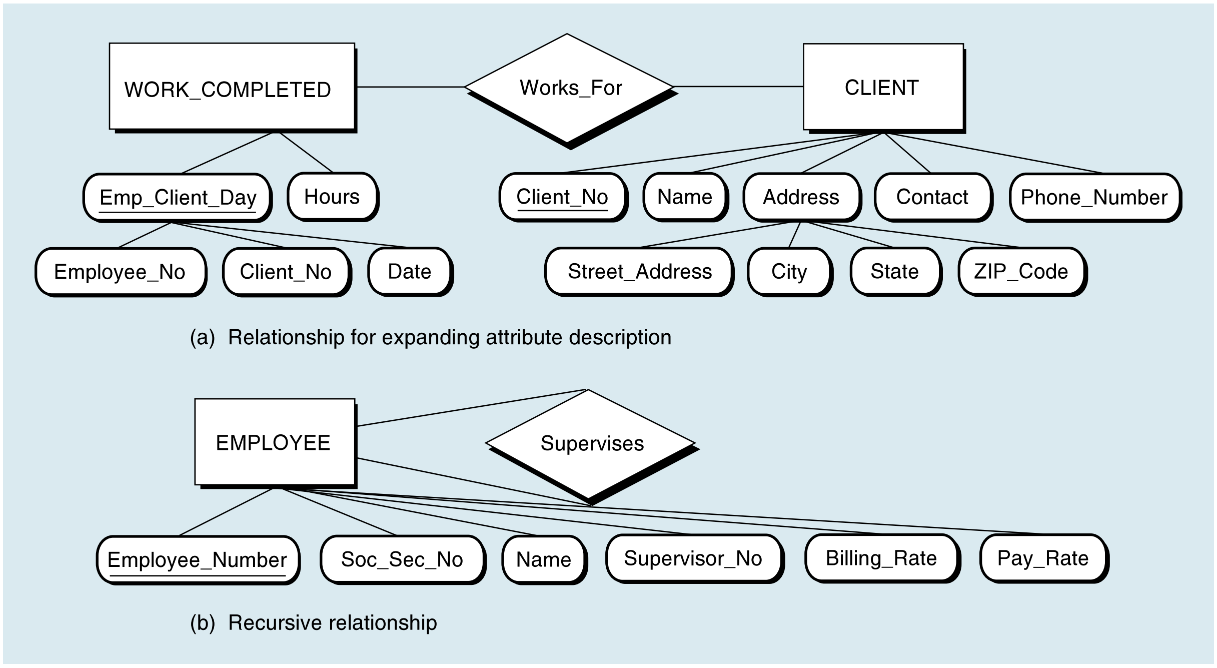
When designing a database, it is important that you learn about the business events that occur in the company and that you understand users' information requirements. This helps you identify all the ways in which different entities are related to each other in the company. This information will give you an idea of which relationships are required in the data model. The most common way to gather information about relationships in a particular company is to conduct interviews of the company's employees. All employees (not just managers) who work with the business process you are modeling can be good sources of information about relationships.

In the remainder of this chapter, we use an illustrative example of the client billing process that many public accounting, consulting, and legal firms use. In this client billing system, employees of the firm keep track of how much time they spend working on each client. Each employee fills out a weekly time sheet to record the time spent on each client. The hours spent on each client are then multiplied by the employee's billable rate for each hour worked. The cumulative fees for all employees' work are used to generate a bill for each client. The business process here is the capture of all information necessary to track employees' work hours and client billing information.

Examine Figure 6.3 (pg. 182) briefly before we go on. The figure includes information about three entities and their attributes. Using the REA approach, we have identified one event and two agents that participate in the business process of billing for professional services. The WORK_COMPLETED entity is an event. The CLIENT and EMPLOYEE entities are agents. No resources or locations are tracked in this data model for the client billing business process.

Desirable linkages between entities will often be fairly easy to recognize when the relationship defines an attribute. If our billing system requires that we know for which client an employee has worked, the entity representing work completed needs to include a client number. This client number would link the WORK_COMPLETED entity to the CLIENT entity that provides a full description of the attribute denoted by client number in WORK_COMPLETED. As you can see in Figure 6.3, part (a), CLIENT is an entity and not an attribute of WORK_COMPLETED. However, CLIENT does improve the description of an attribute for the work completed—the client for whom

FIGURE 6.3 Relationship Types in the REA Model of the Client Billing Business Process



the work was performed. This descriptive value suggests that a relationship exists between the CLIENT entity and the entity capturing the completed work as shown in Figure 6.3, part (a). Hence, we often can identify the need for defining relationships (such as Works_For) by examining the prescribed entities as pairs (in this case, we examined the pair CLIENT and WORK_COMPLETED) to identify logical linkages that would improve the description of an entity's attributes.

Another type of relationship is displayed in Figure 6.3, part (b). The relationship Supervises is called a *recursive relationship*. A **recursive relationship** is a relationship between two different instances of an entity. For example, most organizations have relationships among employees in which one employee supervises other employees. This relationship is often important in business processes and in decision-making contexts. Thus, this relationship should be represented in our database. Figure 6.3, part (b) shows how a recursive relationship is displayed in an REA data model diagram. One tempting alternative is to represent supervisors and their supervised employees as separate entities in the model. Unfortunately, this separate entity approach yields data redundancies when the supervisor is supervised by a third employee. Thus, it is easier and more logically correct to use a recursive relationship to the entity, EMPLOYEE. In this recursive relationship, a link is created between the employee who is being supervised and another employee who is the supervisor. As shown in Figure 6.3, part (b), the diamond represents the recursive relationship, Supervises, just as it would be used to show any relationship (such as the Works_For relationship in part (a)).

Model Constraints

In this section, we explore the various types of relationships that can occur and discuss the constraints used to specify such relationships. In Chapter 5, we briefly explored three different relationship types: 1:N (one-to-many), M:N (many-to-many), and 1:1 (one-to-one). You learned in Chapter 5 that the degree of these three relationship types is called *cardinality*.

FIGURE 6.4 Relationship Constraints in the Client Billing Business Process

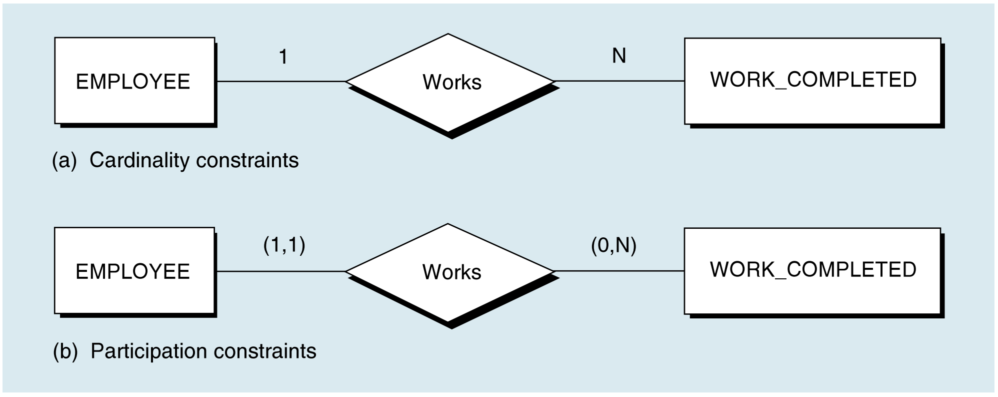


Figure 6.4, part (a) is an REA diagram that shows the maximum cardinalities for the Works relationship between the agent entity EMPLOYEE and the event entity WORK_COMPLETED in the client billing business process. The "1" above the left line of the relationship indicates that one employee performs each completed work entry. The "N" above the right line indicates that an employee can perform many work entries.

To determine the cardinality of a relationship, ask yourself the question, "How many items (records) in this entity could be related to any one item (record) in the other entity—one or many?" The answer determines that half of the cardinality ratio, and then the same question is asked in the reverse direction of the relationship to determine the other half of the cardinality ratio. In our example, we take the relationship in Figure 6.4, part (a), and ask the question, "How many work completed entries can an employee have?" The answer is many (based on the attributes specified for WORK_COMPLETED in Figure 6.3, part (a), which indicates that a given occurrence in the WORK_COMPLETED entity relates to one employee's time spent on a given client in a single time period). The question is then reversed to ask, "How many employees can provide a specific work completed entry?" The maximum number will be one. Hence, the cardinality of the relationship is specified as one-to-many and indicated on the diagram with the "1" and "N" notation. Most DBMSs include a feature that enforces maximum constraints. In other words, the DBMS will ensure that data are never entered that connect more than one employee to a single work completed entry.

Cardinality is the most common constraint specified in E-R diagrams. The other meaningful constraint that may be specified is participation. The **participation constraint** specifies the degree of minimum participation of one entity in the relationship with the other entity. In Figure 6.4, part (b), the *participation constraints* appear in the diagram. In the Works relationship, not every employee will have completed a billable work activity. Some employees are new and are not yet billable, and others might have nonclient service responsibilities, such as training or new business development. The "many" cardinality that appears in part (a) of the diagram only specifies the maximum participation in the relationship, not the minimum. The minimum participation in the relationship can be zero or one. The notation (0,N) on the line on the right in part (b) reflects the range of zero to many occurrences of work being completed on client projects, where the numbers reflect (minimum, maximum). The notation (1,1) on the line on the left side in part (b), illustrates that for any given occurrence of work completed for a client, the maximum of one employee providing the specific service still holds. In this case, the minimum also will be one because an employee must perform a particular occurrence of the completed work. The (1,1) relationship reflects that there is a required participation of one, and only one, employee.

Although the participation constraint does provide more information, it is still used less frequently than the cardinality constraint. In this book, we will present the diagrams using the maximum cardinality and will omit the participation (or minimum) constraints. You should know that both types of constraints and notation are used because, as a member of the development team, as an auditor, or as a user, you will need to communicate using the methods selected by the organization with which you are working.

REA Data Models and E-R Diagrams

You now have all of the basic knowledge you need to develop effective REA models and their representations in E-R diagrams. You should be ready to start developing an integrated database model. Each of the data model segments included in Figure 6.1 (pg. 179), 6.2 (pg. 180), 6.3 (pg. 182), and 6.4 (pg. 183) are parts of REA data models.

A fundamental requirement for moving toward an event-driven model is the complete integration of data related to an organization's business events. Although the development of a full, comprehensive integrated data model for an entire organization is a major undertaking, you can use your data-modeling skills to explore the integration of two business processes: client billing and human resources.

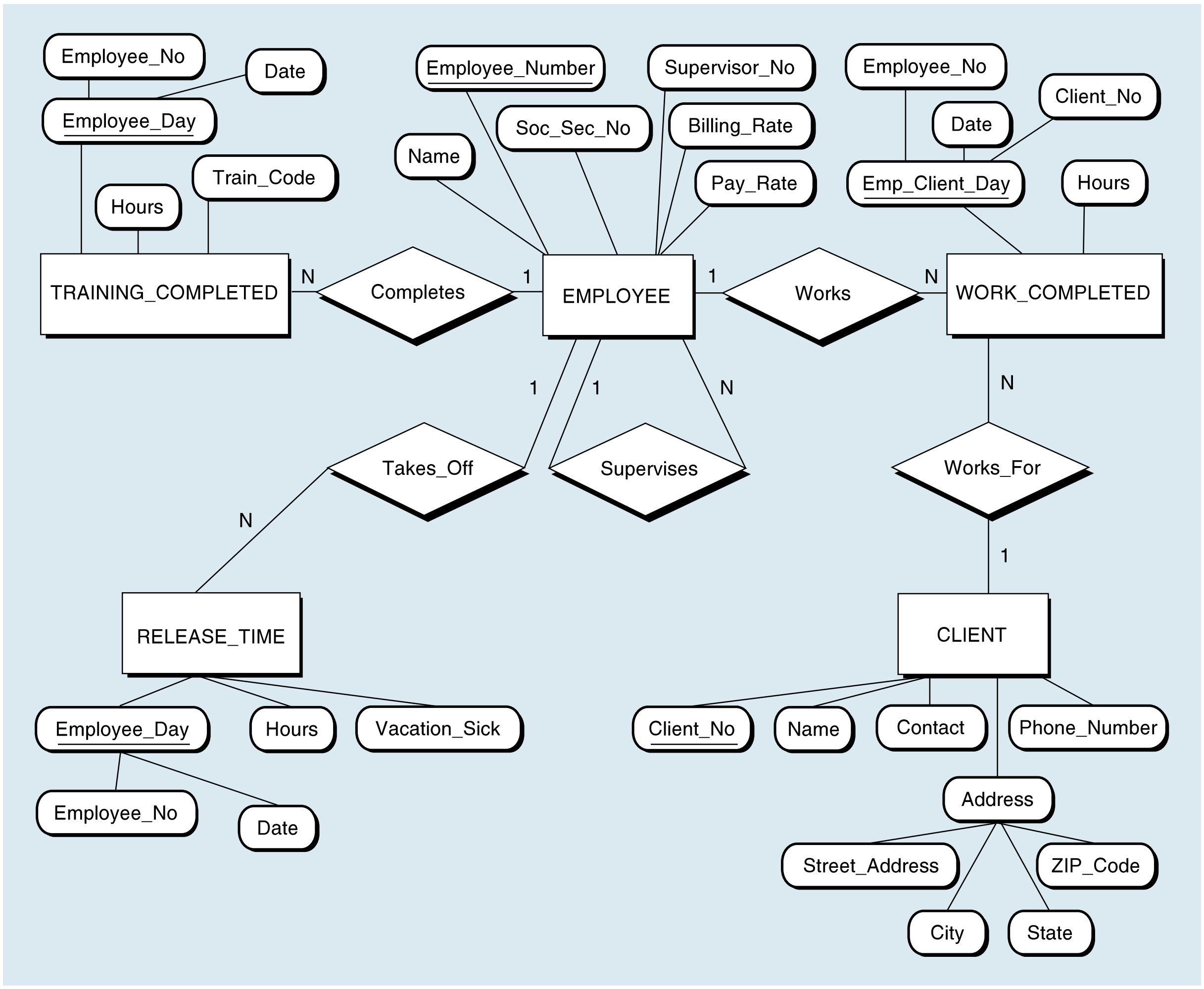
The objective in the development of an REA model is to integrate the data in a way that allows managers and other users access to the information they need to perform effectively. Figure 6.5 presents the integrated REA data model for the billing and human resources business processes. Follow along with the REA diagram as we discuss the data relationships in the following scenario.

In a service organization such as a public accounting or consultancy firm, billing clients requires that the firm track the person-hours spent by each employee who is providing service. To execute the client billing process effectively, the database must capture data about all employees who provided client services. The database must record each employee's work to a specific client. Each employee can have a different billing rate for his or her time. To meet the needs of the billing process, the database must aggregate each employee's time worked, each employee's billing rate, and sufficient information about the client to deliver the billing statement. Three entities are involved in the billing process: the agent EMPLOYEE, the agent CLIENT, and the event WORK_COMPLETED. Note in Figure 6.5 that the three entities for the billing process are linked together on the right half of the diagram. The linkages allow us to pull together information related to the employees' hours worked on a specific client, their billing rates, and the contact address for sending the billing statement.

Service businesses also are interested in tracking employee work activities as part of the human resources process. The human resources process includes payroll activities, employee education and development, and other activities. To complete the payroll process, information is needed regarding work hours completed, pay rate, vacation time, sick days, and training time. Using the REA approach, we can identify two additional entities, the events RELEASE_TIME and TRAINING_COMPLETED, which are added to the model that also includes the previously identified agent entity EMPLOYEE and event entity WORK_COMPLETED. These four entities enable the database to aggregate the information it needs to determine the employee's pay rate, hours worked, hours spent in training, and hours of sick and vacation time used.

The human resources department needs information about employee education and development so it can monitor training activities and ensure that the employee is

FIGURE 6.5 An Integrated REA Model for the Client Billing and Human Resources Processes



receiving enough continuing education to comply with state license requirements and the firm's policies. Human resources also will monitor the percentage of billable hours the employee has accumulated as a measure of job performance. To accomplish these activities, human resources must be able to link data about completed work activities and training programs to specific employees. This information can be drawn from the agent entity EMPLOYEE, the event entity TRAINING_COMPLETED, and the event entity WORK_COMPLETED. Human resources can use this information to accumulate a given employee's training record and calculate that employee's percentage of hours worked that were billable hours.

Note that Figure 6.5 shows only a small part of the overall enterprise model that integrates all information across the entire organization. The REA model effectively integrates the data required for the firm's business processes. As other business processes are examined, modeled, and integrated, the REA data model will continue to expand through an explosion of entities and relationships. Many organizations have moved toward integration of all data across the organization. These integrated enterprise

ENTERPRISE SYSTEMS

models are the foundations for implementing enterprise systems, which you learned about in Chapter 2.

In the following sections, we describe how to implement the REA data model in a relational database system. Subsequently, we examine the use of SQL to create and manipulate the database.

Relational Databases

We briefly demonstrated the use of the *relational databases* in Chapter 5. In this section, we expand our examination of relational databases and explore a few of the more technical issues. Despite a push toward *object-oriented databases*, relational database-driven legacy systems exist in most large organizations, and the effort to switch them over to object-oriented databases is cost prohibitive. Many of these **legacy systems** (that is, systems that have existed in an organization over a long period of time and developed using an organization's previous computer hardware and software platforms) have been functioning reliably for decades. As an alternative to making costly changes to object-oriented databases, relational database vendors are providing modified versions of their software that support objects within the relational structure. Relational-based DBMSs likely will remain dominant in the near future.

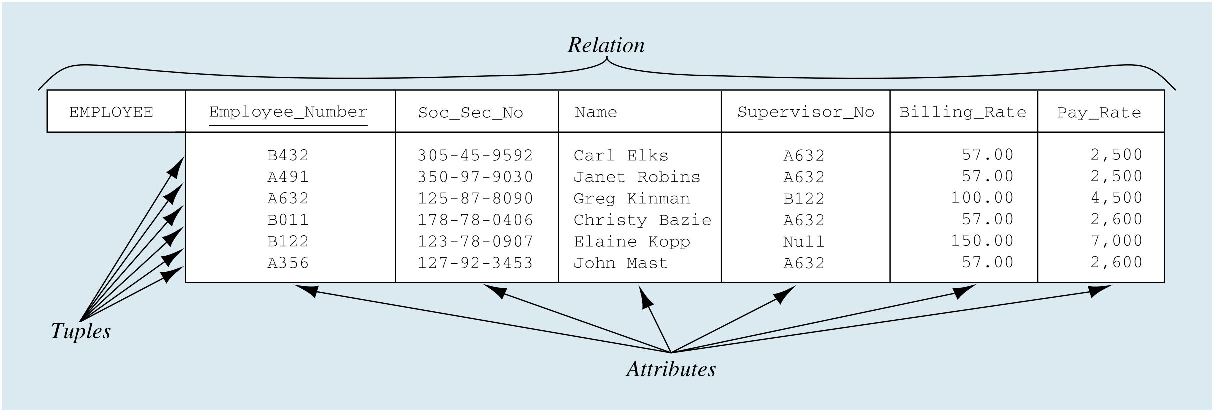
Our exploration of relational databases is divided into two segments. First, we look at the basic concepts that underlie relational databases. Then we explore the task of mapping REA data models into relational database tables and relationships.

Relational Database Concepts

Relational databases often are perceived by users as a collection of tables. This is a reasonable perception because the logical view of the data is a tabular type format referred to as a *relation*. A **relation** is a collection of data representing multiple occurrences of a resource, event, or agent. These relations correspond to the entities in the E-R model and the REA model.

Figure 6.6 displays an example relation along with labels for each of the components (or attributes) of a relation. Consistent with a tabular representation, a relation consists of rows and columns. Rows are sometimes referred to as *tuples* and columns are referred to as *attributes*. A **tuple** is a set of data that describes a single instance of the entity

FIGURE 6.6 Example of a Relation (EMPLOYEE) and Its Parts



EMPLOYEE	Employee_Number	Soc_Sec_No	Name	Supervisor_No	Billing_Rate	Pay_Rate
	B432	305-45-9592	Carl Elks	A632	57.00	2,500
	A491	350-97-9030	Janet Robins	A632	57.00	2,500
	A632	125-87-8090	Greg Kinman	B122	100.00	4,500
	B011	178-78-0406	Christy Bazie	A632	57.00	2,600
	B122	123-78-0907	Elaine Kopp	Null	150.00	7,000
	A356	127-92-3453	John Mast	A632	57.00	2,600

represented by a relation (for example, one employee is an instance of the EMPLOYEE relation). Attributes, as in an E-R model, represent an item of data that characterizes an object, event, or agent. Attributes are often called *fields*.

In viewing the relation in Figure 6.6, note that the data contained in the table do not appear to be in any particular order. In a relational database model, no ordering is given to the tuples contained within a relation. This is different from the traditional file structures and the database models that existed before the relational model that you learned about in Chapter 5. In those systems, sequence or keyed locations could be critical. The ordering of the tuples in a relational database is unimportant because the tuples are recalled from the database by matching an attribute's value with some prescribed value or through a query in which the ordering of the output could be established on any attribute (for example, a query could sort on the Pay_Rate or Billing_Rate attribute).

To identify a tuple uniquely, each tuple must be distinct from all other tuples. This means that each tuple in a relation must be identified uniquely by a single attribute or some combination of multiple attributes. In each table, a primary key is specified to identify each tuple in the relation. Notice in Figure 6.6 that Employee_Number is the primary key, which is unique for each tuple. Other attributes in the relation could serve as a key attribute. In a relation, these additional attributes are secondary keys often called **candidate keys**. Any attribute that is specified as a key attribute must have a unique value that exists for each tuple. A missing value is called a **null**, and key attributes are required to be **non-null** in every tuple in the relation. Notice that Soc_Sec_No would also be unique and could possibly be used as a candidate key, but constraints would have to be implemented that require every tuple to have a value because an employee might not have a Social Security number, especially if the firm has operations outside the United States.

CONTROLS

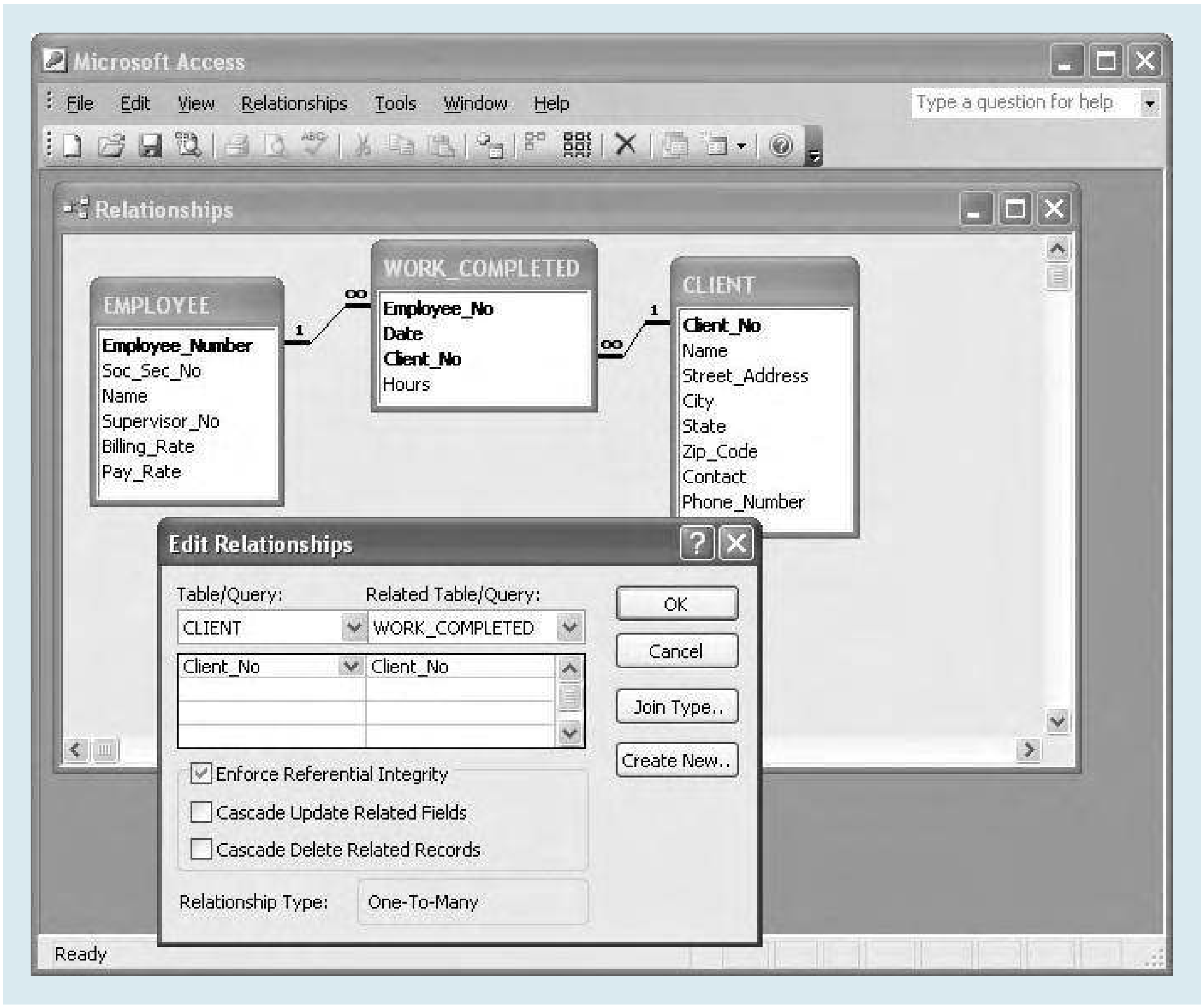
Additionally, constraints should be implemented to ensure that the *referential integrity* of the database is maintained. **Referential integrity** specifies that for every attribute value in one relation that has been specified to allow reference to another relation, the tuple being referenced must remain intact. For example, consider the relation EMPLOYEE in Figure 6.6. You might recall that EMPLOYEE was involved in a recursive relation in Figure 6.3, part (b) on pg. 182. In that recursive relation, Supervisor_No is used to reference the Employee_Number of the supervising employee. If the tuple for Greg Kinman were deleted from the database, four other employees would no longer have a valid Supervisor_No (because the Supervisor_No would be referencing a tuple that no longer exists). Thus, a *referential integrity constraint* would require the user to reassign the four employees to a new supervisor before the tuple for Greg Kinman could be deleted. Most DBMS software products have built-in mechanisms for enforcing referential integrity. For example, the Microsoft Access DBMS software provides a check box for that purpose when a designer is creating a relationship, as shown in Figure 6.7 (pg. 188).

Mapping an REA Model to a Relational DBMS

So far in this chapter, we have discussed the development of REA models and the foundations for implementing good relational database models. It is now time to put these two concepts together. This process is referred to as *mapping* an REA model onto a logical database model—in this case the relational data model.

In Chapter 5, we outlined a simple five-step process for creating tables and relationships based on an E-R diagram. We will now expand the five-step process to develop a well-constrained relational database implementation. To aid in the comparability with our original discussion, we will reintroduce each of the five steps in the context of the

FIGURE 6.7 Enforcing Referential Integrity on a Relationship in the Microsoft Access DBMS



Table/Query: CLIENT
Related Table/Query: WORK_COMPLETED

Table/Query	Related Table/Query
Client_No	Client_No

[✓] Enforce Referential Integrity
[ ] Cascade Update Related Fields
[ ] Cascade Delete Related Records

Relationship Type: One-To-Many

Source: Relationships Window, Microsoft Access 2003.

discussion on this expanded mapping methodology. Follow along as we map the REA diagram in Figure 6.5 (pg. 185) to the relational database schema in Figure 6.8.

1. **Create a separate relational table for each entity:** This is a logical starting point whenever mapping an REA model onto a relational database model. As a starting point in this process, it is generally useful to first specify the database schema before proceeding to expand the relations to account for specific tuples. Notice that each of the entities in Figure 6.5 (pg. 185) has become a relation in Figure 6.8. To complete the schema, however, steps 2 and 3 also must be completed.
2. **Determine the primary key for each of the relations:** The primary key must uniquely identify any row within the table.
3. **Determine the attributes for each of the entities:** Note in Figure 6.5 (pg. 185) that a complete REA model includes specification of all attributes, including the key attribute. This eliminates the need to do this during development of the relations. Rather, the focus is on step 2 and is simply a matter of determining how to implement the prescribed key attribute within a relation. With a single attribute

FIGURE 6.8 Schema for the Client Billing and Human Resources Portion of the Database

CLIENT

Client_No	Name	Street_Address	City	State	ZIP_Code	Contact	Phone_Number

WORK_COMPLETED

Employee_No	Date	Client_No	Hours

EMPLOYEE

Employee_Number	Soc_Sec_No	Name	Supervisor_No	Billing_Rate	Pay_Rate

TRAINING_COMPLETED

Employee_No	Date	Hours	Train_Code

RELEASE_TIME

Employee_No	Date	Hours	Vacation_Sick

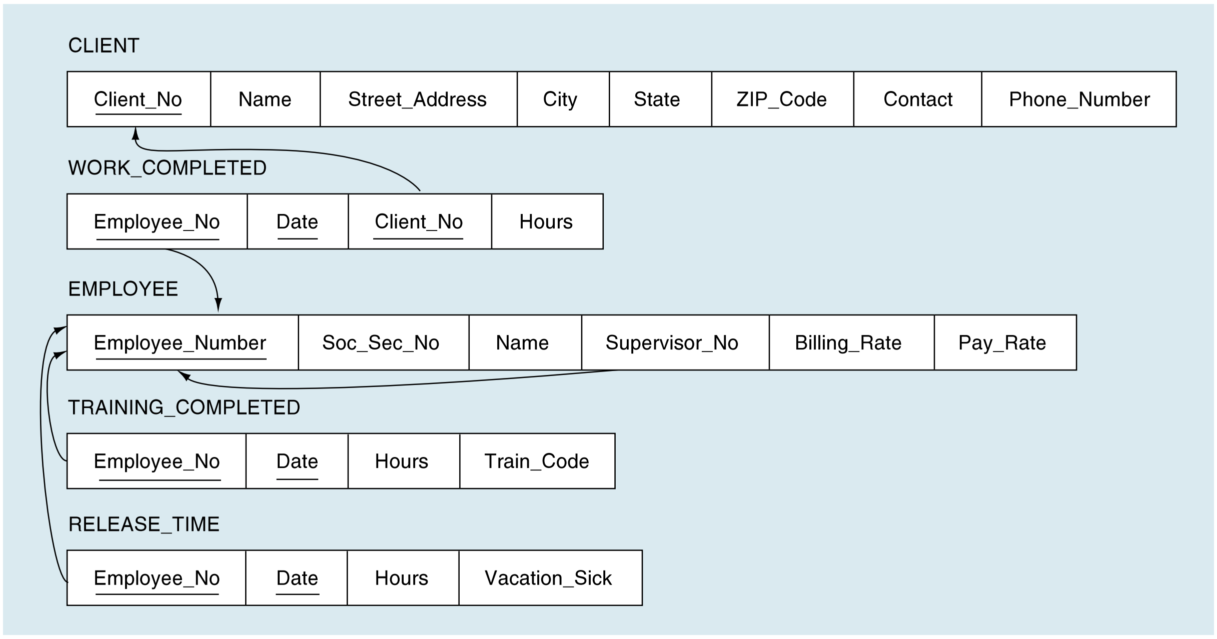
specified as the key, the key attribute specified in the REA model is matched to the corresponding attribute in the relation (for example, Employee_Number in the EMPLOYEE agent entity shown in Figure 6.5 and the EMPLOYEE agent relation in Figure 6.8).

To create a *composite primary key*, you simply break the key down into its component *subattributes*. For instance, in the implementation of the WORK_COMPLETED event relation, Employee_No, Date, and Client_No are three distinct attributes in the relation, but also combine to form the composite primary key. Similarly, in the event relations TRAINING_COMPLETED and RELEASE_TIME, attributes Employee_No and Date combine to form the composite primary key Employee_Day. The completed schema, reflecting the attributes and primary keys, is presented in Figure 6.8. Note the direct mapping between the entities and attributes in the REA model and the relations and attributes, respectively, in the relational schema.

4. **Implement the relationships among the entities by ensuring that the primary key in one table also exists as an attribute in every table for which there is a relationship specified in the REA diagram:** With the availability of the full REA model, the mapping of the relationships in the model to the relationships in the relational schema is straightforward. References to the key attributes of one entity are captured by including a corresponding attribute in the other entity that participates in the relationship. All of the relationships in this example are 1:N relationships, which simplifies the process. Let's consider how the different degrees of relationships (i.e., cardinality constraints) can affect the mapping of relationships to the schema.

   - One-to-many (1:N or N:1) relationships are implemented by including the primary key of the table on the one side of the relationship as an attribute in the table on the many side of the relationship. This is the situation we have for all the relationships in Figure 6.5 (pg. 185). The links between these relations in the schema are drawn in Figure 6.9. Note that Client_No in CLIENT and Employee_Number in

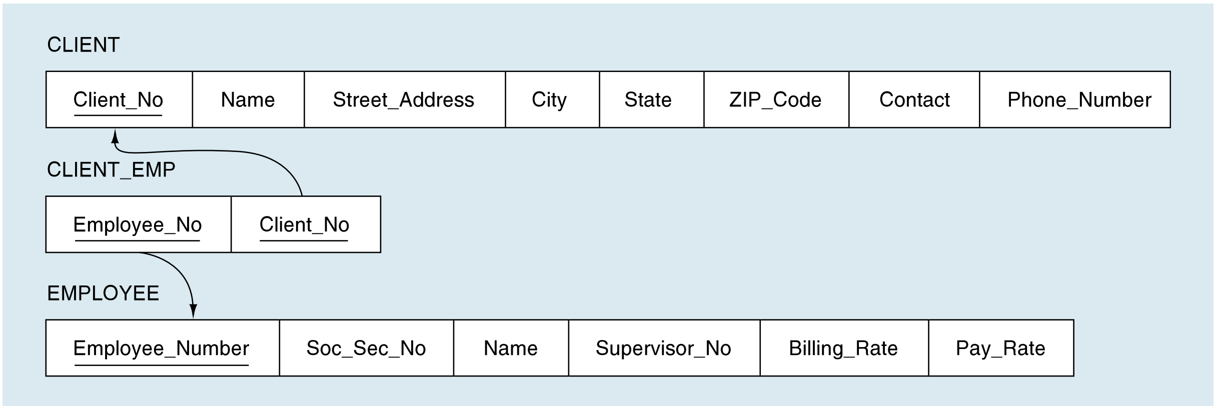
FIGURE 6.9 Referential Constraints for the Relational Schema



EMPLOYEE provide the links to WORK_COMPLETED. Similarly, Employee_ Number in EMPLOYEE provides links to TRAINING_ COMPLETED and RELEASE_TIME. The recursive relationship with EMPLOYEE uses Supervisor_No to identify the correct EMPLOYEE as the supervisor.

- One-to-one (1:1) relationships are even easier. You can follow the same steps used for 1:N relationships, but you can start with either table. For example, if currently one entry in WORK_COMPLETED is sufficient to finish any client project, then a 1:1 relationship exists between WORK_COMPLETED and CLIENT. In this situation, we could still select the Client_No in CLIENT to establish the primary key (see Figure 6.9). Starting with CLIENT has an advantage; if a client engagement in the future might require more than entry in WORK_ COMPLETED to finish or more than one employee to complete, using Client_ No in the CLIENT table would still work (that is, it would form the *many* dimension shown for the relationship Works_For in Figure 6.5, pg. 185).
- Many-to-many (M:N) relationships are implemented by creating a new relation whose primary key is a composite of the primary keys of the relations to be linked. In our model, we do not have any M:N relationships, but if we had not needed to record the Date and Hours in the WORK_COMPLETED entity, that entity would not have existed. Still, we would need a relationship between the EMPLOYEE and CLIENT entities, which would then be an M:N relationship. This creates problems because tables that have been normalized (as discussed in Chapter 5) cannot store multiple client numbers in a single EMPLOYEE tuple. Similarly, a single CLIENT tuple cannot store multiple employee numbers. In that situation, we would need to develop a relation to link the EMPLOYEE and CLIENT relations (see Figure 6.10). This new relation would have a composite key consisting of Employee_Number from EMPLOYEE and Client_No from CLIENT—similar to the composite key in the existing relation, WORK_ COMPLETED (see Figure 6.8 on pg. 189).

FIGURE 6.10 Linking Two Relations in a Many-to-Many Relationship



- Beyond concerns over meeting the constraint requirements for primary keys, we also must ensure adherence to the referential integrity constraints. We identify the referential integrity constraints by locating the corresponding attribute in each relation that is linked via a relationship. We then determine which of the relations contain the tuple that—if the reference attribute were deleted or changed—would jeopardize the integrity of the database. In Figure 6.9, the arrow represents referential integrity constraints; the destination of the arrow is the attribute that must be controlled to achieve referential integrity. In other words, changing the attribute to which the arrow points could cause an attribute not to have a matching value in the attribute at the source of the arrow. To achieve referential integrity, constraints should be established that assure Employee_Number is not altered or deleted for any EMPLOYEE until the referencing attribute values for the Employee_No attributes in WORK_COMPLETED, TRAINING_COMPLETED, and RELEASE_TIME have first been corrected. A similar constraint should be placed on Client_No in CLIENT until Client_No has been corrected in WORK_COMPLETED.

5. **Determine the attributes, if any, for each of the relationship tables:** Again, in the extended version of the REA model, the attributes map directly to the relations. The implementation of the schema is shown in Figure 6.11 (pg. 192).

SQL: A Relational Database Query Language

In this section, you will learn how SQL can be used to create a database, store data in the new database, and manipulate the data stored in the database.

SQL[2] is a powerful database language that can be used to define database systems, query the database for information, generate reports from the database, and access databases from within programs using embedded SQL commands. It has become the *de facto* standard database language—evidenced by continual efforts by the industry to provide standardization guidelines for vendors and the number of variations of the

2 SQL is pronounced two different ways because of an interesting fact in the language's history. The original implementation of the language many years ago was named "SEQUEL," but another company was already selling a product with that name and asserted its trademark rights to the name. So, the query language's name was changed to "SQL." As a result, older database designers usually pronounce the name "sequel." Younger designers are more likely to pronounce the name "ess-cue-el."

FIGURE 6.11 Implementation of the Relational Schema

EMPLOYEE	Employee_Number	Soc_Sec_No	Name	Supervisor_No	Billing_Rate	Pay_Rate
	B432	305-45-9592	Carl Elks	A632	57.00	2,500
	A491	350-97-9030	Janet Robins	A632	57.00	2,500
	A632	125-87-8090	Greg Kinman	B122	100.00	4,500
	B011	178-78-0406	Christy Bazie	A632	57.00	2,600
	B122	123-78-0907	Elaine Kopp	Null	150.00	7,000
	A356	127-92-3453	John Mast	A632	57.00	2,600

TRAINING_COMPLETED	Employee_No	Date	Hour	Train_Code
	A356	070823	8	32
	B011	070823	8	32
	B432	070823	8	32
	A491	070823	8	32
	A356	070824	8	32
	B011	070824	8	32
	B432	070824	8	32
	A491	070824	8	32
	A356	070825	8	32
	B011	070825	8	32
	B432	070825	8	32
	A491	070825	8	32

RELEASE_TIME	Employee_No	Date	Hour	Vacation_Sick
	B011	070826	8	V
	B011	070827	8	V

CLIENT	Client_No	Name	Street_Address	City	State	ZIP_Code	Contact	Phone_Number
	A12345	Arnold, LLP	11 Nayatt Dr.	Barrington	RI	02806	V. Arnold	401-792-8341
	F11555	Fleet Services	10 Mission Rd.	Providence	RI	02835	R. Grass	401-774-9843
	H12456	Hasbro, Inc.	4516 Burton Pike	Providence	RI	02844	T. Bayers	401-837-2132

WORK_COMPLETED	Employee_No	Client_No	Date	Hours
	B122	F11555	070823	8
	A632	F11555	070823	8
	B122	F11555	070824	8
	A632	F11555	070824	8
	B122	F11555	070825	8
	A632	F11555	070825	8
	B122	H12456	070826	8
	A632	H12456	070826	8
	A356	F11555	070826	8
	B432	H12456	070826	8
	A491	H12456	070826	8
	B122	F11555	070827	8
	A632	H12456	070827	8
	A356	F11555	070827	8
	B432	H12456	070827	8
	A491	H12456	070827	8

language that exist in databases from supercomputers to personal computers. SQL has become so critical to many business organizations that more and more software is being developed to provide intelligent interfaces that help the user generate queries more quickly. Using such an interface, a user enters a description of the output desired, and the system generates an SQL query. However, these query utilities can be risky to use because they do not always generate the SQL query intended; therefore, it is imperative that the user thoroughly understands the queries to ensure that the intended data is properly extracted from the database.

SQL is increasingly becoming a survival tool among accounting and business professionals. After you complete your study of the material presented here, you should have the basic skills necessary to develop your own databases and retrieve data from many different databases. Our discussion of SQL will begin with the commands necessary to create databases. From there, we will explore the basic querying commands that will allow us to update the database, retrieve data, and generate reports.

Constructing Relational Databases

SQL was first introduced as a query language named Structured English Query Language (SEQUEL). Much like this early name suggests, the language is an attempt to use somewhat normal English language as its commands. As such, SQL uses terms such as *table*, *row*, and *column* to describe relations during creation and manipulation of relational databases. Similarly, most commands use easily recognizable names.

The first command of interest in creating the database structure is the CREATE command, which we will use to create the relations that form the database structure. Browse the queries in Figure 6.12 before reading the following discussion.

Use Figure 6.12 to follow the steps for creating a database structure with SQL as discussed next. To create a relation, follow these steps:

1. Assign the relation a name (which we have already done for our relations in defining the schema in Figure 6.8, pg. 189).
2. Assign each attribute a name. Again, in the schema demonstrated in Figure 6.8, we have already given our attributes names.

FIGURE 6.12 SQL Commands for Creating Database Relations

```
CREATE TABLE EMPLOYEE              (Employee_Number   Char(4)         NOT NULL,
                                    Soc_Sec_No        Char(11)        NOT NULL,
                                    Name              VarChar(25)     NOT NULL,
                                    Supervisor_No     Char(11),
                                    Billing_Rate      Decimal(5,2),
                                    Pay_Rate          Decimal(7,2);

CREATE TABLE TRAINING_COMPLETED    (Employee_No       Char(11)        NOT NULL,
                                    Date              Integer         NOT NULL,
                                    Hour              Integer,
                                    Train_Code        Integer);

CREATE TABLE RELEASE_TIME          (Employee_No       Char(11)        NOT NULL,
                                    Date              Integer         NOT NULL,
                                    Hour              Integer,
                                    Vacation_Sick     Char(1));

CREATE TABLE CLIENT                (Client_No         Char(6)         NOT NULL,
                                    Name              VarChar(25),
                                    Street_Address    VarChar(30),
                                    City              VarChar(15),
                                    State             Char(2),
                                    ZIP_Code          Integer,
                                    Contact           VarChar(25)     NOT NULL,
                                    Phone_Number      VarChar(12));

CREATE TABLE WORK_COMPLETED        (Employee_No       Char(11)        NOT NULL,
                                    Client_No         Char(6)         NOT NULL,
                                    Date              Integer         NOT NULL);

ALTER TABLE WORK_COMPLETED         (Hours             Integer);
```

3. Specify the data type for each attribute. Data type descriptions generally include some combination of alphanumeric (e.g., with letters and/or symbols) or numeric values. Alphanumeric types include CHAR (for fixed-length strings) and VARCHAR (for varying length alphanumeric strings). Numeric data types include INTEGER, FLOAT (which has a floating decimal point), and DECIMAL (where the number of digits both left and right of the decimal point are fixed and defined).

CONTROLS

4. Specify constraints, when appropriate, on the attributes. Most notably, we need to make sure that the primary key values are not left empty (i.e., null); otherwise, there will be no key value by which to identify and pull the tuple's record from the database. We may want to require that other attributes be assigned some value rather than having the option of being null. In each of these cases, we can assign a value of NOT NULL as the constraint.

The previous conventions are applied in Figure 6.12 (pg. 193) to generate the relations specified in our earlier defined schema in Figure 6.8 (pg. 189). Carefully study how the various conventions have been applied. For instance, note in the EMPLOYEE relation that we have assigned a fixed-length character value (11 characters) for the attribute Soc_Sec_No. This accommodates the use of the hyphens in the number. In many cases, a company might choose to omit the hyphens and even reflect the Social Security number as a numeric. Also, note that despite our longest employee name currently being 13 characters, we have allowed a variable length alphanumeric field of 25 characters to accommodate longer names that may be entered in the future. Similarly, additional size has been built into the specifications for Billing_Rate and Pay_Rate to accommodate future inflationary effects on the two rates.

At the bottom of Figure 6.12, note that we demonstrate a new command—ALTER. ALTER is SQL's way of recognizing that we may not always get the permanent design of a relation right the first time. In this way, additional attributes can be added to a relation in the future. In our case, we accidentally left off the last attribute for WORK_COMPLETED when issuing the CREATE command and came back to add the missing attribute. Note that adding an attribute is fairly easy.

Before going on to updating the data, we should explain how to delete a table as well. If you have a relation you decide you no longer need, or that needs to be completely recreated from scratch rather than altering, you can drop a relation very quickly. The DROP command is as follows:

```
DROP TABLE [table name]
```

For instance, if we wanted to drop the WORK_COMPLETED table, we would enter the command:

```
DROP TABLE WORK_COMPLETED
```

Updating the Database

Data can be changed in the database in three ways. Our first concern is loading data into the structure we just created with the CREATE command. This is accomplished using the INSERT command to add new tuples to a relation. In the future, a user might want to DELETE or UPDATE the data stored within relations, and we will review these commands in this section as well.

The INSERT command is used to add a single tuple to an existing relation. Hence, when you create a relational database using SQL, you must first use the CREATE command to generate the structure of the relation and then use the INSERT command to enter the current data into the structure. The INSERT command in its simplest form only requires the user to specify the SQL table and the values to be inserted for each

FIGURE 6.13 SQL Commands to Add Data to the Database

```
INSERT INTO      EMPLOYEE
VALUES           ('B432','305-45-9592','Carl Elks','125-87-8090',57,2500)

INSERT INTO      EMPLOYEE
VALUES           ('A491','350-97-9030','Janet Robins','125-87-5090',57,2500)

INSERT INTO      EMPLOYEE
VALUES           ('A632','125-87-8090','Greg Kinman','123-78-0907',100,4500)

INSERT INTO      EMPLOYEE
VALUES           ('B011','178-78-0406','Christy Bazie','125-87-8090',57,2600)

INSERT INTO      EMPLOYEE (Employee_Number, Soc_Sec_No, Name, Billing_Rate, Pay_Rate
VALUES           ('B122','123-78-0907','Elaine Kopp',150,7000)

INSERT INTO      EMPLOYEE
VALUES           ('A356','127-92-3453','John Mast','125-87-8090',57,2600)

INSERT INTO      TRAINING_COMPLETED
VALUES           ('A356',990823,8,32)
     :                  :     :     :
     :                  :     :     :

INSERT INTO      RELEASE_TIME
VALUES           ('B011',990826,8,'V')
     :                  :     :     :
     :                  :     :     :

INSERT INTO      CLIENT
VALUES           ('A12345','Arnold,LLP','11 Nayatt Dr.','Barrington','RI',02806,
                 'V. Arnold','401-792-8341')
     :                  :     :     :
     :                  :     :     :

INSERT INTO      WORK_COMPLETED
VALUES           ('B122','F11555',990823,8)
     :                  :     :     :
     :                  :     :     :
```

attribute if a value is provided for every attribute. This simple form of the INSERT command is demonstrated in Figure 6.13 by the command to enter the first tuple into the EMPLOYEE table—that is, the values for Carl Elks. This form of the command also is demonstrated in Figure 6.13 for entering the first record of the remaining tables.

If values are not entered for all attributes for a given tuple, then the INSERT command must be specified more clearly. Namely, when the table is specified in the INSERT command, the attributes for which values are being provided must be specified. If you study Figure 6.13 again, you will notice that when we enter the fifth EMPLOYEE tuple, we specify the attributes to receive values because Elaine Kopp, as the top-level manager, has no supervisor specified—that is, Supervisor_No is NULL.

CONTROLS

Before leaving the INSERT command, note that you can omit values for selected attributes of a tuple if, and only if, a database constraint has not been set to NOT NULL when creating the table. For instance, if you look back to Figure 6.12 (pg. 193) where we issued the CREATE commands, you will notice that a value must be entered for Employee_No in every instance of EMPLOYEE. Hence, if we had a new employee who did not have a Social Security number, we would need to assign that employee a temporary number before we could INSERT the employee into the database.

The DELETE command is, of course, the flip side of the INSERT command and is the method by which we delete a tuple from a relation. The DELETE command requires specification of the table name and inclusion of a WHERE condition, which is used to identify the unique tuple(s) for deletion. For instance, if the EMPLOYEE named Elaine Kopp decided to leave the firm, we could enter the following command to delete her from the EMPLOYEE table:

```
DELETE FROM   EMPLOYEE
WHERE         Employee_Number ='B122'
```

CONTROLS

Notice that we use the *primary key* (Employee_No) to identify the tuple for deletion. The use of the DELETE command demonstrates one of the weaknesses in most forms of SQL—the database will not enforce referential integrity constraints. If we delete Elaine Kopp from the database, we now have a number of other relations with instances of attributes referring to what would now be a deleted record (e.g., all relations would have referential integrity problems other than CLIENT, including the recursive relationship on EMPLOYEE). Most implementations of SQL will require the user to manage referential integrity rather than providing the means for implementing constraints through the database. Most DBMSs available today that include SQL also include mechanisms within the DBMS itself (and not within the SQL function) for establishing and enforcing referential integrity. We depicted one example of such a mechanism in Figure 6.7 (pg. 188), which showed the procedure for enforcing referential integrity in Microsoft Access 2003.

The last command of interest in this segment is UPDATE. The UPDATE command is used when we want to change one or more attribute values for one or more tuples in a table. To accomplish a change of an attribute value, the UPDATE command must be able to identify the table with the value to be updated, the new values to be placed in the database, and the conditions for identifying the correct tuple for UPDATE. To make the change, we identify the tuple using the WHERE condition we just learned for deletion, and we change the existing values by using a SET command to set the new values for the database.

For example purposes, let's assume that our firm has decided to give Elaine Kopp a raise so that she will stay as an employee. We need to place her new Pay_Rate of $7,500 in her record in the database. To accomplish this, we execute the following UPDATE command:

```
UPDATE       EMPLOYEE
SET          Pay_Rate=7500
WHERE        Employee_Number ='B122'
```

Execution of the SQL command first finds the matching Employee_Number, deletes the existing attribute value (i.e., 7000), and enters the new value of 7500 into the Pay_Rate for the matching employee.

Basic Querying Commands

Queries of the database are driven by SELECT commands that allow us to develop elaborate conditions for narrowing our data search to a very narrow view. In their simplest form, SELECT commands retrieve the values for a list of attributes from the tuples of a single relation. In their most complex form, SELECT commands allow us to join data across multiple tables to link specific pieces of information that are of interest. In the following discussion, we outline the foundations from which such complex queries can be generated.

The SELECT statement consists of three parts: (1) a list of attributes that we want to SELECT from the database, (2) a list of tables where these attributes can be found, and (3) a WHERE clause that sets the conditions under which attribute values are to be retrieved. For instance, if we wanted to retrieve a list of all employees with a pay rate over $3,000, we would issue the following SELECT command:

```
SELECT      Name
FROM        EMPLOYEE
WHERE       Pay_Rate>3000
```

SQL would first locate the table named EMPLOYEE and then search all the tuples to identify any with a Pay_Rate greater than 3000. From the identified tuples, the SELECT attributes are extracted, and the values are returned to the screen. The following are the results of this query based on the values shown in Figure 6.11 (pg. 192):

```
Greg Kinman
Elaine Kopp
```

Often, a user will want to aggregate data from more than one table. If we go back to our original reasons for designing this database, one goal was to facilitate the billing process. To process client billing, we need a combination of data from the EMPLOYEE, WORK_COMPLETED, and CLIENT relations. From the EMPLOYEE relation, we need information on Billing_Rate for each record of WORK_COMPLETED. From the CLIENT relation, we need data for Name, Street_Address, City, State, ZIP_Code, and Contact to send the bill to the correct client. From the WORK_COMPLETED relation, we will match the previously noted information with the records of employees' completed work to provide line item detail on the date and hours of work completed. The SELECT command to aggregate the data for the billing for Hasbro, Inc., could be as follows:

```
SELECT     CLIENT.Name, Street_Address, City,
           State, ZIP_Code, Contact, Date, Hours,
           Billing_Rate
FROM       EMPLOYEE, CLIENT, WORK_COMPLETED
WHERE      Employee_Number=Employee_No AND
           CLIENT.Client_No= 'H12456' AND
           WORK_COMPLETED.Client_No='H12456'
```

Note the new conventions we have added in this query. First, in the SELECT attribute list, we used CLIENT.Name to identify the client name. The table name is attached to the front of the attribute name because the attribute Name is ambiguous—that is, SQL would not be able to tell if we wanted the Name attribute from the EMPLOYEE or the CLIENT relation. We did the same thing in the WHERE clause for Client_No. In fact, the rule is that any time we reference an attribute in a SELECT statement and the attribute name appears in more than one relation in the table list specified on the FROM list, we must use the table name extension as a prefix to the attribute name.

The other convention is the use of AND in the WHERE clause. We use connectors such as AND to link multiple conditions that must all be met when selecting tuples from one or more tables. The most common other connector is the OR connector, which allows for a tuple to be selected if either condition in the WHERE clause is true.

One final note on querying regards the output that comes from the previous query. As shown in Figure 6.14, this query would attach the client name and address to every line of output generated when a row of WORK_COMPLETED is identified. Hence, for the seven lines of completed work that are returned in the query, there is a repetitive return of the client name and address each time. In this case, where the name, address, and contact person will be the same for every query response, it makes more sense to split the query into two parts. First, we identify the client billing address information.

FIGURE 6.14 Generation of Client Billing Information (Single-Query Approach)

Name	Street_Address	City	State	ZIP_Code	Contact	Date	Hours	Billing_Rate
Hasbro, Inc.	4516 Burton Pike	Providence	RI	02844	T. Bayers	070826	8	150.00
Hasbro, Inc.	4516 Burton Pike	Providence	RI	02844	T. Bayers	070826	8	100.00
Hasbro, Inc.	4516 Burton Pike	Providence	RI	02844	T. Bayers	070826	8	57.00
Hasbro, Inc.	4516 Burton Pike	Providence	RI	02844	T. Bayers	070826	8	57.00
Hasbro, Inc.	4516 Burton Pike	Providence	RI	02844	T. Bayers	070827	8	100.00
Hasbro, Inc.	4516 Burton Pike	Providence	RI	02844	T. Bayers	070827	8	57.00
Hasbro, Inc.	4516 Burton Pike	Providence	RI	02844	T. Bayers	070827	8	57.00

Then, in a second query, we pull out the multiple lines of billing information for completed work. This two-query approach and the related output are demonstrated in Figure 6.15.

Generating Standard Reports

To this point, we have focused our discussion on designing a logical relational database model that efficiently stores the data in our database. This allows the user to easily manipulate the base-level tables to generate information on an *ad hoc* (as needed) basis. This is the most effective way to provide data availability to users of the database when their information needs change on an ongoing basis.

However, many decisions are not *ad hoc* but are made on an ongoing basis as part of regular business operations. For instance, the billing information we previously discussed will need to be performed on a regular basis. It would be much easier if the information required for the second part of the query in Figure 6.15 was already

FIGURE 6.15 Generation of Client Billing Information (Double-Query Approach)

```
SELECT        Name, Street_Address, City, State, ZIP_Code, Contact
FROM          CLIENT
WHERE         Client_No='H12456'
```

Output

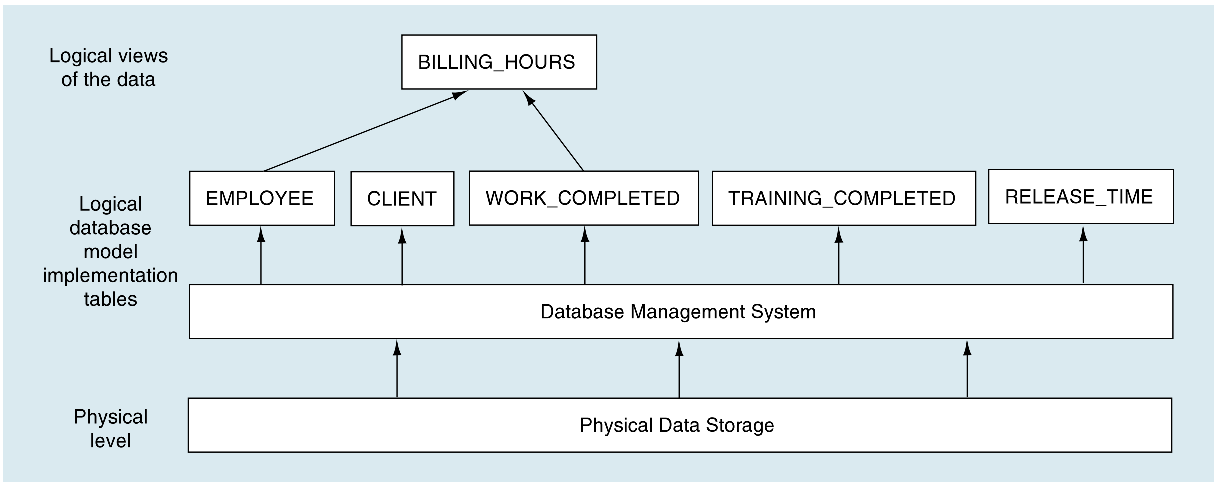
Name	Street_Address	City	State	ZIP_Code	Contact
Hasbro, Inc.	4516 Burton Pike	Providence	RI	02844	T. Bayers

```
SELECT        Date, Hours, Billing_Rate
FROM          EMPLOYEE, WORK_COMPLETED
WHERE         Employee_Number=Employee_No AND Client_No='H12456'
```

Output

Date	Hours	Billing_Rate
070826	8	150.00
070826	8	100.00
070826	8	57.00
070826	8	57.00
070827	8	100.00
070827	8	57.00
070827	8	57.00

FIGURE 6.16 Schema for the Client Billing and Human Resources Portion of the Database



aggregated in a single source table within the database—but only if the data were not replicated, causing a redundancy. (Recall from Chapter 5, our discussion of the negative effects of data redundancy and our arguments for using DBMSs as a solution to the problem.)

In a relational database model, we can do just that. We can actually create *views* of the data that look like additional tables but are just alternative ways to view the data that already exists in the database. The data are not copied to a second physical location in the database. Instead, a view creates the appearance of a different set of tables for the user in the format the user wants to see. We graphically present this concept in Figure 6.16.

Note that the data at the lowest level (i.e., the physical level) are stored in a manner that users need not understand. The DBMS handles all the communication between the relational tables that are used to implement the database model and the physical storage of the data. After the database model has been implemented into relational tables, we can then create as many views as desired, based on some combination of attributes extracted from these base relations. These views do not replicate the data; they only provide alternative ways to examine the data.

To create a view of the data contained in the database tables, we simply tell SQL to do just that. We CREATE VIEW by using a SELECT statement, just as we did when querying the database, but in this case, we will be storing the results in a permanent view. Permanent views can be deleted using the DROP VIEW statement. In Figure 6.17 (pg. 200), we issue the CREATE VIEW command to create the view BILLING_HOURS. Note after the query in Figure 6.17, the data that would be shown in the view based on the data from Figure 6.11 (pg. 192).

After a view has been created, the view can be used in other SQL queries just the same as any of the tables that were created. Notice in Figure 6.17 that we have created a view that represents all employees' hours spent working for clients, matched with the employee billing rates. We can now issue a query directly to this view to aggregate billing information for a specific client (see Figure 6.18), instead of having to use the more complex query in Figure 6.15. The result is exactly the same for the new query in Figure 6.18 as what we received from the second query in Figure 6.15.

FIGURE 6.17 Creating a View of the Client Billing Detail with SQL

SQL commands

```
CREATE VIEW    BILLING_HOURS
AS SELECT      Employee_No, Client_No, Date, Hours, Billing_Rate
FROM           EMPLOYEE, WORK_COMPLETED
WHERE          Employee_No=Employee_Number
```

(Note: Employee_No would not have to be included except that it is needed to form a composite key for the view, BILLING_HOURS.)

Data in the view BILLING_HOURS:

BILLING_HOURS	Employee_No	Client_No	Date	Hours	Billing_Rate
	B122	F11555	070823	8	150
	A632	F11555	070823	8	100
	B122	F11555	070824	8	150
	A632	F11555	070824	8	100
	B122	F11555	070825	8	150
	A632	F11555	070825	8	100
	B122	H12456	070826	8	150
	A632	H12456	070826	8	100
	A356	F11555	070826	8	57
	B432	H12456	070826	8	57
	A491	H12456	070826	8	57
	B122	F11555	070827	8	150
	A632	H12456	070827	8	100
	A356	F11555	070827	8	57
	B432	H12456	070827	8	57
	B491	H12456	070827	8	57

FIGURE 6.18 Query to Extract Client Billing Data for Hasbro, Inc.

```
SELECT         Date, Hours, Billing_Rate
FROM           BILLING_HOURS
WHERE          Client_No='H12456'
```

Output

Date	Hours	Billing_Rate
070826	8	150.00
070826	8	100.00
070826	8	57.00
070826	8	57.00
070827	8	100.00
070827	8	57.00
070827	8	57.00

Views can be used to generate standard reports or to provide all the data necessary to create such reports. This simplifies the process for users of the database and reduces the risk of the user producing incorrect queries. In other cases, an experienced user may decide to create views on a temporary basis for work on a given project, and after the project is completed, drop the view. After the database design is complete, tables generally remain constant in a database except for maintenance and additions prompted by changes in business processes. Views, on the other hand, are much more likely to come and go in the database.

SUMMARY

As the information needs and wants of users continue to escalate, database integration has clearly become the norm rather than the exception. The focus is no longer on finding places in the business where implementing databases is useful. Instead, the focus has shifted to finding ways to integrate as much of the organization's data as possible into a single logical database. This shift in focus offers new opportunities and challenges for accountants as they strive to continue as the primary information providers in organizations.

With these opportunities and challenges come huge responsibilities. The very lifeblood of an organization today often resides in a database that contains all the organization's information. If the database is destroyed and cannot be recovered, the organization might not survive in today's business environment. Likewise, if competitors or other unauthorized persons gain access to the data, the organization's capability to compete can be jeopardized.

Safeguarding data while providing information to users who need it is not a simple task. In Chapters 7, 8, and Chapter 9, our discussion will shift to the issues surrounding data reliability, access, and security. You will learn about procedures that organizations implement to ensure the reliability of information that is updated or added to the database. You will also learn about safeguarding the data and maintaining backups of data so that if something should happen to the database, it can be recovered in a timely manner to allow the organization to carry on its operations. These are truly challenging but exciting times for accountants who are prepared to operate in an information systems environment.

KEY TERMS

semantics	recursive relationship	candidate keys
instance	participation constraint	null
attribute	legacy systems	non-null
composite attributes	relation	referential integrity
key attribute	tuple	

REVIEW QUESTIONS

RQ 6-1 What is an entity? Distinguish an *entity* and an *instance* of an entity.

RQ 6-2 What is an attribute?

RQ 6-3 What is a key attribute? What coding techniques can be used to create good primary key attributes?

RQ 6-4 What is a relationship?

RQ 6-5 What are the three steps in the strategy to effectively identify all the relationships that should be included in a model? What is the most common way to gather information about the relationships?

RQ 6-6 What are the characteristics of recursive relationships that distinguish them from other types of relationships?

RQ 6-7 Describe a situation in which one entity has minimum participation in its relationship with another entity.

RQ 6-8 How can an REA model help an organization improve the level of data integration it achieves across multiple business processes?

RQ 6-9 What is a relation? What is a tuple?

RQ 6-10 What is referential integrity? Give an example of how it may be implemented in a DBMS?

RQ 6-11 What is a composite primary key?

RQ 6-12 What is the difference in implementation of a one-to-many and a one-to-one relationship in a relational database model?

RQ 6-13 What is SQL? In general terms, how can it be used?

RQ 6-14 What are the three ways to use SQL to change data in a database?

RQ 6-15 What are the three components of the SQL SELECT statement?

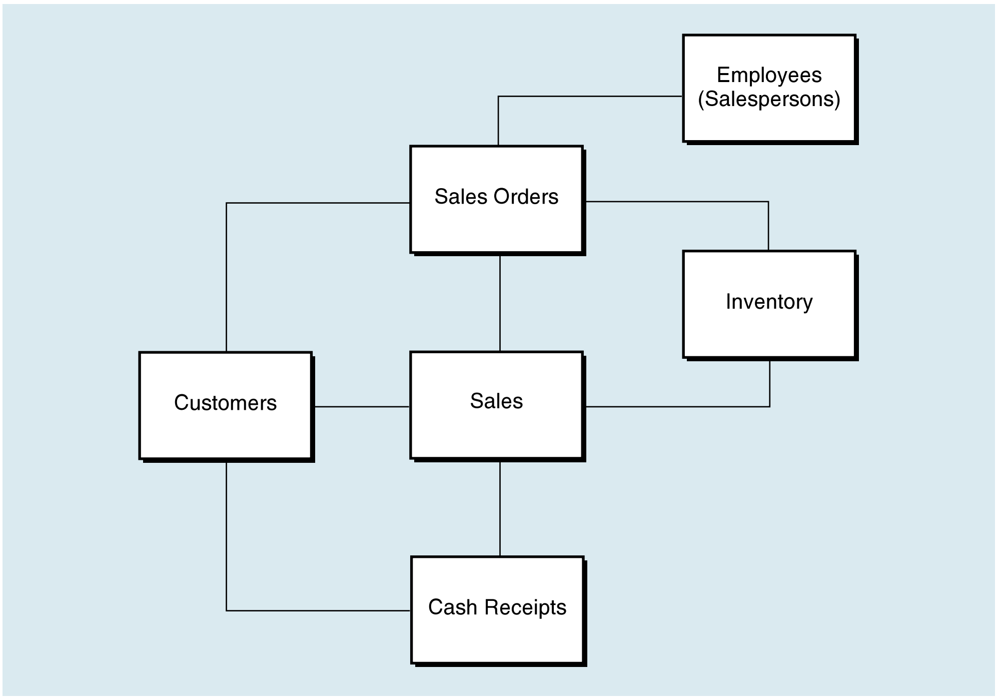
RQ 6-16 Compare and contrast *ad hoc* and *ongoing* decisions.

RQ 6-17 What is a view? Why do we create views?

Discussion Questions

DQ 6-1 Examine Figure 6.19, which contains the REA model for Salem Industrial Supply (SIS). The model is partially completed; it includes all entities and relationships, but it does not include cardinalities or descriptions of the relationships (which would appear in diamonds on the connecting lines between entities). SIS sells replacement

FIGURE 6.19 Partially Completed REA Model of the Salem Industrial Supply Sales Business Process



parts for packaging machinery to companies in several states. SIS accepts orders over the telephone, via fax, and by mail. When an order arrives, one of the salespersons enters it as a sales order. The sales order includes the customer's name and a list of the inventory items that the customer wants to purchase. This inventory list includes the quantity of each inventory item and the price at which SIS is currently selling the item. When the order is ready to ship, SIS completes an invoice and records the sale. Sometimes, inventory items that a customer has ordered are not in stock. In those cases, SIS will ship partial orders. Customers are expected to pay their invoices within 30 days. Most customers do pay on time; however, some customers make partial payments over two or more months. List each entity in the REA model, and identify it as a resource, event, agent, or location. Redraw the REA model to include the diamonds for each relationship and include an appropriate description in each diamond.

DQ 6-2 Examine the REA model for Salem Industrial Supply that appears in Figure 6.19. Determine the maximum cardinalities for each of the eight relationships indicated in the model. State any assumptions you needed to make, and be prepared to defend the rationale for your selection.

DQ 6-3 Examine the REA model for Salem Industrial Supply that appears in Figure 6.19. For each of the six entities in the model, list the attributes that a database designer should include in each table. Identify primary key attributes with (PK) and composite primary key attributes with (CPK). State any assumptions you needed to make, and be prepared to defend the rationale for the attributes you have chosen.

DQ 6-4 Refer to Figure 6.10 (pg. 191). To implement a many-to-many (M:N) relationship between two relations, the figure demonstrates creating a new relation with a composite key made up of the primary keys (Client_No and Employee_Number) of the relations to be linked. As an alternative, you might be tempted to simply add the fields Employee_Number to the Client relation, and Client_No to the Employee relation. Discuss the reporting problems that might occur with that alternative strategy.

DQ 6-5 Although today's enterprise systems incorporate many of the REA concepts, many organizations continue to use legacy systems. Why do you believe this is true? (Although the obvious answer is in the chapter, you may want to look to other sources to support to your answer.)

DQ 6-6 Although SQL is the *de facto* standard database language, there are many variations of the language. Using the Internet (or other sources), answer the following questions. What is a *de facto* standard? Provide examples (other than SQL) of such standards. How does SQL (the *primary* database language) being a *de facto* standard affect you, in your role as an information user?

DQ 6-7 (This question requires thinking across the chapters.) SQL's CREATE VIEW statement effectively brings together normalized data into a view that looks like a table. Examine Figure 6.17 (pg. 200). If the view BILLING_HOURS were a table, what normal form would it represent (unnormalized, 1NF, 2NF, or 3NF)?

Problems

P 6-1 What SQL command(s) would you use to generate a report for human resources from the relational database represented in Figure 6.11 (pg. 192)? Assuming human resources will be interested in tracking work, vacation, and sick time for all employees, be sure to consider all of these factors in designing your report format.

P 6-2 What SQL command(s) would you use to extract data from the report view in Problem 6-1 for Janet Robins?

Note: These problems should be completed with a database software package such as Access. For Problems 6-3 through 6-5, you may use data that you (or your instructor) have downloaded from an accounting database. Problem 6-6 demonstrates the interaction between an application software system and the database.

P 6-3 Using the information from Figures 6.11 (pg. 192), 6.12 (pg. 193), and 6.13 (pg. 195), create the database in the software package of your choice. This will require three steps:

a. Implement the relations from Figure 6.11.

b. Insert the data from Figure 6.11 into the relational tables.

c. Print the data from each of the relations to test for successful implementation.

P 6-4 *Note:* This problem is a continuation of Problem 6-3.

a. Create the standard report view for displaying the billing hours as suggested in Figure 6.17 (pg. 200).

b. Create the standard report view for displaying the human resources information as recommended in Problem 6-1.

c. Extract from the billing hours view the billing information for Hasbro, Inc., and print the results.

d. Extract from the human resources view the information related to Janet Robins, and print the results.

P 6-5 *Note:* This problem is a continuation of Problem 6-3 but requires access to the Internet, a site for posting the database on the Web, and an understanding of Internet access.

a. Take the database developed in Problem 6-3 and place it on the Internet (or your instructor may provide the same).

b. Access the database from your local computer using the appropriate address for the database. (Your software must be capable of using Internet addresses for locating the data.)

c. Use a series of queries similar to that in Figure 6.15 (pg. 198) to pull down the billing information for Fleet Services.

P 6-6 *Note:* This problem is a continuation of Problem 6-3 but requires use of a spreadsheet package that is capable of reading data from your database package (for example, Excel can import data from an Access database).

a. Using your spreadsheet package, construct SQL queries (or use the software's menu generation for queries) to import the billing information for Fleet Services.

b. Develop a report format for using the information from your queries to generate a nice-looking report in your spreadsheet package.

c. Document the queries used in your spreadsheet package to access the data and explain what each step of the queries does.

P 6-7 Using the REA model in Figure 6.19 (pg. 202) and your answers to Discussion Questions 6-1, 6-2, and 6-3, create a database for SIS in the software package of your choice. This will require that you do the following:

a. Create tables for each of the relations you identified in Discussion Question 6-1 that include the attributes you identified in Review Question 6-3.
b. Insert a few tuples of sample data that you devise.
c. Print the data from each of the relations to test for successful implementation.