CHAPTER

18

Implementing an REA Model in a Relational Database

LEARNING OBJECTIVES

After studying this chapter, you should be able to:

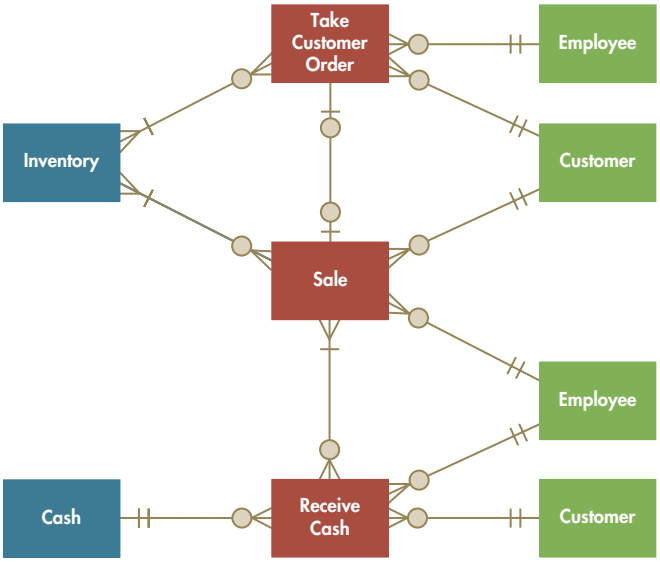
1. Integrate separate REA diagrams for individual business cycles into a single, comprehensive, organization-wide REA diagram.
2. Build a set of tables to implement an REA model of an AIS in a relational database.
3. Use the REA data model to write queries to retrieve information from an AIS relational database.

INTEGRATIVE CASE Fred's Train Shop

Paul Stone shows Fred the set of REA diagrams that he has developed to model the business activities for the revenue, expenditure, and payroll cycles of Fred's Train Shop. Fred verifies that Paul has correctly represented his company's business processes. He then says that although the diagrams "look nice," he wondered why Paul has spent so much time developing them, instead of building Fred the database he had promised. Paul responds that the time spent up front in thoroughly understanding Fred's Train Shop's business processes is necessary to properly design a database that will satisfy Fred's needs.

Paul asks Fred whether he has a database program. Fred replies that a relational database was part of the "business applications" he had purchased as part of an office productivity package. Fred says that although he knows how to use the program, he has not been able to figure out how to import data to it from his AIS and store it in a manner that will allow him to analyze his store's business activities. Paul says that he will create a useful database for Fred by following these steps:

1. First, he will integrate the separate REA diagrams he has developed into a single, comprehensive, enterprise-wide data model.
2. Second, he will use the integrated data model to design a set of relational database tables.
3. Third, he will show Fred how he can query the resulting database to generate both traditional financial statements as well as any custom performance reports.



Introduction

The previous chapter introduced the topic of REA data modeling and explained how to develop REA diagrams for an individual business cycle. This chapter shows how to implement an REA diagram in a database. We focus on relational databases because they are commonly used to support transaction processing systems and are likely to be familiar to most business students. Nevertheless, REA data modeling is not limited for use only in designing relational databases; it can also be used to design object-oriented databases.

We begin by showing how to integrate separate REA diagrams developed for individual business cycles into a single, comprehensive, enterprise-wide data model. Next, we explain how to implement the resulting model in a relational database. We then describe how to use the REA diagram to query the database to produce traditional financial statements as well as a variety of management reports.

Integrating REA Diagrams Across Cycles

Figures 18-1, 18-2, and 18-3 present REA diagrams of Fred's Train Shop's revenue, expenditure, and payroll cycles, respectively. These separate diagrams should be integrated to provide a single, comprehensive, enterprise-wide model of the organization. Doing so requires understanding what the cardinalities in each separate diagram reveal about the organization's business policies and activities. Figures 18-1 and 18-2 were explained in Chapter 17 (see discussion of Figures 17-6 and the comprehensive end-of-chapter problem, respectively), so we focus here on Figure 18-3.

Figure 18-3 depicts the payroll portion of Fred's Train Shop's HR/payroll cycle activities. The basic economic exchange involves acquiring the use of each employee's time and skills in exchange for which the employee receives a paycheck. Like many small businesses, Fred's Train Shop uses an electronic time clock to record the hours worked by each employee each day. Thus, each Time Worked event records the time an employee began and ended working on a specific day. Each such event must be linked to a particular employee and his or her supervisor; each employee or supervisor, however, may be linked to many different events. Similarly, a paycheck is issued to a particular employee and signed by a particular cashier, but each employee and cashier may be associated with many different Disburse Cash events over time. Hence, Figure 18-3 depicts the relationships between agents and events as being 1:N. The minimum cardinality on the agent side of those relationships is always 1, because each event *must* be linked to a specific employee. (For example, Fred would not want to issue a paycheck and leave the payee name blank.) The minimum cardinality on the event side of the relationships is always 0 in order to accommodate storing data about new employees prior to their beginning work and because the event entities are empty at the beginning of each new fiscal year.

FIGURE 18-1
Fred's Train Shop
Revenue Cycle

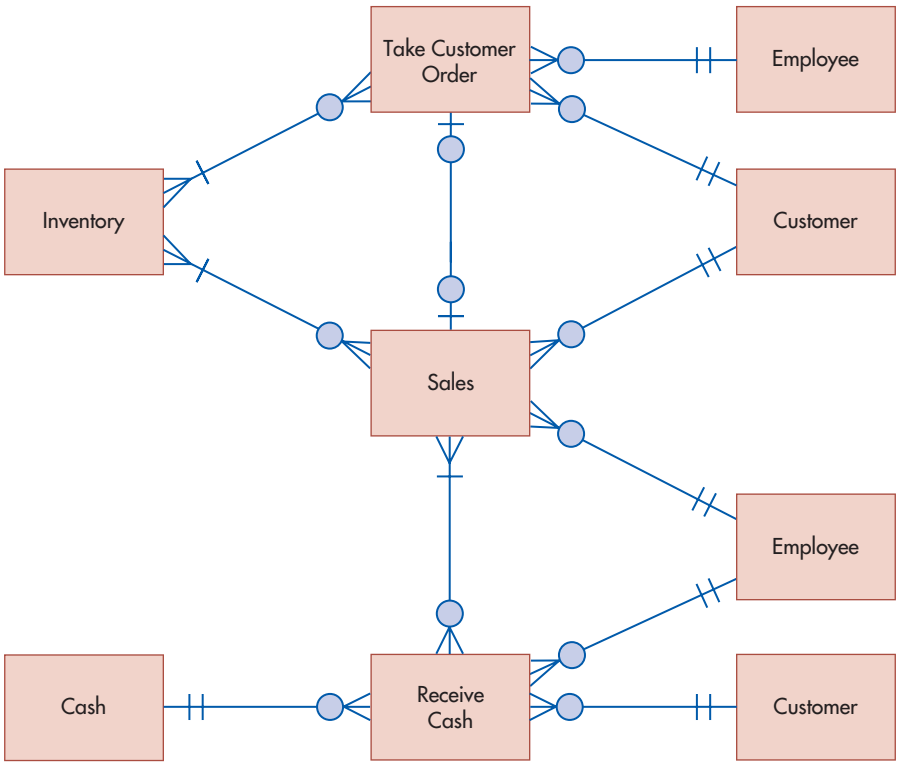
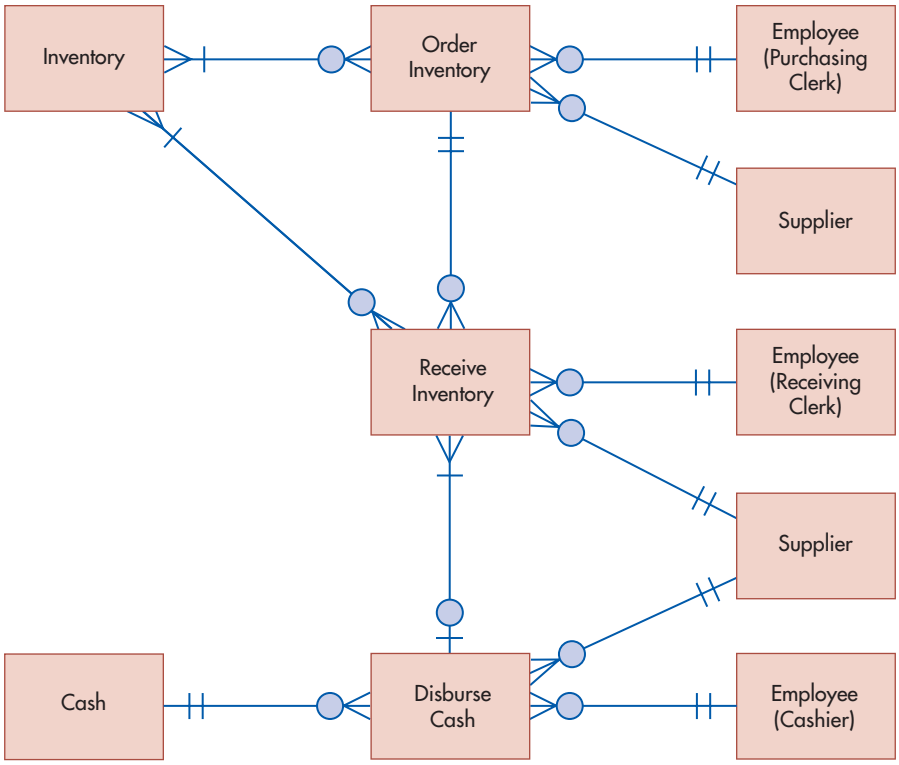


FIGURE 18-2
Fred's Train Shop
Expenditure Cycle



The relationship between the Time Worked and Disburse Cash events reflects the basic economic exchange of getting the use of an employee's time and paying for it. Figure 18-3 shows that the relationship between these two events is 1:N. This is because Fred's Train Shop, like most businesses, pays employees periodically but records their time worked daily. Thus, each Disburse Cash event is linked to many daily Time Worked events. Like most businesses,

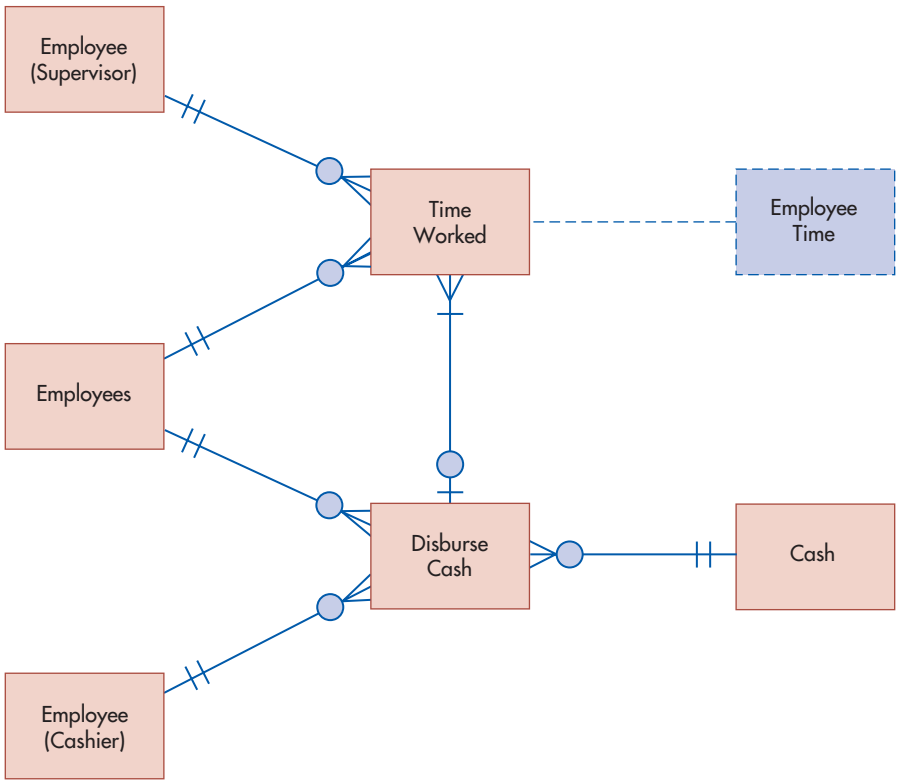


FIGURE 18-3
Fred's Train Shop Payroll Cycle

however, Fred's Train Shop does not divide one day into two different pay periods, nor does it pay employees in installments; thus, each Time Worked event is linked to only 1 Disburse Cash event. The minimum cardinalities on each side of the relationship reflect the normal business practice of paying employees after they have worked, rather than in advance.

The Employee Time entity requires some explanation. It represents the fact that the resource being acquired by the Time Worked event is the use of an employee's skills and knowledge for a particular period of time. Time, however, is different from inventory, cash, and other tangible resources, as well as from intangible resources like trade secrets or other forms of intellectual property, in that it cannot be stored. In addition, there are only a few relevant attributes about employee time: the hours worked and how that time was used. Every organization needs to monitor how much time each employee works in order to calculate payroll. The Time Worked event, which is an example of a "Get" resource event, serves this purpose. Chapter 19 will discuss how some organizations, such as manufacturers and professional services firms (e.g., law firms, consulting organizations, and accounting firms) also collect detailed records of how employees use their time, which is an example of a "Give" resource event, in order to properly bill clients for those services. These two events (Time Worked and Time Used) capture all of the information about employee time that is practical to collect and monitor. Consequently, the Employee Time resource entity is almost never implemented in an actual database. Therefore, it is depicted with dotted lines in Figure 18-3.

Finally, the cardinalities of the relationship between the Disburse Cash event and the Cash resource are identical to those in the expenditure cycle (Figure 18-2): Each check or electronic funds transfer must be linked to at least one cash account and can be linked to only one cash account, whereas the same cash account may be linked to many Disburse Cash events.

Now that we understand the business policies underlying Figures 18-1, 18-2, and 18-3, we can proceed to merge them into one integrated REA diagram. You have probably noticed that Figures 18-1, 18-2, and 18-3 each contain some of the same entities. For example, the Inventory resource appears in both Figures 18-1 and 18-2. The Disburse Cash event appears in both Figures 18-2 and 18-3. Both the Employee agent and the Cash resource appear in all three diagrams. Such redundancies provide the basis for combining REA diagrams depicting individual business cycles into a single, comprehensive, enterprise-wide REA model.

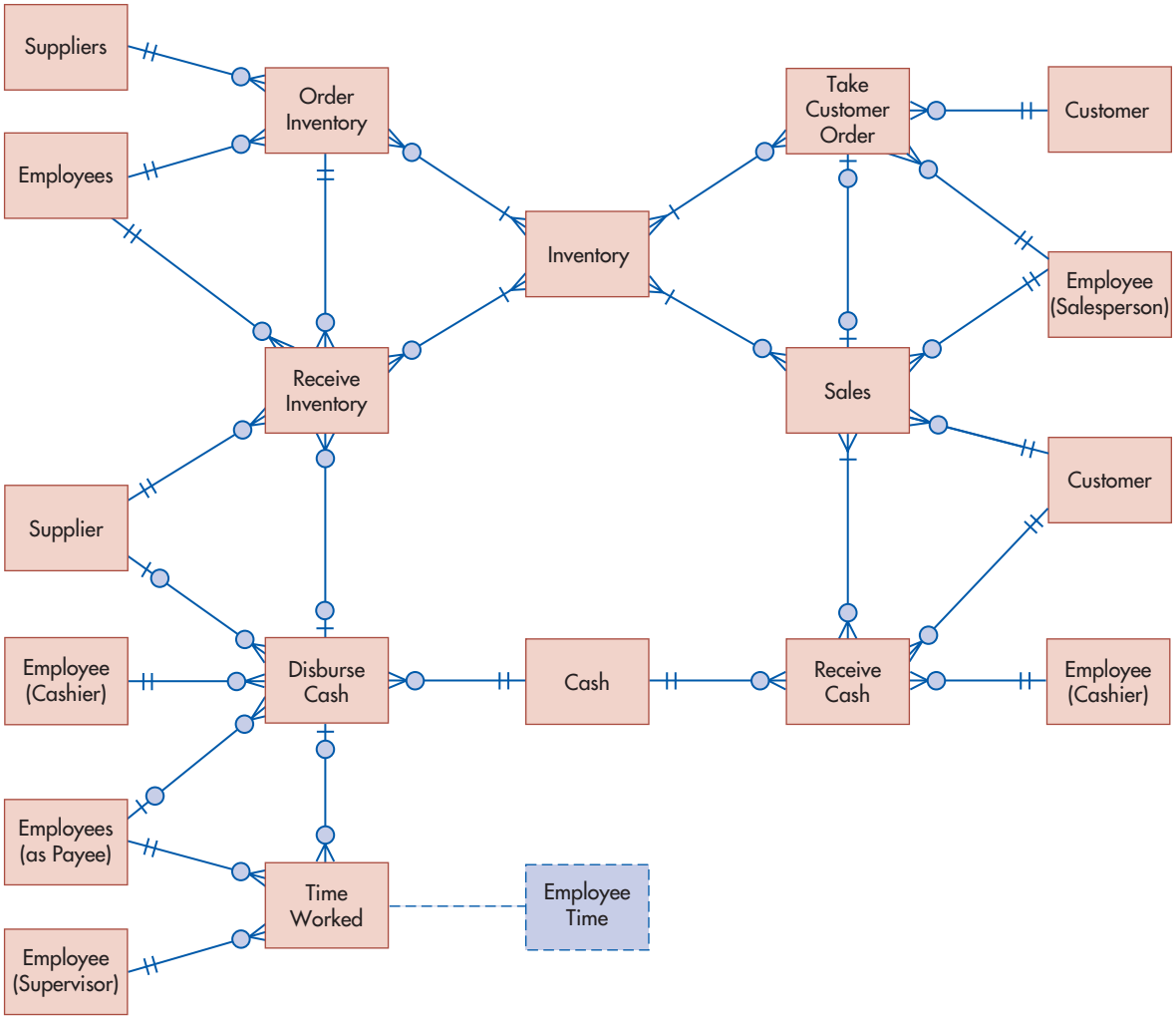


FIGURE 18-4
Integrated REA Diagram for Fred's Train Shop

Figure 18-4 shows such a model for Fred's Train Shop. Notice that the integrated diagram merges multiple copies of resource and event entities but retains multiple copies of agent entities. This maximizes the legibility of the comprehensive REA diagram by avoiding the need to have relationship lines cross one another. Let us now examine how to combine redundant resource and event entities.

MERGING REDUNDANT RESOURCE ENTITIES

Recall that REA diagrams for individual business cycles are built around basic give-to-get economic exchanges. Such economic duality relationships explain why a resource is either acquired or disposed of. They provide only a part of the story about each resource, however. For example, Figure 18-1 shows that inventory is reduced (the Sales event) in exchange for cash (the Receive Cash event). But Figure 18-1 does not show how that inventory was initially acquired. Nor does it show how the organization uses the cash it receives from customers. Conversely, Figure 18-2 shows how inventory was acquired (the Receive Inventory event) by giving up cash (the Disburse Cash event). Yet, Figure 18-2 does not show what the organization does with the inventory or how it acquired the cash used to pay suppliers.

Thus, REA diagrams for individual business cycles provide only partial information about the resources controlled by an organization. The complete picture would show how

each resource is acquired and how it is used. As shown in Figure 18-4, this can be done by redrawing an REA diagram to place common resources between the events that affect them. Doing so reflects another important duality that must be depicted in a complete REA model of any organization: Every resource must be connected to at least one event that increases that resource and to at least one event that decreases it.

MERGING REDUNDANT EVENT ENTITIES

REA diagrams for individual business cycles may include some events that also appear in the REA diagrams of another cycle. For example, Figures 18-2 and 18-3 both contain the Disburse Cash event entity. As was the case with resources, merging these multiple occurrences of the same event improves the legibility of the resulting comprehensive REA diagram. Thus, Figure 18-4 shows that the Disburse Cash event is linked to both the Receive Inventory and the Time Worked events.

Close examination of Figure 18-4 reveals an important difference, however, between merging redundant events and merging redundant resources: Merging redundant resources does not affect any cardinalities, but merging redundant events alters the minimum cardinalities associated with the other events that are related to the merged event. Thus, in Figure 18-4 the cardinalities between the Inventory resource and each of the four events to which it is related are the same as those depicted in Figures 18-1 and 18-2. In contrast, the cardinalities between the Disburse Cash event and the other events with which it is linked are different in Figure 18-4 than in Figures 18-2 and 18-3.

The reason for this difference lies in the underlying semantics about the nature of the relationship between the merged entity and other entities. An instance of a resource entity can be, and usually is, linked to multiple events. For example, a given inventory item carried by Fred's Train Shop can be linked not only to a Receive Inventory event, when it is acquired from a supplier, but also to a Sales event, when it is sold to a customer. In other words, the resource entity is linked to event entities in one business cycle *and* to event entities in the other cycle. Because both links are possible, none of the cardinalities in the individual REA diagrams needs to change.

The situation is different when merging an event across business cycles. The event that appears in both individual business cycle REA diagrams may be linked to *either* an event that is part of one business cycle *or* to an event that is part of another cycle *but cannot be linked to both* events. For example, in Figure 18-4, a particular Disburse Cash event (i.e., a particular check or EFT transaction) could be associated with a prior receipt of inventory from a supplier or with time worked by an employee, but the same check (or EFT transaction) cannot be used *both* to pay a supplier for receipt of inventory *and* to pay an employee for working the previous week. Consequently, the minimum cardinality associated with the other events *must* be 0 in the integrated REA diagram, regardless of what it was in each of the individual transaction cycle REA diagrams. To understand why, recall that a minimum of 1 means that each instance of that entity has to be associated with at least one instance of the other entity. In terms of cash disbursements in Figure 18-4, retaining the minimum 1 with the Time Worked event, for example, would mean that every Disburse Cash must be linked to a Time Worked event—which is clearly not true, because Fred may make a cash disbursement to pay a supplier. For similar reasons, the minimum cardinality from the Disburse Cash event to the Receive Inventory event must also be 0.

Merging two transaction cycles on a common event may also affect the minimum cardinalities between the merged event and the agents participating in that event. For example, in Figure 18-4 the minimum cardinality between the Disburse Cash event and the Supplier entity is now 0, instead of 1, as it was in Figure 18-2. The reason is that a given check (cash disbursement) may be written *either* to a supplier as payee or to an employee as payee, but the same check *cannot* be written to *both* agents simultaneously. That is why the minimum cardinality between the Disburse Cash event and the Employee (payee) agent is also 0. Thus, whenever a merged event involves different agents in each of the individual business cycles being merged, the minimum cardinalities between that event and those agents change from the usual 1 to 0, because the event may now be linked to either of the two types of agents, but not both.

VALIDATING THE ACCURACY OF INTEGRATED REA DIAGRAMS

Chapter 17 presented three basic principles for drawing REA diagrams for individual business cycles; the preceding discussion for combining such diagrams into a single, comprehensive, enterprise-wide model adds three more rules. Thus, a correctly drawn, integrated REA diagram must satisfy these six rules:

1. Every event must be linked to at least one resource.
2. Every event must be linked to two agents who participate in that event.
3. Every event that involves the disposition of a resource must be linked to an event that involves the acquisition of a resource. (This reflects the economic duality underlying "give-to-get" economic exchanges.)
4. Every resource must be linked to at least one event that increments that resource and to at least one event that decrements that resource.
5. If event A can be linked to more than one other event, but cannot be linked simultaneously to all of those other events, then the REA diagram should show that event A is linked to a minimum of 0 of each of those other events.
6. If an event can be linked to any one of a set of agents, but cannot be simultaneously linked to all those agents, then the REA diagram should show that event is linked to a minimum of 0 of each of those agents.

Notice that these six rules can be used not only to develop an integrated REA diagram but also as "check figures" to validate the accuracy of a completed REA diagram. Technically, Figure 18-4 is not complete because rule 4 is not satisfied for the Employee Time resource. We will correct this shortcoming in Chapter 19. For now, let us ignore it and proceed to the next step in the database design process: implementation of an REA data model in a relational database.

Implementing an REA Diagram in a Relational Database

Once an REA diagram has been developed, it can be used to design a well-structured relational database. In fact, creating a set of tables from an REA diagram automatically results in a well-structured relational database that is not subject to the update, insert, and delete anomaly problems discussed in Chapter 4.

There are three steps to implementing an REA diagram in a relational database:

1. Create a table for each *distinct* entity in the diagram and for each many-to-many relationship.
2. Assign attributes to appropriate tables.
3. Use foreign keys to implement one-to-one and one-to-many relationships.

Recall that even though REA diagrams for different organizations may include the same entities, differences in business policies are likely and will result in differences in relationship cardinalities. For example, the REA diagram for one organization may show a 1:1 relationship between the Sales and Receive Cash events, whereas the REA diagram for another organization may model that same pair of events as being involved in a M:N relationship. Thus, the design of a database (number of tables, placement of attributes) is specific to the organization being modeled.

STEP 1: CREATE TABLES FOR EACH DISTINCT ENTITY AND M:N RELATIONSHIP

A properly designed relational database has a table for each distinct entity and for each many-to-many relationship in an REA diagram. Figure 18-4 has 13 distinct entities, but as previously discussed, one, Employee Time, will not be implemented in the database. The

remaining 12 distinct entities that are depicted in Figure 18-4 need to be implemented as tables in a relational database. Seven tables will represent the event entities in the diagram: Order Inventory, Receive Inventory, Disburse Cash, Time Worked, Take Customer Order, Sales, and Receive Cash. There are two tables for resource entities: Inventory and Cash. Three tables are needed to implement the distinct agent entities: Employees, Customers, and Suppliers (supervisors are labeled separately to make the diagram easier to read, but are themselves employees).

Figure 18-4 also depicts five M:N relationships. Three are from the revenue cycle: Take Customer Orders–Inventory, Sales–Inventory, and Sales–Receive Cash. Two others are from the expenditure cycle: Inventory–Order Inventory and Inventory–Receive Inventory. Therefore, the 17 tables listed in Table 18-1 must be created to accurately implement Figure 18-4 in a relational database. Notice that the suggested table names in Table 18-1 correspond to the names of entities in the REA diagram or, in the case of tables for M:N relationships, are

TABLE 18-1 Table Names and Attribute Placement for Figure 18-4

		ATTRIBUTES	
TABLES	**PRIMARY KEY**	**FOREIGN KEYS**	**OTHER ATTRIBUTES**
Order Inventory	Purchase order number	Supplier number, employee number	Date, time, reason
Receive Inventory	Receiving report number	Supplier number, employee number, purchase order number, check number	Date, time, remarks, vendor invoice number
Disburse Cash	Check number	Supplier number, employee number (payee), employee number (signer), account number	Amount, description, date
Take Customer Order	Sales order number	Customer number, employee number	Date, time, special remarks
Sales	Invoice number	Customer number, employee number, sales order number	Date, time, invoice sent (Y/N)
Receive Cash	Remittance number	Customer number, employee number, account number	Date, time, method of payment
Time Worked	Timecard number	Employee number, supervisor number, paycheck number	Date, time in, time out
Inventory	Product number		Description, list price, standard cost, beginning quantity-on-hand, beginning quantity-available, reorder quantity, reorder point
Cash	Account number		Beginning-balance, type of account
Employees	Employee number		Name, date hired, date of birth, pay rate, job title
Customers	Customer number		Name, address,[a] beginning account balance, credit limit
Suppliers	Supplier number		Name, address,[a] beginning account balance, performance rating
Order Inventory—Inventory	Purchase order number, product number		Quantity ordered, actual unit cost
Receive Inventory—Inventory	Receiving report number, product number		Quantity received, condition
Take Customer Order—Inventory	Sales order number, product number		Quantity ordered
Sales—Inventory	Invoice number, product number		Quantity sold, actual sale price
Sales—Receive Cash	Invoice number, remittance number		Amount applied to invoice

[a]Actually, only the street address and zip code would be stored in these tables. In both tables zip code would be a foreign key. Zip code would also be the primary key of an "address table," which would also include city and state as other attributes.

hyphenated concatenations of the entities involved in the relationship. This makes it easier to verify that all necessary tables have been created and also makes it easier to use the REA diagram as a guide when querying the database.

STEP 2: ASSIGN ATTRIBUTES TO EACH TABLE

The next step is to determine which attributes should be included in each table.[1] The database designer needs to interview users and management to identify which facts need to be included in the database. The database designer must use the REA diagram to help determine in which table(s) to place those facts, depending upon whether that fact is a primary key or is just a descriptive attribute.

IDENTIFY PRIMARY KEYS As explained in Chapter 4, every table in a relational database must have a primary key, consisting of an attribute, or combination of attributes, that uniquely identifies each row in that table. Companies often create numeric identifiers for specific resources, events, and agents. These numeric identifiers are good candidates for primary keys. For example, Table 18-1 shows that Fred's Train Shop uses invoice number as the primary key of the sales table and customer number as the primary key of the customer table.

Usually the primary key of a table representing an entity is a single attribute. The primary key for M:N relationship tables, however, always consists of two attributes that represent the primary keys of each entity linked in that relationship. For example, the primary key of the Sales–Inventory table consists of both the invoice number (the primary key of the sales entity) and product number (the primary key of the inventory entity). Such multiple-attribute primary keys are called **concatenated keys**.

concatenated keys - Two or more primary keys of other database tables that, together, become the unique identifier or primary key of an M:N relationship table.

ASSIGN OTHER ATTRIBUTES TO APPROPRIATE TABLES Additional attributes besides the primary key are included in each table to satisfy transaction processing requirements and management's information needs. As discussed in Chapter 4, any other attribute included in a relational database table must either be a fact about the object represented by the primary key or a foreign key. Thus, information about customers, such as their name and address, belongs in the Customer table, because those attributes describe the object represented by the primary key (customer number) of the Customer table; those attributes do not belong in the Sales table, because they do not describe some property of the object represented by the primary key (invoice number) of that table.

Table 18-1 shows some of the attributes that Paul Stone has assigned to the various tables he has created to implement Figure 18-4 in a relational database. Some of these attributes, such as the date of each sale and the amount remitted by a customer, are necessary for complete and accurate transaction processing and the production of financial statements and managerial reports. Other attributes are stored because they facilitate the effective management of an organization's resources, events, and agents. For example, Fred can use data about the time of day when each sales transaction occurs to design staff work schedules.

Table 18-1 also includes other attributes in some of the M:N relationship tables. Let us examine the placement of these attributes in some of the M:N tables to see why they must be stored in those particular tables. Consider first the Sales–Receive Cash table. Recall that Fred's Train Shop allows its customers to make multiple purchases on credit and to make installment payments on their outstanding balances. Thus, one customer payment may need to be applied to several different invoices (sales transactions). Therefore, the attribute "amount applied" cannot be placed in the Receive Cash table because it could take on more than one value (one for each invoice being paid), thereby violating the basic requirement of relational databases that every attribute in every row be single-valued (i.e., the requirement that every table be a flat file). Nor can the attribute "amount applied" be placed in the Sales table, because the possibility of installment payments also creates a situation in which that attribute can have multiple values (i.e., one entry for each installment payment related to that particular sale). Thus, analysis of the underlying business process indicates

[1]As explained in Chapter 17, some designers prefer to include attributes as part of the REA diagram itself. We choose to list them in a separate table to reduce the clutter on the diagram.

that the attribute "amount applied" is a fact about both a specific customer payment (remittance) and a specific sales transaction and, therefore, belongs in the M:N table linking those two events.

Now examine the Sales–Inventory table. Each row in this table contains information about a line item in an invoice. Although many customers of Fred's Train Shop buy just one of each kind of product it sells, some sales to customers involve larger quantities. For example, a department store may buy five identical coal cars (product number 31125) for its window display. Consequently, Fred's Train Shop must record the quantity sold of each item. Each sales event, however, may include more than one inventory item. Thus, the attribute "quantity sold" may have several values on a single sales invoice, one for each different item (product number) sold. Consequently, the attribute "quantity sold" cannot be placed in the Sales table, because there can be more than one "quantity sold" value associated with a given invoice number. In addition, recall that Fred's Train Shop, like most retail stores, tracks inventory by kinds of items, each of which is identified by product number, not by specific identification. Therefore, a given item, such as an orange diesel locomotive, product number 14887, may be sold as part of many different sales transactions. Consequently, "quantity sold" cannot be an attribute in the Inventory table because it can take on multiple values. Instead, the preceding analysis makes it clear that the attribute "quantity sold" pertains to a specific product number on a specific sales invoice. Therefore, it belongs in the M:N relationship table that links the inventory and sales entities.

Price and Cost Data. In Table 18-1, notice that information about prices and costs are stored as an attribute in several different tables. For example, the Inventory table stores the suggested list price for the item, which generally remains constant for a given fiscal period. The Sales table stores the actual sales price, which varies during the course of the year as a result of sales promotions. Similarly, the standard and actual purchase costs of each item are stored in different tables. The general rule is that time-independent data should be stored as an attribute of a resource or agent, but data that varies across time should be stored with event entities or M:N relationships that link a resource and an event.

Cumulative and Calculable Data. Notice that Table 18-1 does not contain cumulative data, such as "quantity-on-hand" in the inventory table, or calculable data, such as "total amount of sale" in the sales table. The reason is that neither type of data item is needed because those values can be derived from other attributes that are stored. For example, the quantity-on-hand of a given inventory item equals the quantity-on-hand at the beginning of the current fiscal period (an attribute of the Inventory table) plus the total quantity purchased this period (which is itself calculated by summing the quantity received attribute in the Inventory–Receive Inventory table) minus the total quantity sold (which is calculated by summing the quantity sold attribute in the Sales–Inventory table) in this period. Similarly, the total amount of a sales transaction can be calculated by multiplying the quantity sold by the actual sale price of each item sold and summing that result for every row in the Sales–Inventory table that has the same invoice number.

STEP 3: USE FOREIGN KEYS TO IMPLEMENT 1:1 AND 1:N RELATIONSHIPS

Although 1:1 and 1:N relationships also can be implemented as separate tables, it is usually more efficient to implement them by means of foreign keys. Recall from Chapter 4 that a foreign key is an attribute of one entity that is itself the primary key of another entity. For example, the attribute "customer number" might appear in both the Customer and the Sales tables. It would be the primary key of the Customer table, but a foreign key in the Sales table.

USING FOREIGN KEYS TO IMPLEMENT 1:1 RELATIONSHIPS In a relational database, 1:1 relationships between entities can be implemented by including the primary key of either entity as a foreign key in the table representing the other entity. For purposes of designing a well-structured database, the choice of which table to place the foreign key in is arbitrary. Careful analysis of the minimum cardinalities of the relationship, however, may suggest which approach is likely to be more *efficient*.

Consider the case of a 1:1 relationship between sales and customer payments (see Figure 17-7, panel A). The minimum cardinality for the Receive Cash event is 0, indicating the existence of credit sales, and the minimum cardinality for the Sale event is 1, indicating that customer payments only occur after a sale has been made (e.g., there are no advance deposits). In this case, including invoice number (the primary key of the sales event) as a foreign key in the Receive Cash event may be more efficient because then only that one table would have to be accessed and updated when processing each customer payment. Moreover, for 1:1 relationships between two sequential events, including the primary key of the event that occurs first as a foreign key in the event that occurs second may improve internal control, because the employee given access to update the table containing data about the second event need not access the table used to store information about the first event. However, the internal control benefits of doing this must be weighed against a possible increase in difficulty of querying the database.

USING FOREIGN KEYS TO IMPLEMENT 1:N RELATIONSHIPS As with 1:1 relationships, 1:N relationships also should be implemented in relational databases with foreign keys. There is only one way to do this: The primary key of the entity that can be linked to multiple instances of the other entity *must* become a foreign key in that other entity. Thus, in Table 18-1, the primary keys of the Salesperson and Customer tables are included as foreign keys in the Sales table. Similarly, the primary keys of the Cash, Customer, and Cashier tables are included as foreign keys in the Receive Cash table. Reversing this procedure would violate one of the fundamental rules of relational database design. For example, placing invoice number as a foreign key in the Customer table would not work because it can have more than one value (i.e., a given customer may be, and one hopes is, associated with multiple invoice numbers because of participation in many sales transactions).

Note that this is why M:N relationships *must* be implemented as separate tables: Since each entity can be linked to multiple occurrences of the entity on the other side of the relationship, it is not possible to make either entity's primary key a foreign key in the other entity. Consider the M:N relationship between the Sales event and the Inventory resource. Each Sales event may be linked to many different inventory items. Therefore, product number cannot be used as a foreign key in the Sales table because it would be multivalued. Conversely, each product may be involved in many different sales transactions. Therefore, invoice number cannot be used as a foreign key in the Inventory table because it would be multivalued. Thus, the only way to link the Sales and Inventory tables is to create a new table that has separate rows for each actual combination of invoice number and product number. Notice that in the resulting M:N table a particular invoice number (e.g., 787923) will appear in many different rows, one for each different item that was part of that sales transaction. Conversely, a particular product number (e.g., 12345) will appear in many different rows, once for each different sales transaction in which it was sold. Thus, neither attribute, by itself, uniquely identifies a given row. However, there will be only one row that contains the combination of a particular invoice number and product number (e.g., invoice number 787923 and product number 12345); thus, both attributes together can serve as a primary key for the M:N table.

COMPLETENESS CHECK

The list of attributes that users and management want included in the database provides a means to check and validate the implementation process. Every attribute in that list should appear in at least one table, as either a primary key or "other" attribute.

Checking this list against the table column names may reveal not only the fact that a particular attribute has not been assigned to the appropriate table in the database but may even indicate the need to modify the REA diagram itself. For example, when Paul Stone double-checked the list of desired attributes, he found that he did not have any table in which to place the attribute "product discussed during sales call." In such a situation, the database designer needs to revisit users and management to understand the purpose for collecting that particular attribute. In this case, Fred explains that he plans to have one of his employees call on corporate customers to set up sample displays. Fred wants to collect information about such demonstrations to evaluate their effectiveness.

Paul realizes that this necessitates creating another event entity, Call on Customers, which would be linked to both the Customer and Employee agent tables, the Inventory table, and the Take Customer Order table (see Figure 19-1 on p. 586). The primary key of this new event would be "appointment number." Employee number and customer number would be foreign keys in the table, which would also include attributes for the date and time of the demonstration, along with a text field for comments. Because each demonstration could involve multiple items and each item could be demonstrated in many different calls, there would be a M:N relationship between the Call on Customer event and the Inventory table. The set of rows in that table with the same appointment number would identify which products were shown during a particular sales call. Some calls would result in orders, but others would not. In addition, some orders would occur without any sales call (e.g., because the customer saw an advertisement). Therefore, the minimum cardinality is 0 on each side of the relationship between the Call on Customer and Take Customer Order events. The maximum cardinality on each side of the relationship is 1 to simplify tracking the effect of sales calls.

Paul's need to modify his REA diagram to accommodate additional facts is not unusual. Indeed, it is often useful to create tables and assign attributes to them even before completely finishing an REA diagram. This helps clarify exactly what each entity represents, which often resolves dilemmas about the correct cardinalities for various relationships. The database designer can then modify and refine the REA diagram to include additional entities and relationships to accommodate other facts that are supposed to be in the database but that have not yet been assigned to existing tables.

Once all attributes have been assigned to tables, the basic requirements for designing well-structured relational databases that were discussed in Chapter 4 can be used as a final accuracy check:

1. Every table must have a primary key.
2. Other nonkey attributes in each table must be either a fact about the thing designated by the primary key or foreign keys used to link that table to another table.
3. Every attribute in every table is single-valued (i.e., each table is a flat file).

Note how the set of relational tables listed in Table 18-1 satisfy these three basic requirements. Moreover, they also correspond to Figure 18-4 and, therefore, reflect Fred's Train Shop's business policies. This correspondence also facilitates using the REA diagram to guide the design of queries and reports to retrieve and display information about the organization's business activities.

Using REA Diagrams to Retrieve Information from a Database

Thus far, we have shown how to use the REA data model to guide the design of an AIS that will efficiently store information about an organization's business activities. In this section we refer to Figure 18-4 and Table 18-1 to show how to use completed REA diagrams to facilitate the retrieval of that information to evaluate performance.

CREATING JOURNALS AND LEDGERS

At first glance, it may appear that a number of elements found in traditional AIS, such as journals, ledgers, and information about receivables and payables, are missing. We will see that such information, although not explicitly represented as entities in an REA diagram, can be obtained through appropriate queries. These queries need only be created once and can then be stored and rerun whenever desired.

DERIVING JOURNALS FROM QUERIES Journals provide a chronological listing of transactions. In a relational database designed according to the REA data model, event entities store information about transactions. Thus, the information normally found in a journal is contained

in the tables used to record data about events. For example, the Sales and Sales–Inventory tables contain information about a particular sales transaction. Thus, a sales journal can be produced by writing a query that references both tables to calculate the amount of sales made during a given period.

Doing so, however, would not necessarily create the traditional sales journal because it would produce a list of *all* sales transactions, including both credit and cash sales. Traditionally, however, sales journals record only *credit* sales. In a relational database built on the REA model, such as the one in Figure 18-4, customer payments are recorded in the Receive Cash event table. Consequently, a query to produce a traditional sales journal (i.e., listing only those sales made on credit) would have to also include both the Receive Cash and the Sales–Receive Cash tables. A database built on the REA model would create a row in the Sales table for each sale of merchandise to a customer and a row in the Receive Cash table to record receipt of payment from a customer. For cash sales, both rows would have the same values in the date and customer number columns. Therefore, the logic of a query to produce a traditional sales journal would restrict the output to display only those sales that are *not* linked to a corresponding customer payment event (i.e., the same customer number appears in both tables, and the amount of the Receive Cash event equals the amount of the sale) that occurred *on the same day* as the Sales event. (Rows in the Receive Cash table with dates later than the date of the corresponding sales transaction represent payments on credit sales.) Similar processes can be followed to write queries to produce other special journals, such as all credit purchases or all cash disbursements not related to payroll.

LEDGERS Ledgers are master files that contain cumulative information about specific accounts. In a relational database designed according to the REA data model, resource entities contain permanent information that is carried over from one fiscal year to the next. Thus, much of the information about an organization's assets that is traditionally recorded in ledgers is stored in resource tables in an REA-based relational database. For example, each row in the Equipment resource table would contain information about a specific piece or class of machinery, such as its acquisition cost, useful life, depreciation method, and estimated salvage value. Similarly, each row in the Cash resource table contains information about a specific account that holds the organization's cash and cash equivalents, and each row in the Inventory resource table stores information about a specific inventory item.

Each of these resource accounts is affected by increment and decrement events: Equipment is purchased and used; cash is received and disbursed; inventory is purchased and sold. Thus, queries to display the current cumulative balance for these accounts must reference not only the appropriate table for that resource entity but also the event tables that affect it. For example, a query to display the current balance in a specific bank account would reference not only the Cash resource table, to identify the account number and the balance as of the beginning of the current fiscal period, but also the Receive Cash and Disburse Cash tables, to find the inflows and outflows affecting that account during the current fiscal period.

GENERATING FINANCIAL STATEMENTS

A completed REA diagram can also be used to guide the writing of queries to produce the information that would be included in financial statements. Many financial statement accounts, such as Cash, Inventory, and Fixed Assets, are represented as resources in the REA model. An important exception, however, is claims: Figure 18-4 includes neither an entity called Accounts Receivable nor one called Accounts Payable. As explained in Chapter 17, the reason is that both of these accounts merely represent an imbalance between two related events. Accounts receivable represents sales transactions for which customer payments have not yet been received, and accounts payable represents purchases from suppliers that have not yet been paid for. Therefore, neither accounts receivable nor accounts payable needs to be explicitly stored as a separate table in an REA database. Instead, those claims can be derived from a set of queries against the relevant agent and event tables. For example, three queries can be

used to calculate total Accounts Receivable.[2] First, sum the total beginning balances in every customer account. Second, calculate total new sales this period by writing a query against the M:N Sales–Inventory relationship table to sum the product of quantity sold times unit price. Third, determine the total cash received from customers this period by summing the amount column in the Receive Cash table. Total Accounts Receivable equals beginning Accounts Receivable (query 1) plus new sales (query 2) minus cash receipts (query 3). A similar set of queries will produce total Accounts Payable.

CREATING MANAGERIAL REPORTS

The REA data model facilitates creating a wide variety of managerial reports because it integrates nonfinancial and financial data. For example, Table 18-1 shows that the Sales entity in Figure 18-4 includes an attribute to record the *time* that the sale occurred. Fred can use this data to track sales activity during different times of the day to better plan staffing needs at the train shop. Other useful nonfinancial attributes could be included in other tables. For example, the Customer table could be modified to include an attribute that identifies whether a customer has an indoor or outdoor train layout. If Fred can collect this information from his customers, he may be able to better target his advertising to meet each individual customer's interests. In addition, Table 18-1 can be modified easily to integrate data from external sources. For example, to better evaluate the creditworthiness of business customers, Fred may decide to collect information from a credit rating agency, such as Dun & Bradstreet. This information could be added to the database by creating an additional column in the Customer table to store the customer's credit rating. A similar process could be used to store information about suppliers that could be used in the vendor selection process.

Summary and Case Conclusion

REA diagrams for individual business cycles depict basic give-to-get economic duality relationships but usually provide only a partial view of resources, showing either how they are acquired or how they are used, but not both. Therefore, individual business cycle REA diagrams need to be combined in order to provide a comprehensive enterprise-wide data model. This is usually done by merging resource and event entities that appear in two or more individual REA diagrams. Merging two or more REA diagrams that contain the same resource entity does not require any changes to the cardinality pairs in the individual diagrams. Merging two or more diagrams that contain a common event entity, however, often requires changing the minimum cardinalities associated with the other events to 0 to reflect the fact that the merged event may be connected to any one of several different events but not to all of them simultaneously. The minimum cardinalities associated with the agents participating in those merged events may also have to be changed to 0.

A data model documented in an REA diagram can be implemented in a relational database in three steps. First, create tables for all unique entities and M:N relationships in the REA diagram. Second, assign primary keys and nonkey attributes to each table. Third, use foreign keys to implement 1:1 and 1:N relationships.

Paul Stone follows these steps to implement a database AIS for Fred's Train Shop based on Figure 18-4. He first creates separate tables for each of the 12 distinct entities and 5 M:N relationships in the figure. Next, Paul identifies primary keys for each table and uses some of them as foreign keys to implement the 1:1 and 1:N relationships in Figure 18-4. He then assigns the remaining attributes that Fred wants to monitor to the appropriate tables. Paul then demonstrates how easy it is to write queries to retrieve a variety of managerial reports and financial statements from the relational DBMS. Fred is quite impressed and immediately begins to use the new system to provide detailed information about the business activities of Fred's Train Shop.

[2]To calculate the account balance for an individual customer, follow a similar process but restrict the queries to just those Sales and Receive Cash events that have a specific customer number as a foreign key.