6 ADVANCED DATA MODELING

SIX

In this chapter, you will learn:

- About the extended entity relationship (EER) model
- How entity clusters are used to represent multiple entities and relationships
- The characteristics of good primary keys and how to select them
- How to use flexible solutions for special data modeling cases
- What issues to check for when developing data models based on EER diagrams

Preview

In the previous three chapters, you learned how to use entity relationship diagrams (ERDs) and normalization techniques to properly create a data model. In this chapter, you learn about the extended entity relationship (EER) model. The EER model builds on ER concepts and adds support for entity supertypes, subtypes, and entity clustering.

Most current database implementations are based on relational databases. Because the relational model uses keys to create associations among tables, it is essential to learn the characteristics of good primary keys and how to select them. Selecting a good primary key is too important to be left to chance, so in this chapter we cover the critical aspects of primary key identification and placement.

Focusing on practical database design, this chapter also illustrates some special design cases that highlight the importance of flexible designs, which can be adapted to meet the demands of changing data and information requirements. Data modeling is a vital step in the development of databases that in turn provide a good foundation for successful application development. Remember that good database applications cannot be based on bad database designs, and no amount of outstanding coding can overcome the limitations of poor database design.

To help you carry out data modeling tasks, the chapter concludes with a checklist that outlines basic database modeling principles.

6.1 THE EXTENDED ENTITY RELATIONSHIP MODEL

As the complexity of the data structures being modeled has increased and as application software requirements have become more stringent, there has been an increasing need to capture more information in the data model. The **extended entity relationship model** (**EERM**), sometimes referred to as the enhanced entity relationship model, is the result of adding more semantic constructs to the original entity relationship (ER) model. As you might expect, a diagram using this model is called an **EER diagram** (**EERD**). In the following sections, you will learn about the main EER model constructs—entity supertypes, entity subtypes, and entity clustering—and see how they are represented in ERDs.

6.1.1 Entity Supertypes and Subtypes

Because most employees possess a wide range of skills and special qualifications, data modelers must find a variety of ways to group employees based on employee characteristics. For instance, a retail company could group employees as salaried and hourly employees, while a university could group employees as faculty, staff, and administrators.

The grouping of employees to create various *types* of employees provides two important benefits:

- It avoids unnecessary nulls in the employee attributes when some employees have characteristics that are not shared by other employees.
- It enables a particular employee type to participate in relationships that are unique to that employee type.

To illustrate those benefits, let's explore the case of an aviation business. The aviation business employs pilots, mechanics, secretaries, accountants, database managers, and many other types of employees. Figure 6.1 illustrates how pilots share certain characteristics with other employees, such as a last name (EMP_LNAME) and hire date (EMP_HIRE_DATE). On the other hand, many pilot characteristics are not shared by other employees. For example, unlike other employees, pilots must meet special requirements such as flight hour restrictions, flight checks, and periodic training. Therefore, if all employee characteristics and special qualifications were stored in a single EMPLOYEE entity, you would have a lot of nulls or you would have to make a lot of needless dummy entries. In this case, special pilot characteristics such as EMP_LICENSE, EMP_RATINGS, and EMP_MED_TYPE will generate nulls for employees who are not pilots. In addition, pilots participate in some relationships that are unique to their qualifications. For example, not all employees can fly airplanes; only employees who are pilots can participate in the "employee flies airplane" relationship.

FIGURE 6.1 **Nulls created by unique attributes**

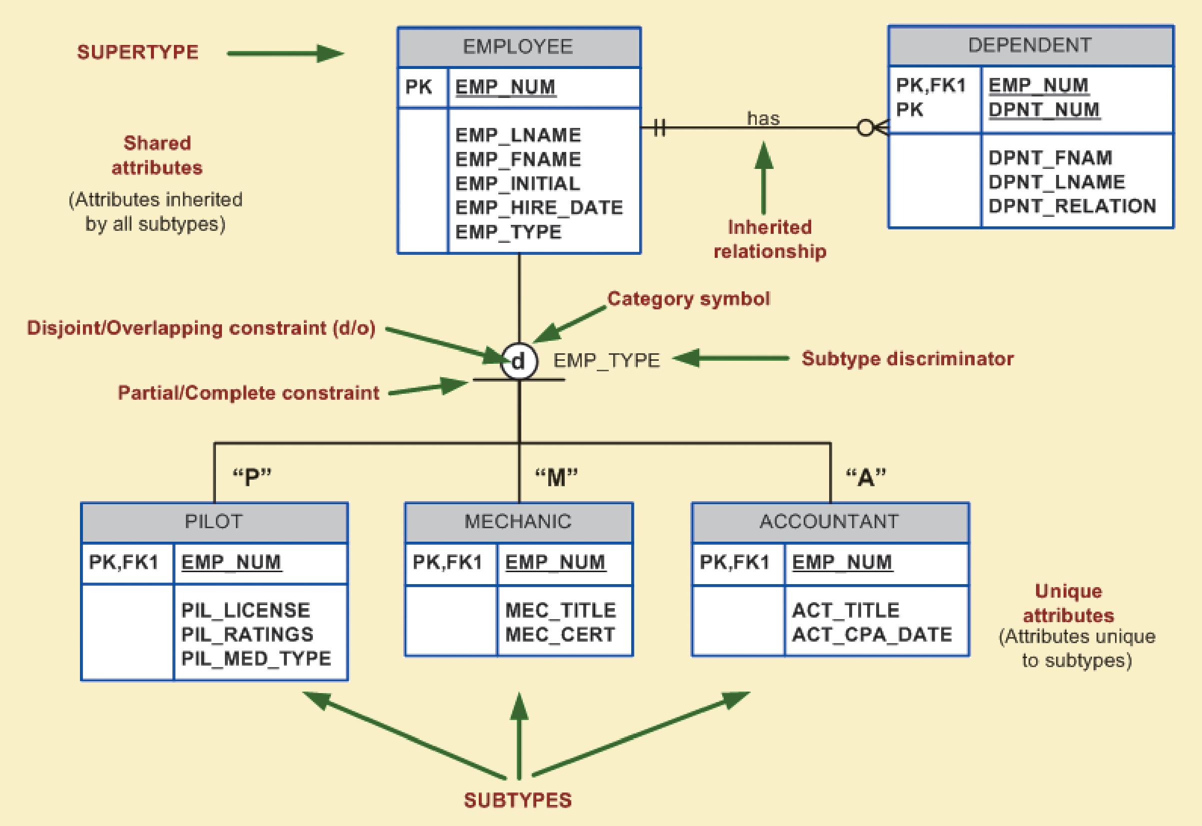
EMP_NUM	EMP_LNAME	EMP_FNAME	EMP_INITIAL	EMP_LICENSE	EMP_RATINGS	EMP_MED_TYPE	EMP_HIRE_DATE
100	Kolmycz	Xavier	T				15-Mar-88
101	Lewis	Marcos		ATP	SEL/MEL/Instr/CFII	1	25-Apr-89
102	Vandam	Jean					20-Dec-93
103	Jones	Victoria	R				28-Aug-03
104	Lange	Edith		ATP	SEL/MEL/Instr	1	20-Oct-97
105	Williams	Gabriel	U	COM	SEL/MEL/Instr/CFI	2	08-Nov-97
106	Duzak	Mario		COM	SEL/MEL/Instr	2	05-Jan-04
107	Diante	Venite	L				02-Jul-97
108	Wiesenbach	Joni					18-Nov-95
109	Travis	Brett	T	COM	SEL/MEL/SES/Instr/CFII	1	14-Apr-01
110	Genkazi	Stan					01-Dec-03

Based on the preceding discussion, you would correctly deduce that the PILOT entity stores only those attributes that are unique to pilots, and that the EMPLOYEE entity stores attributes that are common to all employees. Based on that hierarchy, you can conclude that PILOT is a *subtype* of EMPLOYEE, and that EMPLOYEE is the *supertype* of PILOT. In modeling terms, an **entity supertype** is a generic entity type that is related to one or more **entity subtypes**, where the entity supertype contains the common characteristics, and the entity subtypes contain the unique characteristics of each entity subtype. In the next section, you will learn how the entity supertypes and subtypes are related in a specialization hierarchy.

6.1.2 Specialization Hierarchy

Entity supertypes and subtypes are organized in a **specialization hierarchy**, which depicts the arrangement of higher-level entity supertypes (parent entities) and lower-level entity subtypes (child entities). Figure 6.2 shows the specialization hierarchy formed by an EMPLOYEE supertype and three entity subtypes—PILOT, MECHANIC, and ACCOUNTANT. The specialization hierarchy reflects the 1:1 relationship between EMPLOYEE and its subtypes. For example, a PILOT subtype occurrence is related to one instance of the EMPLOYEE supertype, and a MECHANIC subtype occurrence is related to one instance of the EMPLOYEE supertype. The terminology and symbols in Figure 6.2 are explained throughout this chapter.

FIGURE 6.2 A specialization hierarchy



The relationships depicted within the specialization hierarchy are sometimes described in terms of "is-a" relationships. For example, a pilot *is an* employee, a mechanic *is an* employee, and an accountant *is an* employee. It is important to understand that within a specialization hierarchy, a subtype can exist only within the context of a supertype, and every subtype can have only one supertype to which it is directly related. However, a specialization hierarchy can have many levels of supertype/subtype relationships—that is, you can have a specialization hierarchy in which a supertype has many subtypes; in turn, one of the subtypes is the supertype to other lower-level subtypes.

ONLINE CONTENT

This chapter covers only specialization hierarchies. The EER model also supports specialization *lattices,* where a subtype can have multiple parents (supertypes). However, those concepts are better covered under the object-oriented model in **Appendix G, Object-Oriented Databases.** The appendix is available in the Student Online Companion for this book.

As you can see in Figure 6.2, the arrangement of entity supertypes and subtypes in a specialization hierarchy is more than a cosmetic convenience. Specialization hierarchies enable the data model to capture additional semantic content (meaning) into the ERD. A specialization hierarchy provides the means to:

- Support attribute inheritance.
- Define a special supertype attribute known as the subtype discriminator.
- Define disjoint/overlapping constraints and complete/partial constraints.

The following sections cover such characteristics and constraints in more detail.

6.1.3 Inheritance

The property of **inheritance** enables an entity subtype to inherit the attributes and relationships of the supertype. As discussed earlier, a supertype contains those attributes that are common to all of its subtypes. In contrast, subtypes contain only the attributes that are unique to the subtype. For example, Figure 6.2 illustrates that pilots, mechanics, and accountants all inherit the employee number, last name, first name, middle initial, hire date, and so on from the EMPLOYEE entity. However, Figure 6.2 also illustrates that pilots have attributes that are unique; the same is true for mechanics and accountants. *One important inheritance characteristic is that all entity subtypes inherit their primary key attribute from their supertype.* Note in Figure 6.2 that the EMP_NUM attribute is the primary key for each of the subtypes.

At the implementation level, the supertype and its subtype(s) depicted in the specialization hierarchy maintain a 1:1 relationship. For example, the specialization hierarchy lets you replace the undesirable EMPLOYEE table structure in Figure 6.1 with two tables—one representing the supertype EMPLOYEE and the other representing the subtype PILOT. (See Figure 6.3.)

FIGURE 6.3 The EMPLOYEE-PILOT supertype-subtype relationship

Table Name: EMPLOYEE

EMP_NUM	EMP_LNAME	EMP_FNAME	EMP_INITIAL	EMP_HIRE_DATE	EMP_TYPE
100	Kolmycz	Xavier	T	15-Mar-88	
101	Lewis	Marcos		25-Apr-89	P
102	Vandam	Jean		20-Dec-93	A
103	Jones	Victoria	R	28-Aug-03	
104	Lange	Edith		20-Oct-97	P
105	Williams	Gabriel	U	08-Nov-97	P
106	Duzak	Mario		05-Jan-04	P
107	Diante	Venite	L	02-Jul-97	M
108	Wiesenbach	Joni		18-Nov-95	M
109	Travis	Brett	T	14-Apr-01	P
110	Genkazi	Stan		01-Dec-03	A

Table Name: PILOT

EMP_NUM	PIL_LICENSE	PIL_RATINGS	PIL_MED_TYPE
101	ATP	SEL/MEL/Instr/CFII	1
104	ATP	SEL/MEL/Instr	1
105	COM	SEL/MEL/Instr/CFI	2
106	COM	SEL/MEL/Instr	2
109	COM	SEL/MEL/SES/Instr/CFII	1

Entity subtypes inherit all relationships in which the supertype entity participates. For example, Figure 6.2 shows the EMPLOYEE entity supertype participating in a 1:M relationship with a DEPENDENT entity. Through inheritance, all subtypes also participate in that relationship. In specialization hierarchies with multiple levels of supertype/subtypes, a lower-level subtype inherits all of the attributes and relationships from all of its upper-level supertypes.

6.1.4 Subtype Discriminator

A **subtype discriminator** is the attribute in the supertype entity that determines to which subtype the supertype occurrence is related. As seen in Figure 6.2, the subtype discriminator is the employee type (EMP_TYPE).

It is common practice to show the subtype discriminator and its value for each subtype in the ER diagram, as seen in Figure 6.2. However, not all ER modeling tools follow that practice. For example, MS Visio shows the subtype discriminator, but not its value. In Figure 6.2, the Visio text tool was used to manually add the discriminator value above the entity subtype, close to the connector line. Using Figure 6.2 as your guide, note that the supertype is related to a PILOT subtype if the EMP_TYPE has a value of "P." If the EMP_TYPE value is "M," the supertype is related to a MECHANIC subtype. And if the EMP_TYPE value is "A," the supertype is related to the ACCOUNTANT subtype.

It's important to note that the default comparison condition for the subtype discriminator attribute is the equality comparison. However, there may be situations in which the subtype discriminator is not necessarily based on an equality comparison. For example, based on business requirements, you might create two new pilot subtypes, PIC (pilot-in-command)-qualified and copilot-qualified only. A PIC-qualified pilot will be anyone with more than 1,500 PIC flight hours. In this case, the subtype discriminator would be "Flight_Hours," and the criteria would be > 1,500 or <= 1,500, respectively.

Note

In Visio, you select the subtype discriminator when creating a category using the Category shape from the available shapes. The Category shape is a small circle with a horizontal line under it that connects the supertype to its subtypes.

Online Content

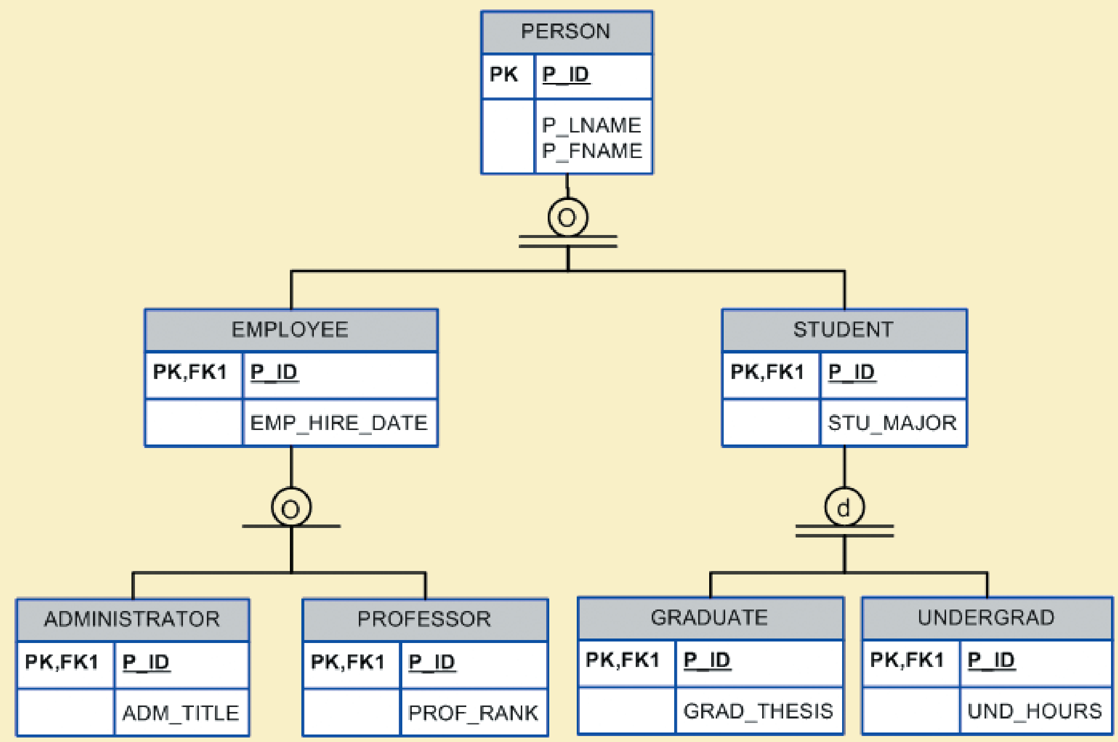
For a tutorial on using MS Visio to create a specialization hierarchy, see **Appendix A, Designing Databases with Visio Professional: A Tutorial,** in the Student Online Companion for this book.

6.1.5 Disjoint and Overlapping Constraints

An entity supertype can have disjoint or overlapping entity subtypes. For example, in the aviation example, an employee can be a pilot or a mechanic or an accountant. Assume that one of the business rules dictates that an employee cannot belong to more than one subtype at a time; that is, an employee cannot be a pilot and a mechanic at the same time. **Disjoint subtypes**, also known as **non-overlapping subtypes**, are subtypes that contain a *unique* subset of the supertype entity set; in other words, each entity instance of the supertype can appear in only one of the subtypes. For example, in Figure 6.2, an employee (supertype) who is a pilot (subtype) can appear only in the PILOT subtype, not in any of the other subtypes. In Visio, such disjoint subtypes are indicated by the letter *d* inside the category shape.

On the other hand, if the business rule specifies that employees can have multiple classifications, the EMPLOYEE supertype may contain *overlapping* job classification subtypes. **Overlapping subtypes** are subtypes that contain nonunique subsets of the supertype entity set; that is, each entity instance of the supertype may appear in more than one subtype. For example, in a university environment, a person may be an employee or a student or both. In turn, an employee may be a professor as well as an administrator. Because an employee also may be a student, STUDENT and EMPLOYEE are overlapping subtypes of the supertype PERSON, just as PROFESSOR and ADMINISTRATOR are overlapping subtypes of the supertype EMPLOYEE. Figure 6.4 illustrates overlapping subtypes with the use of the letter *o* inside the category shape.

FIGURE 6.4 **Specialization hierarchy with overlapping subtypes**



It is common practice to show the disjoint/overlapping symbols in the ERD. (See Figure 6.2 and Figure 6.4.) However, not all ER modeling tools follow that practice. For example, by default, Visio shows only the subtype discriminator (using the Category shape) but not the disjoint/overlapping symbol. Therefore, the Visio text tool was used to manually add the *d* and *o* symbols in Figures 6.2 and 6.4.

NOTE

Alternative notations exist for representing disjoint/overlapping subtypes. For example, Toby J. Teorey popularized the use of G and Gs to indicate disjoint and overlapping subtypes.

As you learned earlier in this section, the implementation of disjoint subtypes is based on the value of the subtype discriminator attribute in the supertype. However, *implementing* overlapping subtypes requires the use of one discriminator attribute for each subtype. For example, in the case of the Tiny College database design you saw in Chapter 4, Entity Relationship (ER) Modeling, a professor can also be an administrator. Therefore, the EMPLOYEE supertype would have the subtype discriminator attributes and values shown in Table 6.1.

TABLE 6.1 **Discriminator Attributes with Overlapping Subtypes**

DISCRIMINATOR ATTRIBUTES		COMMENT
PROFESSOR	ADMINISTRATOR	
"Y"	"N"	The Employee is a member of the Professor subtype.
"N"	"Y"	The Employee is a member of the Administrator subtype.
"Y"	"Y"	The Employee is both a Professor and an Administrator.

6.1.6 Completeness Constraint

The **completeness constraint** specifies whether each entity supertype occurrence must also be a member of at least one subtype. The completeness constraint can be partial or total. **Partial completeness** (symbolized by a circle over a single line) means that not every supertype occurrence is a member of a subtype; that is, there may be some supertype occurrences that are not members of any subtype. **Total completeness** (symbolized by a circle over a double line) means that every supertype occurrence must be a member of at least one subtype.

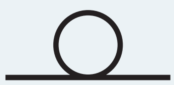
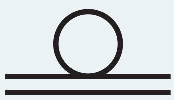
The ERDs in Figures 6.2 and 6.4 represent the completeness constraint based on the Visio Category shape. A single horizontal line under the circle represents a partial completeness constraint; a double horizontal line under the circle represents a total completeness constraint.

Note

Alternative notations exist to represent the completeness constraint. For example, some notations use a single line (partial) or double line (total) to connect the supertype to the Category shape.

Given the disjoint/overlapping subtypes and completeness constraints, it's possible to have the specialization hierarchy constraint scenarios shown in Table 6.2.

TABLE 6.2 Specialization Hierarchy Constraint Scenarios

TYPE	DISJOINT CONSTRAINT	OVERLAPPING CONSTRAINT
Partial 	Supertype has optional subtypes. Subtype discriminator can be null. Subtype sets are unique.	Supertype has optional subtypes. Subtype discriminators can be null. Subtype sets are not unique.
Total 	Every supertype occurrence is a member of a (at least one) subtype. Subtype discriminator cannot be null. Subtype sets are unique.	Every supertype occurrence is a member of a (at least one) subtype. Subtype discriminators cannot be null. Subtype sets are not unique.

6.1.7 Specialization and Generalization

You can use various approaches to develop entity supertypes and subtypes. For example, you can first identify a regular entity, and then identify all entity subtypes based on their distinguishing characteristics. You also can start by identifying multiple entity types and then later extract the common characteristics of those entities to create a higher-level supertype entity.

Specialization is the top-down process of identifying lower-level, more specific entity subtypes from a higher-level entity supertype. Specialization is based on grouping unique characteristics and relationships of the subtypes. In the aviation example, you used specialization to identify multiple entity subtypes from the original employee supertype. **Generalization** is the bottom-up process of identifying a higher-level, more generic entity supertype from lower-level entity subtypes. Generalization is based on grouping common characteristics and relationships of the subtypes. For example, you might identify multiple types of musical instruments: piano, violin, and guitar. Using the generalization approach, you could identify a "string instrument" entity supertype to hold the common characteristics of the multiple subtypes.

6.2 ENTITY CLUSTERING

Developing an ER diagram entails the discovery of possibly hundreds of entity types and their respective relationships. Generally, the data modeler will develop an initial ERD containing a few entities. As the design approaches completion, the ERD will contain hundreds of entities and relationships that crowd the diagram to the point of making it unreadable and inefficient as a communication tool. In those cases, you can use entity clusters to minimize the number of entities shown in the ERD.

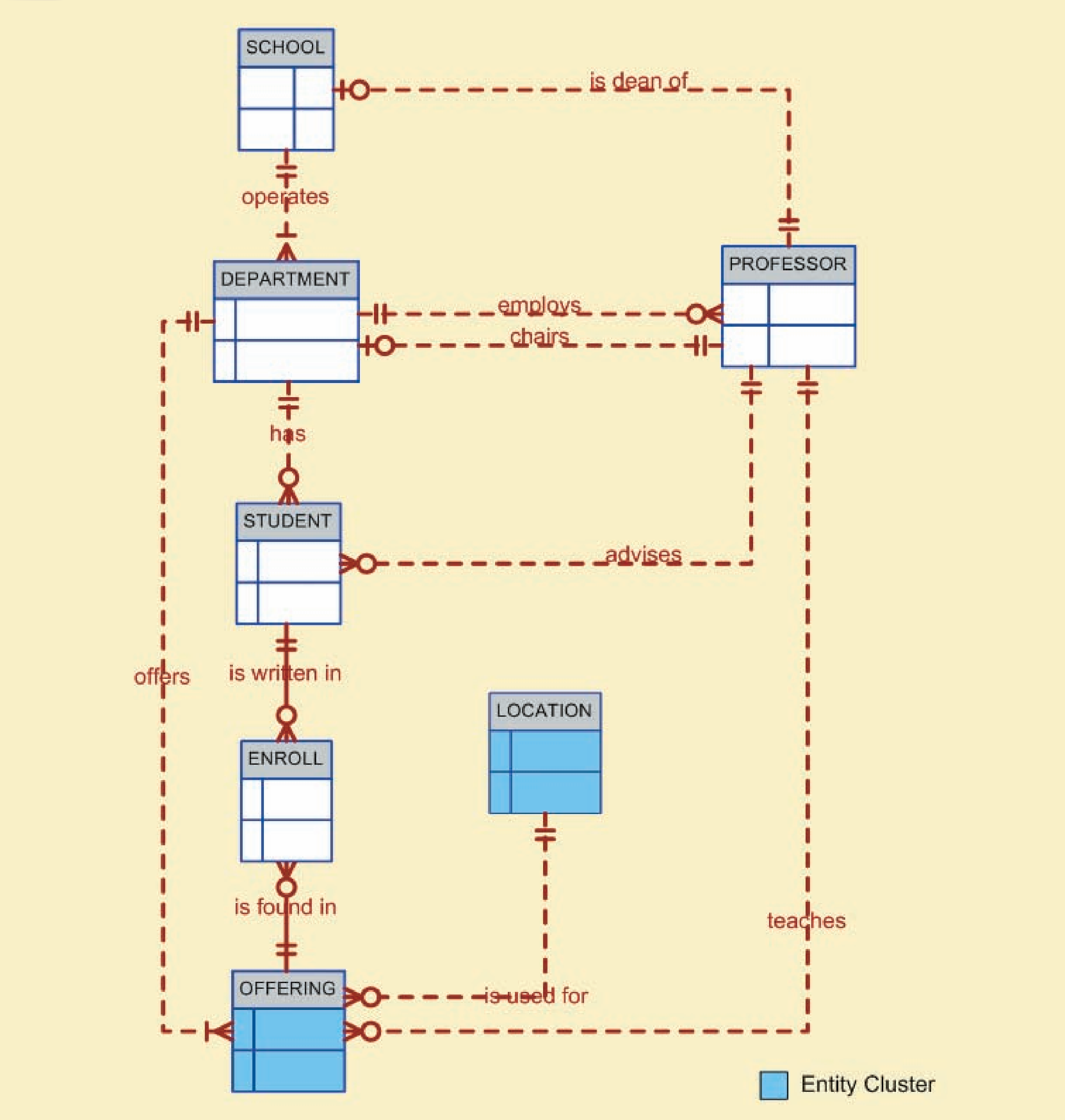
An **entity cluster** is a "virtual" entity type used to represent multiple entities and relationships in the ERD. An entity cluster is formed by combining multiple interrelated entities into a single abstract entity object. An entity cluster is considered "virtual" or "abstract" in the sense that it is not actually an entity in the final ERD. Instead, it is a temporary entity used to represent multiple entities and relationships, with the purpose of simplifying the ERD and thus enhancing its readability.

Figure 6.5 illustrates the use of entity clusters based on the Tiny College example in Chapter 4. Note that the ERD contains two entity clusters:

- OFFERING, which groups the COURSE and CLASS entities and relationships.
- LOCATION, which groups the ROOM and BUILDING entities and relationships.

Note also that the ERD in Figure 6.5 does not show attributes for the entities. When using entity clusters, the key attributes of the combined entities are no longer available. Without the key attributes, primary key inheritance rules change. In turn, the change in the inheritance rules can have undesirable consequences, such as changes in relationships—from identifying to nonidentifying or vice versa—and the loss of foreign key attributes from some entities. To eliminate those problems, the general rule is to *avoid the display of attributes when entity clusters are used*.

FIGURE 6.5 Tiny College ERD using entity clusters



6.3 ENTITY INTEGRITY: SELECTING PRIMARY KEYS

Arguably, the most important characteristic of an entity is its primary key (a single attribute or some combination of attributes), which uniquely identifies each entity instance. The primary key's function is to guarantee entity integrity. Furthermore, primary keys and foreign keys work together to implement relationships in the relational model. Therefore, the importance of properly selecting the primary key has a direct bearing on the efficiency and effectiveness of database implementation.

6.3.1 Natural Keys and Primary Keys

The concept of a unique identifier is commonly encountered in the real world. For example, you use class (or section) numbers to register for classes, invoice numbers to identify specific invoices, account numbers to identify credit cards, and so on. Those examples illustrate natural identifiers or keys. A **natural key** or **natural identifier** is a real-world, generally accepted identifier used to distinguish—that is, uniquely identify—real-world objects. As its name implies, a natural key is familiar to end users and forms part of their day-to-day business vocabulary.

Usually, if an entity *has* a natural identifier, a data modeler uses that as the primary key of the entity being modeled. Generally, most natural keys make acceptable primary key identifiers. However, there are occasions when the entity being modeled does not have a natural primary key, or the natural key is not a *good* primary key. For example, assume an ASSIGNMENT entity composed of the following attributes:

ASSIGNMENT (ASSIGN_DATE, PROJ_NUM, EMP_NUM, ASSIGN_HOURS, ASSIGN_CHG_HOUR, ASSIGN_CHARGE)

What attribute (or combination of attributes) would make a good primary key? You learned in Chapter 5, Normalization of Database Tables, that tradeoffs were associated with the selection of various combinations of attributes to serve as the primary key for the ASSIGNMENT table. You also learned about the use of surrogate keys. Given that knowledge, is the composite primary key (ASSIGN_DATE, PROJ_NUM, EMP_NUM) a *good* primary key? Or would a surrogate key be a better choice? Why? The next section presents some basic guidelines for selecting primary keys.

6.3.2 Primary Key Guidelines

A primary key is the attribute or combination of attributes that uniquely identifies entity instances in an entity set. However, can the primary key be based on, say, 12 attributes? And just how long can a primary key be? In previous examples, why was EMP_NUM selected as a primary key of EMPLOYEE and not a combination of EMP_LNAME, EMP_FNAME, EMP_INITIAL, and EMP_DOB? Can a single 256-byte text attribute be a good primary key? There is no single answer to those questions, but there is a body of practice that database experts have built over the years. This section examines that body of documented practices.

First, you should understand the function of a primary key. The primary key's main function is to uniquely identify an entity instance or row within a table. In particular, given a primary key value—that is, the determinant—the relational model can determine the value of all dependent attributes that "describe" the entity. Note that "*identification*" and "*description*" are separate semantic constructs in the model. *The function of the primary key is to guarantee entity integrity, not to "describe" the entity.*

Second, primary keys and foreign keys are used to implement relationships among entities. However, the implementation of such relationships is done mostly behind the scenes, hidden from end users. In the real world, end users identify objects based on the characteristics they know about the objects. For example, when shopping at a grocery store, you select products by taking them from a store display shelf and reading the labels, not by looking at the stock number. It's wise for database applications to mimic the human selection process as much as possible. Therefore, database applications should let the end user choose among multiple descriptive narratives of different objects while using primary key values behind the scenes. Keeping those concepts in mind, look at Table 6.3, which summarizes desirable primary key characteristics.

TABLE 6.3 Desirable Primary Key Characteristics

PK CHARACTERISTIC	RATIONALE
Unique values	The PK must uniquely identify each entity instance. A primary key must be able to guarantee unique values. It cannot contain nulls.
Nonintelligent	The PK should not have embedded semantic meaning (factless). An attribute with embedded semantic meaning is probably better used as a descriptive characteristic of the entity rather than as an identifier. In other words, a student ID of 650973 would be preferred over Smith, Martha L. as a primary key identifier. In short, the PK should be factless.
No change over time	If an attribute has semantic meaning, it might be subject to updates. This is why names do not make good primary keys. If you have Vickie Smith as the primary key, what happens when she gets married? If a primary key is subject to change, the foreign key values must be updated, thus adding to the database work load. Furthermore, changing a primary key value means that you are basically changing the identity of an entity. In short, the PK should be permanent and unchangeable.
Preferably single-attribute	A primary key should have the minimum number of attributes possible (irreducible). Single-attribute primary keys are desirable but not required. Single-attribute primary keys simplify the implementation of foreign keys. Having multiple-attribute primary keys can cause primary keys of related entities to grow through the possible addition of many attributes, thus adding to the database work load and making (application) coding more cumbersome.
Preferably numeric	Unique values can be better managed when they are numeric, because the database can use internal routines to implement a counter-style attribute that automatically increments values with the addition of each new row. In fact, most database systems include the ability to use special constructs, such as Autonumber in Microsoft Access, to support self-incrementing primary key attributes.
Security compliant	The selected primary key must not be composed of any attribute(s) that might be considered a security risk or violation. For example, using a Social Security number as a PK in an EMPLOYEE table is not a good idea.

6.3.3 When to Use Composite Primary Keys

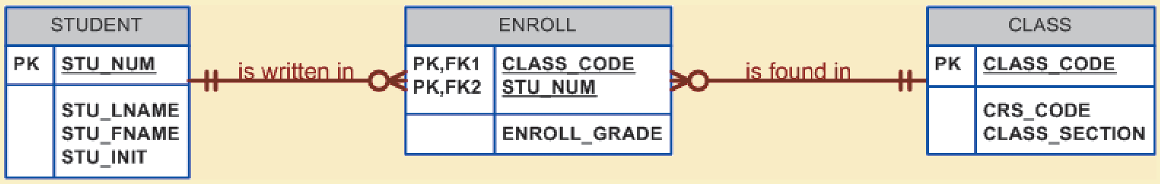
In the previous section, you learned about the desirable characteristics of primary keys. For example, you learned that the primary key should use the minimum number of attributes possible. However, that does *not* mean that composite primary keys are not permitted in a model. In fact, composite primary keys are particularly useful in two cases:

- As identifiers of composite entities, where each primary key combination is allowed only once in the M:N relationship.
- As identifiers of weak entities, where the weak entity has a strong identifying relationship with the parent entity.

To illustrate the first case, assume that you have a STUDENT entity set and a CLASS entity set. In addition, assume that those two sets are related in an M:N relationship via an ENROLL entity set in which each student/class combination may appear only once in the composite entity. Figure 6.6 shows the ERD to represent such a relationship.

As shown in Figure 6.6, the composite primary key automatically provides the benefit of ensuring that there cannot be duplicate values—that is, it ensures that the same student cannot enroll more than once in the same class.

FIGURE 6.6 The M:N relationship between STUDENT and CLASS



STU_NUM	STU_LNAME	STU_FNAME	STU_INIT
321452	Bowser	William	C
324257	Smithson	Anne	K
324258	Brewer	Juliette	
324269	Oblonski	Walter	H
324273	Smith	John	D
324274	Katinga	Raphael	P
324291	Robertson	Gerald	T
324299	Smith	John	B

CLASS_CODE	STU_NUM	ENROLL_GRADE
10014	321452	C
10014	324257	B
10018	321452	A
10018	324257	B
10021	321452	C
10021	324257	C

CLASS_CODE	CRS_CODE	CLASS_SECTION
10012	ACCT-211	1
10013	ACCT-211	2
10014	ACCT-211	3
10015	ACCT-212	1
10016	ACCT-212	2
10017	CIS-220	1
10018	CIS-220	2
10019	CIS-220	3
10020	CIS-420	1
10021	QM-261	1
10022	QM-261	2
10023	QM-362	1
10024	QM-362	2
10025	MATH-243	1

In the second case, a weak entity in a strong identifying relationship with a parent entity is normally used to represent one of two situations:

1. *A real-world object that is existent-dependent on another real-world object*. Those types of objects are distinguishable in the real world. A dependent and an employee are two separate people who exist independent of each other. However, such objects can exist in the model only when they relate to each other in a strong identifying relationship. For example, the relationship between EMPLOYEE and DEPENDENT is one of existence dependency in which the primary key of the dependent entity is a composite key that contains the key of the parent entity.
2. *A real-world object that is represented in the data model as two separate entities in a strong identifying relationship*. For example, the real-world invoice object is represented by two entities in a data model: INVOICE and LINE. Clearly, the LINE entity does not exist in the real world as an independent object, but rather as part of an INVOICE.

In both situations, having a strong identifying relationship ensures that the dependent entity can exist only when it is related to the parent entity. In summary, the selection of a composite primary key for composite and weak entity types provides benefits that enhance the integrity and consistency of the model.

6.3.4 When to Use Surrogate Primary Keys

There are some instances when a primary key doesn't exist in the real world or when the existing natural key might not be a suitable primary key. For example, consider the case of a park recreation facility that rents rooms for small parties. The manager of the facility keeps track of all events, using a folder with the format shown in Table 6.4.

TABLE 6.4 Data Used to Keep Track of Events

DATE	TIME_START	TIME_END	ROOM	EVENT_NAME	PARTY_OF
6/17/08	11:00AM	2:00PM	Allure	Burton Wedding	60
6/17/08	11:00AM	2:00PM	Bonanza	Adams Office	12
6/17/08	3:00PM	5:30PM	Allure	Smith Family	15
6/17/08	3:30PM	5:30PM	Bonanza	Adams Office	12
6/18/08	1:00PM	3:00PM	Bonanza	Boy Scouts	33
6/18/08	11:00AM	2:00PM	Allure	March of Dimes	25
6/18/08	11:00AM	12:30PM	Bonanza	Smith Family	12

Given the data shown in Table 6.4, you would model the EVENT entity as:

EVENT (DATE, TIME_START, TIME_END, ROOM, EVENT_NAME, PARTY_OF)

What primary key would you suggest? In this case, there is no simple natural key that could be used as a primary key in the model. Based on the primary key concepts you learned about in previous chapters, you might suggest one of these options:

(**DATE**, **TIME_START**, **ROOM**) or (**DATE**, **TIME_END**, **ROOM**)

Assume you select the composite primary key (**DATE**, **TIME_START**, **ROOM**) for the EVENT entity. Next, you determine that one EVENT may use many RESOURCEs (such as tables, projectors, PCs, and stands), and that the same RESOURCE may be used for many EVENTs. The RESOURCE entity would be represented by the following attributes:

RESOURCE (**RSC_ID**, RSC_DESCRIPTION, RSC_TYPE, RSC_QTY, RSC_PRICE)

Given the business rules, the M:N relationship between RESOURCE and EVENT would be represented via the EVNTRSC composite entity with a composite primary key as follows:

EVNTRSC (**DATE, TIME_START, ROOM, RSC_ID,** QTY_USED)

You now have a lengthy four-attribute composite primary key. What would happen if the EVNTRSC entity's primary key were inherited by another existence-dependent entity? At this point, you can see that the composite primary key could make the implementation of the database and program coding unnecessarily complex.

As a data modeler, you probably noticed that the EVENT entity's selected primary key might not fare well, given the primary key guidelines in Table 6.3. In this case, the EVENT entity's selected primary key contains embedded semantic information and is formed by a combination of date, time, and text data columns. In addition, the selected primary key would cause lengthy primary keys for existence-dependent entities. The solution to the problem is to use a numeric single-attribute surrogate primary key.

Surrogate primary keys are accepted practice in today's complex data environments. They are especially helpful when there is no natural key, when the selected candidate key has embedded semantic contents, or when the selected candidate key is too long or cumbersome. However, there is a tradeoff: if you use a surrogate key, you must ensure that the candidate key of the entity in question performs properly through the use of "unique index" and "not null" constraints.

6.4 DESIGN CASES: LEARNING FLEXIBLE DATABASE DESIGN

Data modeling and design require skills that are acquired through experience. In turn, experience is acquired through practice—regular and frequent repetition, applying the concepts learned to specific and different design problems. This section presents four special design cases that highlight the importance of flexible designs, proper identification of primary keys, and placement of foreign keys.

NOTE

In describing the various modeling concepts throughout this book, the focus is on relational models. Also, given the focus on the practical nature of database design, all design issues are addressed with the implementation goal in mind. Therefore, there is no sharp line of demarcation between design and implementation.

At the pure conceptual stage of the design, foreign keys are not part of an ER diagram. The ERD displays only entities and relationships. Entities are identified by identifiers that may become primary keys. During design, the modeler attempts to understand and define the entities and relationships. Foreign keys are the mechanism through which the relationship *designed* in an ERD is *implemented* in a relational model. If you use Visio Professional as your modeling tool, you will discover that this book's methodology is reflected in the Visio modeling practice.

6.4.1 Design Case #1: Implementing 1:1 Relationships

Foreign keys work with primary keys to properly implement relationships in the relational model. The basic rule is very simple: put the primary key of the "one" side (the parent entity) on the "many" side (the dependent entity) as a foreign key. However, where do you place the foreign key when you are working with a 1:1 relationship? For example, take the case of a 1:1 relationship between EMPLOYEE and DEPARTMENT based on the business rule "one EMPLOYEE is the manager of one DEPARTMENT, and one DEPARTMENT is managed by one EMPLOYEE." In that case, there are two options for selecting and placing the foreign key:

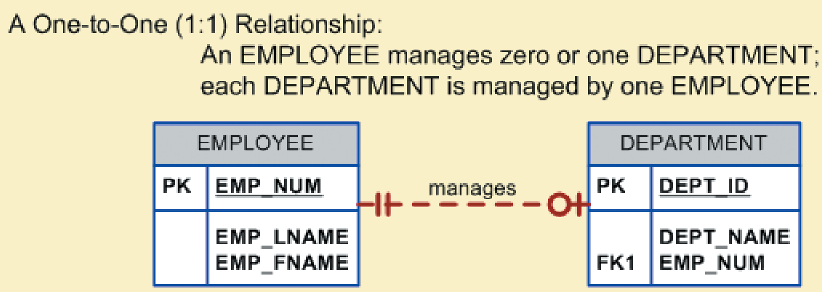
1. *Place a foreign key in both entities*. This option is derived from the basic rule you learned in Chapter 4. Place EMP_NUM as a foreign key in DEPARTMENT, and place DEPT_ID as a foreign key in EMPLOYEE. However, that solution is not recommended, as it would create duplicated work, and it could conflict with other existing relationships. (Remember that DEPARTMENT and EMPLOYEE also participate in a 1:M relationship—one department employs many employees.)
2. *Place a foreign key in one of the entities*. In that case, the primary key of one of the two entities appears as a foreign key in the other entity. That is the preferred solution, but there is a remaining question: *which* primary key should be used as a foreign key? The answer to that question is found in Table 6.5. Table 6.5 shows the rationale for selecting the foreign key in a 1:1 relationship based on the relationship properties in the ERD.

TABLE 6.5 Selection of Foreign Key in a 1:1 Relationship

CASE	ER RELATIONSHIP CONSTRAINTS	ACTION
I	One side is mandatory and the other side is optional.	Place the PK of the entity on the mandatory side in the entity on the optional side as an FK, and make the FK mandatory.
II	Both sides are optional.	Select the FK that causes the fewest number of nulls, or place the FK in the entity in which the (relationship) role is played.
III	Both sides are mandatory.	See Case II, or consider revising your model to ensure that the two entities do not belong together in a single entity.

Figure 6.7 illustrates the "EMPLOYEE manages DEPARTMENT" relationship. Note that in this case, EMPLOYEE is mandatory to DEPARTMENT. Therefore, EMP_NUM is placed as the foreign key in DEPARTMENT. Alternatively, you might also argue that the "manager" role is played by the EMPLOYEE in the DEPARTMENT.

FIGURE 6.7 **A 1:1 relationship between DEPARTMENT and EMPLOYEE**



As a designer, you must recognize that 1:1 relationships exist in the real world, and therefore, they should be supported in the data model. In fact, a 1:1 relationship is used to ensure that two entity sets are not placed in the same table. In other words, EMPLOYEE and DEPARTMENT are clearly separate and unique entity types that do not belong together in a single entity. If you group them together in one entity, what would be the name of that entity?

6.4.2 Design Case #2: Maintaining History of Time-Variant Data

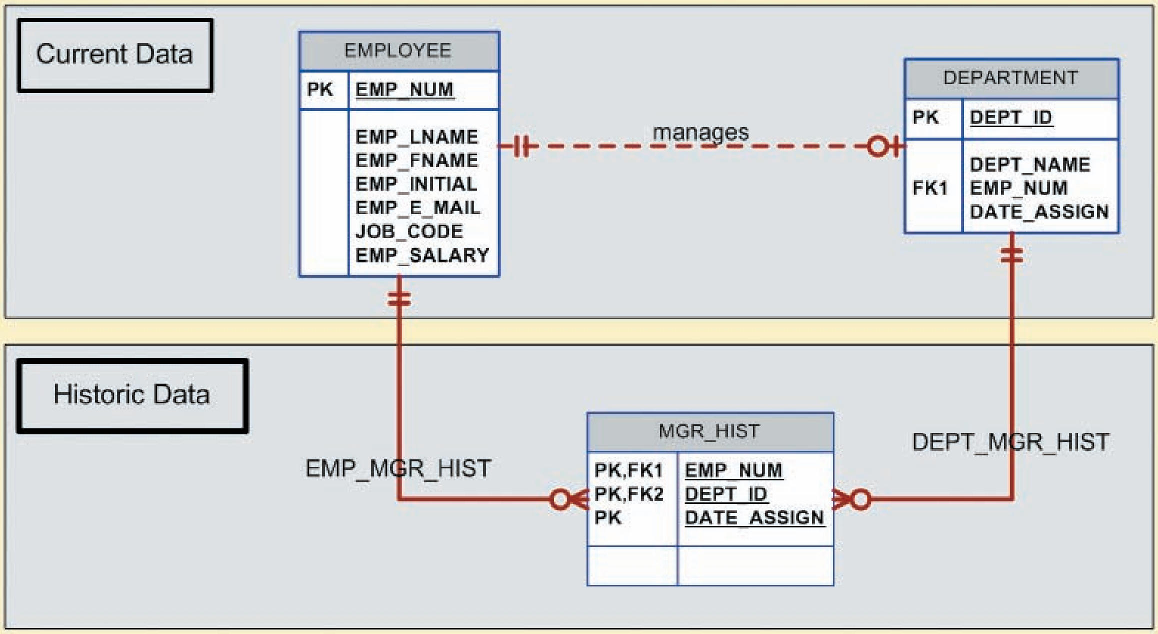
Company managers generally realize that good decision making is based on the information that is generated through the data stored in databases. Such data reflect current as well as past events. In fact, company managers use the data stored in databases to answer questions such as, "How do the current company profits compare to those of previous years?" and, "What are XYZ product's sales trends?" In other words, the data stored on databases reflect not only current data, but also historic data.

Normally, data changes are managed by replacing the existing attribute value with the new value, without regard to the previous value. However, there are situations when the history of values for a given attribute must be preserved. From a data modeling point of view, **time-variant data** refer to data whose values change over time and for which you *must* keep a history of the data changes. You could argue that all data in a database are subject to change over time and are, therefore, time variant. However, some attribute values, such as your date of birth or your Social Security number, are not time variant. On the other hand, attributes such as your student GPA or your bank account balance are subject to change over time. Sometimes the data changes are externally originated and event driven, such as a product price change. On other occasions, changes are based on well-defined schedules, such as the daily stock quote "open" and "close" values.

In any case, keeping the history of time-variant data is equivalent to having a multivalued attribute in your entity. To model time-variant data, you must create a new entity in a 1:M relationship with the original entity. This new entity will contain the new value, the date of the change, and whatever other attribute is pertinent to the event being modeled. For example, if you want to keep track of the current manager as well as the history of all department managers over time, you could create the model shown in Figure 6.8.

Note that in Figure 6.8, the MGR_HIST entity has a 1:M relationship with EMPLOYEE and a 1:M relationship with DEPARTMENT to reflect the fact that, over time, an employee could be the manager of many different departments, and a department could have many different employee managers. Because you are recording time-variant data, you must store the DATE_ASSIGN attribute in the MGR_HIST entity to provide the date on which the employee (EMP_NUM) became the manager of the department. The primary key of MGR_HIST permits the same employee to

FIGURE 6.8 **Maintaining manager history**



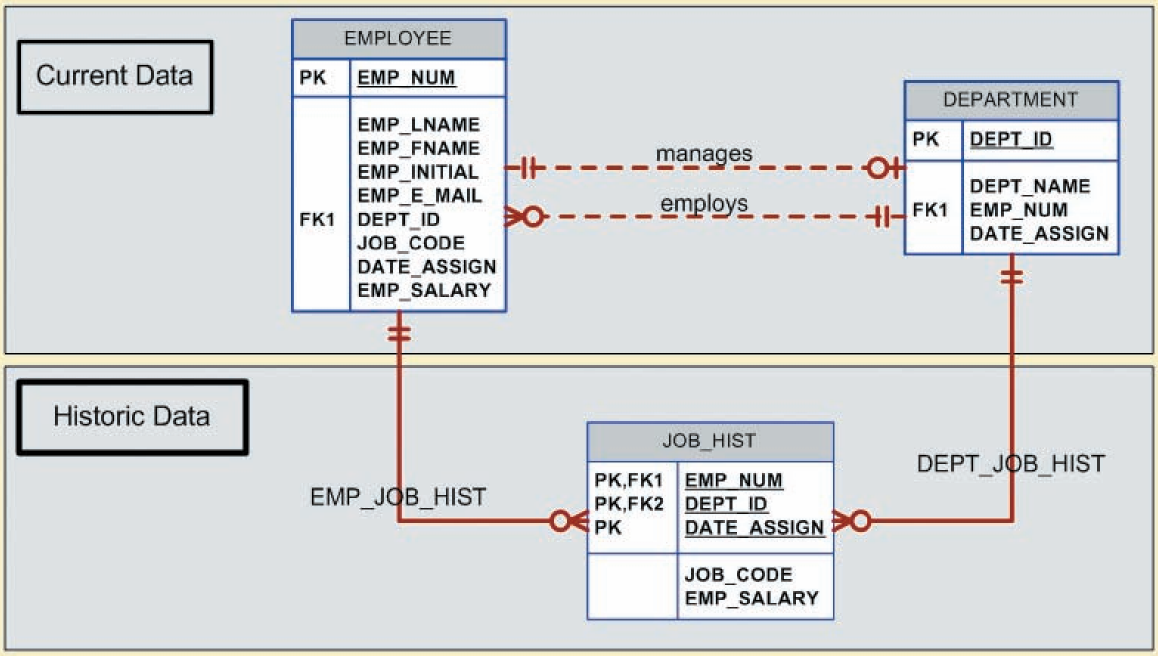
be the manager of the same department, but on different dates. If that scenario is not the case in your environment—if, for example, an employee is the manager of a department only once—you could make DATE_ASSIGN a nonprime attribute in the MGR_HIST entity.

Note in Figure 6.8 that the "manages" relationship is optional in theory and redundant in practice. At any time, you could find out who the manager of a department is by retrieving the most recent DATE_ASSIGN date from MGR_HIST for a given department. On the other hand, the ERD in Figure 6.8 differentiates between current data and historic data. The *current* manager relationship is implemented by the "manages" relationship between EMPLOYEE and DEPARTMENT. Additionally, the historic data are managed through EMP_MGR_HIST and DEPT_MGR_HIST. The trade-off with that model is that each time a new manager is assigned to a department, there will be two data modifications: one update in the DEPARTMENT entity and one insert in the MGR_HIST entity.

The flexibility of the model proposed in Figure 6.8 becomes more apparent when you add the 1:M "one department employs many employees" relationship. In that case, the PK of the "1" side (DEPT_ID) appears in the "many" side (EMPLOYEE) as a foreign key. Now suppose you would like to keep track of the job history for each of the company's employees—you'd probably want to store the department, the job code, the date assigned, and the salary. To accomplish that task, you would modify the model in Figure 6.8 by adding a JOB_HIST entity. Figure 6.9 shows the use of the new JOB_HIST entity to maintain the employee's history.

Again, it's worth emphasizing that the "manages" and "employs" relationships are theoretically optional and redundant in practice. You can always find out where each employee works by looking at the job history and selecting only the most current data row for each employee. However, as you will discover in Chapter 7, An Introduction to Structured Query Language (SQL), and in Chapter 8, Advanced SQL, finding where each employee works is not a trivial task. Therefore, the model represented in Figure 6.9 includes the admittedly redundant but unquestionably useful "manages" and "employs" relationships to separate current data from historic data.

FIGURE 6.9 Maintaining job history



6.4.3 Design Case #3: Fan Traps

Creating a data model requires proper identification of the data relationships among entities. However, due to miscommunication or incomplete understanding of the business rules or processes, it is not uncommon to misidentify relationships among entities. Under those circumstances, the ERD may contain a design trap. A **design trap** occurs when a relationship is improperly or incompletely identified and is therefore represented in a way that is not consistent with the real world. The most common design trap is known as a *fan trap*.

A **fan trap** occurs when you have one entity in two 1:M relationships to other entities, thus producing an association among the other entities that is not expressed in the model. For example, assume the JCB basketball league has many divisions. Each division has many players, and each division has many teams. Given those “incomplete” business rules, you might create an ERD that looks like the one in Figure 6.10.

As you can see in Figure 6.10, DIVISION is in a 1:M relationship with TEAM and in a 1:M relationship with PLAYER. Although that representation is semantically correct, the relationships are not properly identified. For example, there is no way to identify what players belong to what team. Figure 6.10 also shows a sample instance relationship representation for the ERD. Note that the relationship lines for the DIVISION instances fan out to the TEAM and PLAYER entity instances—thus the “fan trap” label.

Figure 6.11 shows the correct ERD after the fan trap has been eliminated. Note that, in this case, DIVISION is in a 1:M relationship with TEAM. In turn, TEAM is in a 1:M relationship with PLAYER. Figure 6.11 also shows the instance relationship representation after eliminating the fan trap.

Given the design in Figure 6.11, note how easy it is to see which players play for which team. However, to find out which players play in which division, you first need to see what teams belong to each division; then you need to find out what players play on each team. In other words, there is a transitive relationship between DIVISION and PLAYER via the TEAM entity.

FIGURE 6.10 Incorrect ERD with fan trap problem

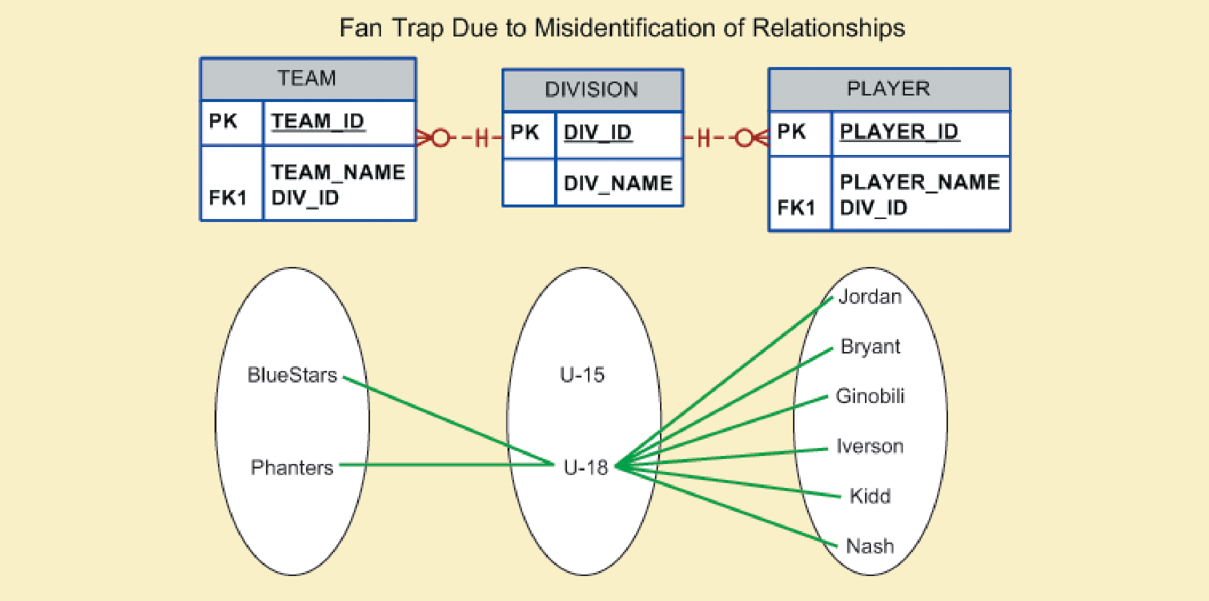
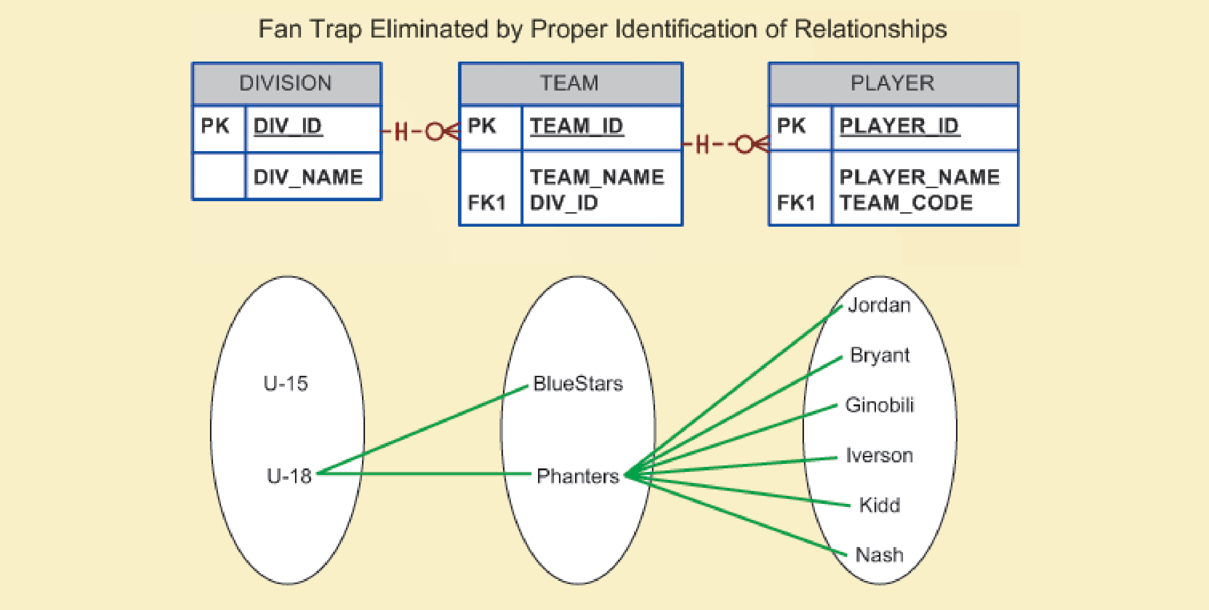


FIGURE 6.11 Corrected ERD after removal of the fan trap

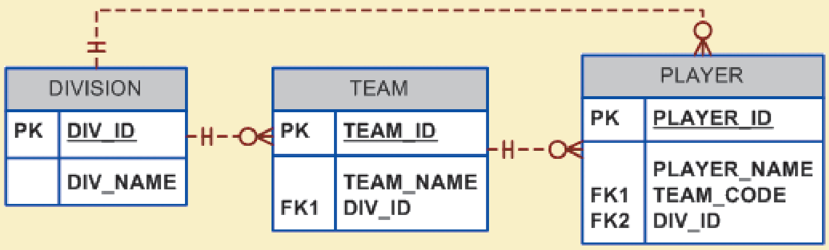


6.5.4 Design Case #4: Redundant Relationships

Although redundancy is often a good thing to have in computer environments (multiple backups in multiple places, for example), redundancy is seldom a good thing in the database environment. (As you learned in Chapter 3, The Relational Database Model, redundancies can cause data anomalies in a database.) Redundant relationships occur when there are multiple relationship paths between related entities. The main concern with redundant relationships is that they remain consistent across the model. However, it's important to note that some designs use redundant relationships as a way to simplify the design.

An example of redundant relationships was first introduced in Figure 6.8 during the discussion on maintaining a history of time-variant data. However, the use of the redundant "manages" and "employs" relationships was justified by the fact that such relationships were dealing with current data rather than historic data. Another more specific example of a redundant relationship is represented in Figure 6.12.

FIGURE 6.12 A redundant relationship



In Figure 6.12, note the transitive 1:M relationship between DIVISION and PLAYER through the TEAM entity set. Therefore, the relationship that connects DIVISION and PLAYER is, for all practical purposes, redundant. In that case, the relationship could be safely deleted without losing any information-generation capabilities in the model.

6.5 DATA MODELING CHECKLIST

Data modeling translates a specific real-world environment into a data model that represents the real-world data, users, processes, and interactions. You learned in this chapter how the EERM enables the designer to add more semantic content to the model. You also learned about the trade-offs and intricacies in the selection of primary keys, and you studied the modeling of time-variant data. The modeling techniques you have learned thus far give you the tools needed to produce successful database designs. However, just as any good pilot uses a checklist to ensure that all is in order for a successful flight, the data modeling checklist shown in Table 6.6 will help ensure that you perform data modeling tasks successfully. (The data modeling checklist in Table 6.6 is based on the concepts and tools you learned beginning in Chapter 3—the relational model, the entity relationship model, normalization, and the extended entity relationship model.) Therefore, it is assumed that you are familiar with the terms and labels used in the checklist, such as *synonyms, aliases,* and *3NF*.

TABLE 6.6 Data Modeling Checklist

BUSINESS RULES

- Properly document and verify all business rules with the end users.
- Ensure that all business rules are written precisely, clearly, and simply. The business rules must help identify entities, attributes, relationships, and constraints.
- Identify the source of all business rules, and ensure that each business rule is accompanied by the reason for its existence and by the date and person(s) responsible for verifying and approving the business rule.

DATA MODELING

Naming Conventions: All names should be limited in length (database-dependent size).

- Entity Names:
  - Should be nouns that are familiar to business and should be short and meaningful
  - Should include abbreviations, synonyms, and aliases for each entity
  - Should be unique within the model
  - For composite entities, may include a combination of abbreviated names of the entities linked through the composite entity
- Attribute Names:
  - Should be unique within the entity
  - Should use the entity abbreviation or prefix
  - Should be descriptive of the characteristic
  - Should use suffixes such as _ID, _NUM, or _CODE for the PK attribute
  - Should not be a reserved word
  - Should not contain spaces or special characters such as @, !, or &
- Relationship Names:
  - Should be active or passive verbs that clearly indicate the nature of the relationship

Entities:

- Each entity should represent a single subject.
- Each entity should represent a set of distinguishable entity instances.
- All entities should be in 3NF or higher.
- The granularity of the entity instance is clearly defined.
- The PK is clearly defined and supports the selected data granularity.

Attributes:

- Should be simple and single-valued (atomic data)
- Should include default values, constraints, synonyms, and aliases
- Derived attributes should be clearly identified and include source(s)
- Should not be redundant unless they are required for transaction accuracy or for maintaining a history or are used as a foreign key
- Non-key attributes must be fully dependent on the PK attribute

Relationships:

- Should clearly identify relationship participants
- Should clearly define participation and cardinality rules

ER Model:

- Should be validated against expected processes: inserts, updates, and deletes
- Should evaluate where, when, and how to maintain a history
- Should not contain redundant relationships except as required (see attributes)
- Should minimize data redundancy to ensure single-place updates
- Should conform to the minimal data rule: "All that is needed is there and all that is there is needed."

SUMMARY

- The extended entity relationship (EER) model adds semantics to the ER model via entity supertypes, subtypes, and clusters. An entity supertype is a generic entity type that is related to one or more entity subtypes.
- A specialization hierarchy depicts the arrangement and relationships between entity supertypes and entity subtypes. Inheritance means that an entity subtype inherits the attributes and relationships of the supertype. Subtypes can be disjoint or overlapping. A subtype discriminator is used to determine to which entity subtype the supertype occurrence is related. The subtypes can exhibit partial or total completeness. There are basically two approaches to developing a specialization hierarchy of entity supertypes and subtypes: specialization and generalization.
- An entity cluster is a "virtual" entity type used to represent multiple entities and relationships in the ERD. An entity cluster is formed by combining multiple interrelated entities and relationships into a single, abstract entity object.
- Natural keys are identifiers that exist in the real world. Natural keys sometimes make good primary keys, but this is not necessarily true. Primary keys should have these characteristics: They must have unique values, they should be nonintelligent, they must not change over time, and they are preferably numeric and composed of a single attribute.
- Composite keys are useful to represent M:N relationships and weak (strong-identifying) entities.
- Surrogate primary keys are useful when there is no natural key that makes a suitable primary key, when the primary key is a composite primary key with multiple different data types, or when the primary key is too long to be usable.
- In a 1:1 relationship, place the PK of the mandatory entity as a foreign key in the optional entity, as an FK in the entity that causes the least number of nulls, or as an FK where the role is played.
- Time-variant data refers to data whose values change over time and whose requirements mandate that you keep a history of data changes. To maintain the history of time-variant data, you must create an entity containing the new value, the date of change, and any other time-relevant data. This entity maintains a 1:M relationship with the entity for which the history is to be maintained.
- A fan trap occurs when you have one entity in two 1:M relationships to other entities and there is an association among the other entities that is not expressed in the model. Redundant relationships occur when there are multiple relationship paths between related entities. The main concern with redundant relationships is that they remain consistent across the model.
- The data modeling checklist provides a way for the designer to check that the ERD meets a set of minimum requirements.

KEY TERMS

completeness constraint, 199
design trap, 209
disjoint subtype (non-overlapping subtype), 197
EER diagram (EERD), 194
entity cluster, 200
entity subtype, 194
entity supertype, 194
extended entity relationship model (EERM), 194
fan trap, 209
inheritance, 196
natural key (natural identifier), 202
overlapping subtype, 197
partial completeness, 160
specialization hierarchy, 195
subtype discriminator, 197
time-variant data, 207
total completeness, 199

ONLINE CONTENT

Answers to selected Review Questions and Problems for this chapter are contained in the Student Online Companion for this book.

REVIEW QUESTIONS

1. What is an entity supertype, and why is it used?
2. What kinds of data would you store in an entity subtype?
3. What is a specialization hierarchy?
4. What is a subtype discriminator? Give an example of its use.
5. What is an overlapping subtype? Give an example.
6. What is the difference between partial completeness and total completeness?
7. What is an entity cluster, and what advantages are derived from its use?
8. What primary key characteristics are considered desirable? Explain *why* each characteristic is considered desirable.
9. Under what circumstances are composite primary keys appropriate?
10. What is a surrogate primary key, and when would you use one?
11. When implementing a 1:1 relationship, where should you place the foreign key if one side is mandatory and one side is optional? Should the foreign key be mandatory or optional?
12. What are time-variant data, and how would you deal with such data from a database design point of view?
13. What is the most common design trap, and how does it occur?
14. Using the design checklist shown in this chapter, what naming conventions should you use?
15. Using the design checklist shown in this chapter, what characteristics should entities have?

PROBLEMS

1. AVANTIVE Corporation is a company specializing in the commercialization of automotive parts. AVANTIVE has two types of customers: retail and wholesale. All customers have a customer ID, a name, an address, a phone number, a default shipping address, a date of last purchase, and a date of last payment. Retail customers have the customer attributes, plus the credit card type, credit card number, expiration date, and e-mail address. Wholesale customers have the customer attributes, plus a contact name, contact phone number, contact e-mail address, purchase order number and date, discount percentage, billing address, tax status (if exempt), and tax identification number. A retail customer cannot be a wholesale customer and vice versa. Given that information, create the ERD containing all primary keys, foreign keys, and main attributes.
2. AVANTIVE Corporation has five departments: administration, marketing, sales, shipping, and purchasing. Each department employs many employees. Each employee has an ID, a name, a home address, a home phone number, and a salary and tax ID (Social Security number). Some employees are classified as sales representatives, some as technical support, and some as administrators. Sales representatives receive a commission based on sales. Technical support employees are required to be certified in their areas of expertise. For example, some are certified as drivetrain specialists; others, as electrical systems specialists. All administrators have a title and a bonus. Given that information, create the ERD containing all primary keys, foreign keys, and main attributes.

3. AVANTIVE Corporation operates under the following business rules:
   - AVANTIVE keeps a list of car models with information about the manufacturer, model, and year. AVANTIVE keeps several parts in stock. A part has a part ID, description, unit price, and quantity on hand. A part can be used for many car models, and a car model has many parts.
   - A retail customer normally pays by credit card and is charged the list price for each purchased item. A wholesale customer normally pays via purchase order with terms of net 30 days and is charged a discounted price for each item purchased. (The discount varies from customer to customer.)
   - A customer (retail or wholesale) can place many orders. Each order has an order number; a date; a shipping address; a billing address; and a list of part codes, quantities, unit prices, and extended line totals. Each order also has a sales representative ID (an employee) to identify the person who made the sale, an order subtotal, an order tax total, a shipping cost, a shipping date, an order total cost, an order total paid, and an order status (open, closed, or cancel).

   Given that information, create the complete ERD containing all primary keys, foreign keys, and main attributes.
4. In Chapter 4, you saw the creation of the Tiny College database design. That design reflected such business rules as "a professor may advise many students" and "a professor may chair one department." Modify the design shown in Figure 4.36 to include these business rules:
   - An employee could be staff or a professor or an administrator.
   - A professor may also be an administrator.
   - Staff employees have a work level classification, such a Level I and Level II.
   - Only professors can chair a department. A department is chaired by only one professor.
   - Only professors can serve as the dean of a college. Each of the university's colleges is served by one dean.
   - A professor can teach many classes.
   - Administrators have a position title.

   Given that information, create the complete ERD containing all primary keys, foreign keys, and main attributes.
5. Tiny College wants to keep track of the history of all administrative appointments (date of appointment and date of termination). (*Hint*: Time variant data are at work.) The Tiny College chancellor may want to know how many deans worked in the College of Business between January 1, 1960 and January 1, 2008 or who the dean of the College of Education was in 1990. Given that information, create the complete ERD containing all primary keys, foreign keys, and main attributes.
6. Some Tiny College staff employees are information technology (IT) personnel. Some IT personnel provide technology support for academic programs. Some IT personnel provide technology infrastructure support. Some IT personnel provide technology support for academic programs and technology infrastructure support. IT personnel are not professors. IT personnel are required to take periodic training to retain their technical expertise. Tiny College tracks all IT personnel training by date, type, and results (completed vs. not completed). Given that information, create the complete ERD containing all primary keys, foreign keys, and main attributes.
7. The FlyRight Aircraft Maintenance (FRAM) division of the FlyRight Company (FRC) performs all maintenance for FRC's aircraft. Produce a data model segment that reflects the following business rules:
   - All mechanics are FRC employees. Not all employees are mechanics.
   - Some mechanics are specialized in engine (EN) maintenance. Some mechanics are specialized in airframe (AF) maintenance. Some mechanics are specialized in avionics (AV) maintenance. (Avionics are the electronic components of an aircraft that are used in communication and navigation.) All mechanics take periodic refresher courses to stay current in their areas of expertise. FRC tracks all courses taken by each mechanic—date, course type, certification (Y/N), and performance.
   - FRC keeps a history of the employment of all mechanics. The history includes the date hired, date promoted, date terminated, and so on. (*Note:* The "and so on" component is, of course, not a real-world requirement. Instead, it has been used here to limit the number of attributes you will show in your design.)

   Given those requirements, create the Crow's Foot ERD segment.

8. You have been asked to create a database design for the BoingX Aircraft Company (BAC), which has two HUD (heads-up display) products: TRX-5A and TRX-5B. The database must enable managers to track blueprints, parts, and software for each HUD, using the following business rules:
   - For simplicity's sake, you may assume that the TRX-5A unit is based on two engineering blueprints and that the TRX-5B unit is based on three engineering blueprints. You are free to make up your own blueprint names.
   - All parts used in the TRX-5A and TRX-5B are classified as hardware. For simplicity's sake, you may assume that the TRX-5A unit uses three parts and that the TRX-5B unit uses four parts. You are free to make up your own part names.

NOTE

Some parts are supplied by vendors, while others are supplied by the BoingX Aircraft Company. Parts suppliers must be able to meet the technical specification requirements (TCRs) set by the BoingX Aircraft Company. Any parts supplier that meets the BoingX Aircraft Company's TCRs may be contracted to supply parts. Therefore, any part may be supplied by multiple suppliers and a supplier can supply many different parts.

   - BAC wants to keep track of all part price changes and the dates of those changes.
   - BAC wants to keep track of all TRX-5A and TRX-5B software. For simplicity's sake, you may assume that the TRX-5A unit uses two named software components and that the TRX-5B unit also uses two named software components. You are free to make up your own software names.
   - BAC wants to keep track of all changes made in blueprints and software. Those changes must reflect the date and time of the change, the description of the change, the person who authorized the change, the person who actually made the change, and the reason for the change.
   - BAC wants to keep track of all HUD test data by test type, test date, and test outcome.

   Given those requirements, create the Crow's Foot ERD.

NOTE

Problem 9 is sufficiently complex to serve as a class project.

9. Global Computer Solutions (GCS) is an information technology consulting company with many offices located throughout the United States. The company's success is based on its ability to maximize its resources—that is, its ability to match highly skilled employees with projects according to region. To better manage its projects, GCS has contacted you to design a database so that GCS managers can keep track of their customers, employees, projects, project schedules, assignments, and invoices.

   The GCS database must support all of GCS's operations and information requirements. A basic description of the main entities follows:
   - The *employees* working for GCS have an employee ID, an employee last name, a middle initial, a first name, a region, and a date of hire.
   - Valid *regions* are as follows: Northwest (NW), Southwest (SW), Midwest North (MN), Midwest South (MS), Northeast (NE), and Southeast (SE).
   - Each employee has many skills, and many employees have the same skill.
   - Each *skill* has a skill ID, description, and rate of pay. Valid skills are as follows: data entry I, data entry II, systems analyst I, systems analyst II, database designer I, database designer II, Cobol I, Cobol II, C++ I, C++ II, VB I, VB II, ColdFusion I, ColdFusion II, ASP I, ASP II, Oracle DBA, MS SQL Server DBA, network engineer I, network engineer II, web administrator, technical writer, and project manager. Table P6.9a shows an example of the Skills Inventory.

TABLE
P6.9a

SKILL	EMPLOYEE
Data Entry I	Seaton Amy; Williams Josh; Underwood Trish
Data Entry II	Williams Josh; Seaton Amy
Systems Analyst I	Craig Brett; Sewell Beth; Robbins Erin; Bush Emily; Zebras Steve
Systems Analyst II	Chandler Joseph; Burklow Shane; Robbins Erin
DB Designer I	Yarbrough Peter; Smith Mary
DB Designer II	Yarbrough Peter; Pascoe Jonathan
Cobol I	Kattan Chris; Ephanor Victor; Summers Anna; Ellis Maria
Cobol II	Kattan Chris; Ephanor Victor, Batts Melissa
C++ I	Smith Jose; Rogers Adam; Cope Leslie
C++ II	Rogers Adam; Bible Hanah
VB I	Zebras Steve; Ellis Maria
VB II	Zebras Steve; Newton Christopher
ColdFusion I	Duarte Miriam; Bush Emily
ColdFusion II	Bush Emily; Newton Christopher
ASP I	Duarte Miriam; Bush Emily
ASP II	Duarte Miriam; Newton Christopher
Oracle DBA	Smith Jose; Pascoe Jonathan
SQL Server DBA	Yarbrough Peter; Smith Jose
Network Engineer I	Bush Emily; Smith Mary
Network Engineer II	Bush Emily; Smith Mary
Web Administrator	Bush Emily; Smith Mary; Newton Christopher
Technical Writer	Kilby Surgena; Bender Larry
Project Manager	Paine Brad; Mudd Roger; Kenyon Tiffany; Connor Sean

- GCS has many *customers*. Each customer has a customer ID, customer name, phone number, and region.
- GCS works by *projects*. A project is based on a contract between the customer and GCS to design, develop, and implement a computerized solution. Each project has specific characteristics such as the project ID, the customer to which the project belongs, a brief description, a project date (that is, the date on which the project's contract was signed), a project start date (an estimate), a project end date (also an estimate), a project budget (total estimated cost of project), an actual start date, an actual end date, an actual cost, and one employee assigned as manager of the project.
- The actual cost of the project is updated each Friday by adding that week's cost (computed by multiplying the hours each employee worked by the rate of pay for that skill) to the actual cost.
- The employee who is the manager of the project must complete a *project schedule*, which is, in effect, a design and development plan. In the project schedule (or plan), the manager must determine the tasks that will be performed to take the project from beginning to end. Each task has a task ID, a brief task description, the task's starting and ending date, the type of skill needed, and the number of employees (with the required skills) required to complete the task. General tasks are initial interview, database and system design, implementation, coding, testing, and final evaluation and sign-off. For example, GCS might have the project schedule shown in Table P6.9b.

TABLE
P6.9b

PROJECT ID: 1		DESCRIPTION: SALES MANAGEMENT SYSTEM		
COMPANY : SEE ROCKS		CONTRACT DATE: 2/12/2008	REGION: NW	
START DATE: 3/1/2008		END DATE: 7/1/2008	BUDGET: $15,500	
START DATE	END DATE	TASK DESCRIPTION	SKILL(S) REQUIRED	QUANTITY REQUIRED
3/1/08	3/6/08	Initial Interview	Project Manager Systems Analyst II DB Designer I	1 1 1
3/11/08	3/15/08	Database Design	DB Designer I	1
3/11/08	4/12/08	System Design	Systems Analyst II Systems Analyst I	1 2
3/18/08	3/22/08	Database Implementation	Oracle DBA	1
3/25/08	5/20/08	System Coding & Testing	Cobol I Cobol II Oracle DBA	2 1 1
3/25/08	6/7/08	System Documentation	Technical Writer	1
6/10/08	6/14/08	Final Evaluation	Project Manager Systems Analyst II DB Designer I Cobol II	1 1 1 1
6/17/08	6/21/08	On-Site System Online and Data Loading	Project Manager Systems Analyst II DB Designer I Cobol II	1 1 1 1
7/1/08	7/1/08	Sign-Off	Project Manager	1

- Assignments: GCS pools all of its employees by region, and from this pool, employees are assigned to a specific task scheduled by the project manager. For example, for the first project's schedule, you know that for the period 3/1/08 to 3/6/08, a Systems Analyst II, a Database Designer I, and a Project Manager are needed. (The project manager is assigned when the project is created and remains for the duration of the project). Using that information, GCS searches the employees who are located in the same region as the customer, matching the skills required and assigning them to the project task.
- Each project schedule task can have many employees assigned to it, and a given employee can work on multiple project tasks. However, an employee can work on only one project task at a time. For example, if an employee is already assigned to work on a project task from 2/20/08 to 3/3/08, (s)he cannot work on another task until the current assignment is closed (ends). The date on which an assignment is closed does not necessarily match the ending date of the project schedule task, because a task can be completed ahead of or behind schedule.
- Given all of the preceding information, you can see that the assignment associates an employee with a project task, using the project schedule. Therefore, to keep track of the *assignment*, you require at least the following information: assignment ID, employee, project schedule task, date assignment starts, and date assignment ends (which could be any date, as some projects run ahead of or behind schedule). Table P6.9c shows a sample assignment form.

TABLE P6.9c

PROJECT ID: 1 COMPANY: SEEROCKS			DESCRIPTION: SALES MANAGEMENT SYSTEM CONTRACT DATE: 2/12/2008		AS OF: 03/29/08	
SCHEDULED				ACTUAL ASSIGNMENTS		
PROJECT TASK	START DATE	END DATE	SKILL	EMPLOYEE	START DATE	END DATE
Initial Interview	3/1/08	3/6/08	Project Mgr. Sys. Analyst II DB Designer I	101—Connor S. 102—Burklow S. 103—Smith M.	3/1/08 3/1/08 3/1/08	3/6/08 3/6/08 3/6/08
Database Design	3/11/08	3/15/08	DB Designer I	104—Smith M.	3/11/08	3/14/08
System Design	3/11/08	4/12/08	Sys. Analyst II Sys. Analyst I Sys. Analyst I	105—Burklow S. 106—Bush E. 107—Zebras S.	3/11/08 3/11/08 3/11/08	
Database Implementation	3/18/08	3/22/08	Oracle DBA	108—Smith J.	3/15/08	3/19/08
System Coding & Testing	3/25/08	5/20/08	Cobol I Cobol I Cobol II Oracle DBA	109—Summers A. 110—Ellis M. 111—Ephanor V. 112—Smith J.	3/21/08 3/21/08 3/21/08 3/21/08	
System Documentation	3/25/08	6/7/08	Tech. Writer	113—Kilby S.	3/25/08	
Final Evaluation	6/10/08	6/14/08	Project Mgr. Sys. Analyst II DB Designer I Cobol II			
On-Site System Online and Data Loading	6/17/08	6/21/08	Project Mgr. Sys. Analyst II DB Designer I Cobol II			
Sign-Off	7/1/08	7/1/08	Project Mgr.			

(*Note:* The assignment number is shown as a prefix of the employee name; for example, 101, 102.) Assume that the assignments shown previously are the only ones existing as of the date of this design. The assignment number can be whatever number matches your database design.

- The hours an employee works are kept in a *work log* containing a record of the actual hours worked by an employee on a given assignment. The work log is a weekly form that the employee fills out at the end of each week (Friday) or at the end of each month. The form contains the date (of each Friday of the month or the last work day of the month if it doesn't fall on a Friday), the assignment ID, the total hours worked that week (or up to the end of the month), and the number of the bill to which the work log entry is charged. Obviously, each work log entry can be related to only one bill. A sample list of the current work log entries for the first sample project is shown in Table P6.9d.

TABLE
P6.9d

EMPLOYEE NAME	WEEK ENDING	ASSIGNMENT NUMBER	HOURS WORKED	BILL NUMBER
Burklow S.	3/1/08	1-102	4	xxx
Connor S.	3/1/08	1-101	4	xxx
Smith M.	3/1/08	1-103	4	xxx
Burklow S.	3/8/08	1-102	24	xxx
Connor S.	3/8/08	1-101	24	xxx
Smith M.	3/8/08	1-103	24	xxx
Burklow S.	3/15/08	1-105	40	xxx
Bush E.	3/15/08	1-106	40	xxx
Smith J.	3/15/08	1-108	6	xxx
Smith M.	3/15/08	1-104	32	xxx
Zebras S.	3/15/08	1-107	35	xxx
Burklow S.	3/22/08	1-105	40	
Bush E.	3/22/08	1-106	40	
Ellis M.	3/22/08	1-110	12	
Ephanor V.	3/22/08	1-111	12	
Smith J.	3/22/08	1-108	12	
Smith J.	3/22/08	1-112	12	
Summers A.	3/22/08	1-109	12	
Zebras S.	3/22/08	1-107	35	
Burklow S.	3/29/08	1-105	40	
Bush E.	3/29/08	1-106	40	
Ellis M.	3/29/08	1-110	35	
Ephanor V.	3/29/08	1-111	35	
Kilby S.	3/29/08	1-113	40	
Smith J.	3/29/08	1-112	35	
Summers A.	3/29/08	1-109	35	
Zebras S.	3/29/08	1-107	35	

Note: xxx represents the bill ID. Use the one that matches the bill number in your database.

- Finally, every 15 days, a *bill* is written and sent to the customer, totaling the hours worked on the project that period. When GCS generates a bill, it uses the bill number to update the work-log entries that are part of that bill. In summary, a bill can refer to many work log entries, and each work log entry can be related to only one bill. GCS sent one bill on 3/15/08 for the first project (Xerox), totaling the hours worked between 3/1/08 and 3/15/08. Therefore, you can safely assume that there is only one bill in this table and that that bill covers the work-log entries shown in the above form.

C6545_06 8/20/2007 10:57:9 Page 221

Your assignment is to create a database that will fulfill the operations described in this problem. The minimum required entities are employee, skill, customer, region, project, project schedule, assignment, work log, and bill. (There are additional required entities that are not listed.)

- Create all of the required tables and all of the required relationships.
- Create the required indexes to maintain entity integrity when using surrogate primary keys.
- Populate the tables as needed (as indicated in the sample data and forms).