Entity relationship (er) modeling

Contents

- E N T I T Y R E L A T I O N S H I P (E R) M O D E L I N G FIGURE

In this chapter, you will learn: The main characteristics of entity relationship components ? How relationships between entities are defined, refined, and incorporated into the database design process ? How ERD components affect database design and implementation ? That real-world database design often requires the reconciliation of conflicting goals. This chapter expands coverage of the data-modeling aspect of database design. Data modeling is the first step in the database design journey, serving as a bridge between P real-world objects and the database model that is implemented in the computer. Therefore, the importance of data-modeling details, expressed graphically through entity relationship diagrams (ERDs), cannot be overstated.

Most of the basic concepts and definitions used in the entity relationship model (ERM) were introduced in Chapter 2, Data Models. For example, the basic components of entities and relationships and their representation should now be familiar to you. This chapter goes much deeper and further, analyzing the graphic depiction of relationships among the entities and showing how those depictions help you summarize the wealth of data required to implement a successful design. Finally, the chapter illustrates how conflicting goals can be a challenge in database design, possibly requiring you to make design compromises.

FOUR CHAPTER

Note Because this book generally focuses on the relational model, you might be tempted to conclude that the ERM is exclusively a relational tool. Actually, conceptual models such as the ERM can be used to understand and design the data requirements of an organization.

Therefore, the ERM is independent of the database type. Conceptual models are used in the conceptual design of databases, while relational models are used in the logical design of databases. However, because you are now familiar with the relational model from the previous chapter, the relational model is used extensively in this chapter to explain ER constructs and the way they are used to develop database designs.

4. 1 THE ENTITY RELATIONSHIP MODEL (ERM)

You should remember from Chapter 2, Data Models, and Chapter 3, The Relational Database Model, that the ERM forms the basis of an ERD.

The ERD represents the conceptual database as viewed by the end user. ERDs depict the database’s main components: entities, attributes, and relationships. Because an entity represents a real-world object, the words entity and object are often used interchangeably. Thus, the entities (objects) of the Tiny College database design developed in this chapter include students, classes, teachers, and classrooms. The order in which the ERD components are covered in the chapter is dictated by the way the modeling tools are used to develop ERDs that can form the basis for successful database design and implementation.

In Chapter 2, you also learned about the various notations used with ERDs—the original Chen notation and the newer Crow's Foot and UML notations. The first two notations are used at the beginning of this chapter to introduce some basic ER modeling concepts. Some conceptual database modeling concepts can be expressed only using the Chen notation. However, because the emphasis is on design and implementation of databases, the Crow's Foot and UML class diagram notations are used for the final Tiny College ER diagram example. Because of its implementation emphasis, the Crow's Foot notation can represent only what could be implemented.

In other words: ? ? ? The Chen notation favors conceptual modeling. The Crow's Foot notation favors a more implementation-oriented approach. The UML notation can be used for both conceptual and implementation modeling. Online Content To learn how to create ER diagrams with the help of Microsoft Visio, see the Premium Website for this book: • Appendix A, Designing Databases with Visio Professional: A Tutorial shows you how to create Crow's Foot ERDs.

• Appendix H, Unified Modeling Language (UML), shows you how to create UML class diagrams.

4. 1. 1 Entities Recall

That an entity is an object of interest to the end user. In Chapter 2, you learned that at the ER modeling level, an entity actually refers to the entity set and not to a single entity occurrence.

In other words, the word entity in the ERM corresponds to a table—not to a row—in the relational environment. The ERM refers to a table row as an entity instance or entity occurrence. In both the Chen and Crow’s Foot notations, an entity is represented by a rectangle containing the entity’s name. The entity name, a noun, is usually written in all capital letters.

E N T I T Y R E L A T I O N S H I P (E R) M O D E L I N G

4. 1.

2 Attributes

Attributes are characteristics of entities. For example, the STUDENT entity includes, among many others, the attributes STU_LNAME, STU_FNAME, and STU_INITIAL. In the original Chen notation, attributes are represented by ovals and are connected to the entity rectangle with a line. Each oval contains the name of the attribute it represents. In the Crow’s Foot notation, the attributes are written in the attribute box below the entity rectangle.

(See Figure 4. 1.) Because the Chen representation is rather space-consuming, software vendors have adopted the Crow’s Foot attribute display. FIGURE The attributes of the STUDENT entity: Chen and Crow’s Foot 4.

1 Chen Model STU_INITIAL STU_FNAME STU_EMAIL Crow’s Foot Model STU_LNAMESTUDENT STU_PHONE Required and Optional Attributes A required attribute is an attribute that must have a value; in other words, it cannot be left empty. As shown in Figure 4. 1, there are two boldfaced attributes in the Crow’s Foot notation. This indicates that a data entry will be required. In this example, STU_LNAME and STU_FNAME require data entries because of the assumption that all students have a last name and a first

name. But students might not have a middle name, and perhaps they do not (yet) have a phone number and an e-mail address.

Therefore, those ttributes are not presented in boldface in the entity box. An optional attribute is an attribute that does not require a value; therefore, it can be left empty. Domains Attributes have a domain. As you learned in Chapter 3, a domain is the set of possible values for a given attribute. For example, the domain for the grade point average (GPA) attribute is written (0, 4) because the lowest possible GPA value is 0 and the highest possible value is 4.

The domain for the gender attribute consists of only two possibilities: M or F (or some other equivalent code). The domain for a company's date of hire attribute consists of all dates that fit in a range (for example, company startup date to current date). Attributes may share a domain. For instance, a student address and a professor address share the same domain of all possible addresses. In fact, the data dictionary may let a newly declared attribute inherit the characteristics of an existing attribute if the same attribute name is used. For example, the PROFESSOR and STUDENT entities may each have an attribute named ADDRESS and could therefore share a domain.

Identifiers (Primary Keys)The ERM uses identifiers, that is, one or more attributes that uniquely identify each entity instance. In the relational model, such identifiers are mapped to primary keys (PKs) in tables. Identifiers are underlined in the ERD. Key attributes are also underlined in a frequently used table structure shorthand notation using the format: TABLE NAME (KEY_ATTRIBUTE 1, ATTRIBUTE 2, ATTRIBUTE 3, .

. . ATTRIBUTE K) 102 C H A P T E R 4 For example, a CAR entity may be represented by: CAR (CAR_VIN, MOD_CODE, CAR_YEAR, CAR_COLOR) (Each car is identified by a unique vehicle dentification number, or CAR_VIN.) Composite Identifiers Ideally, an entity identifier is composed of only a single attribute. For example, the table in Figure 4. 2 uses a single-attribute primary key named CLASS_CODE.

However, it is possible to use a composite identifier, that is, a primary key composed of more than one attribute. For instance, the Tiny College database administrator may decide to identify each CLASS entity instance (occurrence) by using a composite primary key composed of the combination of CRS_CODE and CLASS_SECTION instead of using CLASS_CODE. Either approach uniquely identifies each entity instance. Given the current structure of the CLASS table shown in Figure 4.

2, CLASS_CODE is the primary key, and the combination of CRS_CODE and CLASS_SECTION is a proper candidate key. If the CLASS_CODE attribute is deleted from the CLASS entity, the candidate key (CRS_CODE and CLASS_SECTION) becomes an acceptable composite primary key. FIGURE The CLASS table (entity) components and contents 4. 2 Note Remember that Chapter 3 made a commonly accepted distinction between COURSE and CLASS.

A CLASS constitutes a specific time and place of a COURSE offering. A class is defined by the course description and its time and place, or section.

Consider a professor who teaches Database I, Section 2; Database I, Section 5; Database I, Section 8; and Spreadsheet II, Section 6. That instructor teaches two courses (Database I and Spreadsheet II), but four classes. Typically, the COURSE offerings are printed in a course catalog, while the CLASS offerings are printed in a class schedule for each semester, trimester, or quarter. If the CLASS_CODE in Figure 4.

2 is used as the primary key, the CLASS entity ay be represented in shorthand form by: CLASS (CLASS_CODE, CRS_CODE, CLASS_SECTION, CLASS_TIME, ROOM_CODE, PROF_NUM).

E N T I T Y R E L A T I O N S H I P (E R) M O D E L I N G

On the other hand, if CLASS_CODE is deleted, and the composite primary key is the combination of CRS_CODE and CLASS_SECTION, the CLASS entity may be represented by: CLASS (CRS_CODE, CLASS_SECTION, CLASS_TIME, ROOM_CODE, PROF_NUM) Note that both key attributes are underlined in the entity notation. Composite and Simple Attributes Attributes are classified as simple or composite. A composite attribute, not to be confused with a composite key, is an attribute that can be further subdivided to yield additional attributes. For example, the attribute ADDRESS can be subdivided into street, city, state, and zip code. Similarly, the attribute PHONE_NUMBER can be subdivided into area code and exchange number.

A simple attribute is an attribute that cannot be subdivided. For example, age, sex, and marital status would be classified as simple attributes. To facilitate detailed queries, it is wise to change composite attributes into a series of simple attributes. Single-Valued Attributes A single-valued attribute is an attribute that can have only a single value.

For example, a person can have only one Social Security number, and a manufactured part can have only one serial number. Keep in mind that a single-valued attribute is not necessarily a simple attribute. For instance, a part’s serial number, such as SE-08-02-189935, is single-valued, but it is a composite attribute because it can be subdivided into the region in which the part was produced (SE), the plant within that region (08), the shift within the plant (02), and the part number (189935). Multivalued Attributes Multivalued attributes are attributes that can have many values. For instance, a person may have several college degrees, and a household may have several different phones, each with its own number.

Similarly, a car’s color may be subdivided into many colors (that is, colors for the roof, body, and trim). In the Chen ERM, the multivalued attributes are shown by a double line connecting the attribute to the entity. The Crow’s Foot notation does not identify multivalued attributes. The ERD in Figure 4.

contains all of the components introduced thus far. In Figure 4. 3, note that CAR_VIN is the primary key, and CAR_COLOR is a multivalued attribute of the CAR entity. FIGURE A multivalued attribute in an entity 4. 3 Chen Model Crow’s Foot Model MOD_CODE CAR_YEAR CAR_VIN CAR CAR_COLOR 104 C H A P T E R 4 Note In the ERD models in Figure 4. 3, the CAR entity’s foreign key (FK) has been typed as MOD_CODE.

This attribute was manually added to the entity. Actually, proper use of database modeling software will automatically produce the FK when the relationship is defined. In addition, the software will label the FK appropriately and write the FK’s implementation details in a data dictionary. Therefore, when you use database modeling software like Visio Professional, never type the FK attribute yourself; let the software handle that task when the relationship between the entities is defined. (You can see how that’s done in Appendix A, Designing Databases with Visio Professional: A Tutorial, in the Premium Website.) Implementing Multivalued Attributes Although the conceptual model can handle M: N relationships and multivalued attributes, you should not implement them in the RDBMS.

Remember from Chapter 3 that in the relational table, each column/row intersection represents a single data value. So if multivalued attributes exist, the designer must decide on one of two possible courses of action: 1. Within the original entity, create several new attributes, one for each of the original multivalued attribute’s components. For example, the CAR entity’s attribute CAR_COLOR can be split to create the new attributes CAR_TOPCOLOR, CAR_BODYCOLOR, and CAR_TRIMCOLOR, which are then assigned to the CAR entity.

(See Figure 4. 4.) FIGURE Splitting the multivalued attribute into new attributes . 4 Chen Model CAR_YEAR MOD_CODE CAR_TOPCOLOR Crow’s Foot Model CAR_VIN CAR CAR_TRIMCOLOR CAR_BODYCOLOR Although this solution seems to work, its adoption can lead to major structural problems in the table.

For example, if additional color components—such as a logo color—are added for some cars, the table structure must be modified to accommodate the new color section. In that case, cars that do not have such color sections generate nulls for the nonexisting components, or their color entries for those sections are entered as N/A to indicate “ not applicable. (Imagine how the solution in Figure 4. 4—splitting a multivalued attribute into new attributes—would cause problems if it were applied to an employee entity containing employee degrees and certifications. If some employees have 10 degrees and certifications while most have fewer or none, the number of degree/certification attributes would number 10, and most of those attribute values would be null for most of the employees.) In short, although you have seen solution 1 applied, it is not an acceptable solution.

2. Create a new entity composed of the original multivalued attribute’s components. This new entity allows the designer to define color for different sections of the car. (See Table 4.

1.) Then, this new CAR_COLOR entity is related to the original CAR entity in a 1: M relationship.

E N T I T Y R E L A T I O N S H I P (E R) M O D E L I N G

TABLE 4. 1 SECTION Top Body Trim Interior Components of the Multivalued Attribute COLOR White Blue Gold Blue Using the approach illustrated in Table 4.

1, you even get a fringe benefit: you are now able to assign as many colors as necessary without having to change the table structure. Note that the ERM shown in Figure 4. reflects the components listed in Table 4. 1.

This is the preferred way to deal with multivalued attributes. Creating a new entity in a 1: M relationship with the original entity yields several benefits: it's a more flexible, expandable solution, and it is compatible with the relational model! FIGURE A new entity set composed of a multivalued attribute's components 4. 5 Derived Attributes Finally, an attribute may be classified as a derived attribute. A derived attribute is an attribute whose value is calculated (derived) from other attributes. The derived attribute need not be physically stored within the database; instead, it can be derived by using an algorithm. For example, an employee's age, EMP_AGE, may be found by computing the integer value of the difference between the current date and the EMP_DOB.

If you use Microsoft Access, you would use the formula INT((DATE() – EMP_DOB)/365). In Microsoft SQL Server, you would use SELECT DATEDIFF(" YEAR", EMP_DOB, GETDATE()), where DATEDIFF is a function that computes the difference between dates. The first parameter indicates the measurement, in this case, years. If you use Oracle, you would use SYSDATE instead of DATE().

You are assuming, of course, that the EMP_DOB was stored in the Julian date format.) Similarly, the total cost of an order can be derived by multiplying the quantity ordered by the unit price. Or the estimated average speed can be derived by dividing trip distance by the time spent en route. A derived attribute is indicated in the Chen notation by a dashed line connecting the attribute and the entity.

(See Figure 4. 6.) The Crow's Foot notation does not have a method for distinguishing the derived attribute from other attributes. Derived attributes are sometimes referred to as computed attributes. A derived attribute computation can be as simple as adding two attribute values located on the same row, or it can be the result of aggregating the sum of values located on many table rows (from the same table or from a different table).

The decision to store derived attributes in database tables depends on the processing requirements and the constraints placed on a particular application. The designer should be able to balance the design in accordance with such constraints. Table 4. 2 shows the advantages and disadvantages of storing (or not storing) derived attributes in the database.

. 1. 3 Relationships Recall from Chapter 2 that a relationship is an association between entities. The entities that participate in a relationship are also known as participants, and each relationship is identified by a name that describes the relationship.

The relationship name is an active or passive verb; for example, a STUDENT takes a CLASS, a PROFESSOR teaches a CLASS, a DEPARTMENT employs a PROFESSOR, a DIVISION is managed by an EMPLOYEE, and an AIRCRAFT is flown by a CREW. 106 C H A P T E R 4 FIGURE Depiction of a derived attribute 4. 6 Chen ModelEMP_FNAME EMP_LNAME EMP_INITIAL EMP_DOB Crow's Foot Model EMP_NUM EMPLOYEE EMP_AGE TABLE 4. 2 Advantages and Disadvantages of Storing Derived Attributes DERIVED ATTRIBUTE STORED NOT STORED Saves storage space Saves CPU processing cycles Computation always yields current value Saves data access time Data value is readily available Can be used to keep track of historical data Uses CPU processing cycles Requires constant maintenance to ensure derived value is current, especially if any values Increases data access time Adds coding complexity to queries used in the calculation changeAdvantage Disadvantage Relationships between entities always operate in both directions. That is, to define the relationship between the entities named CUSTOMER and INVOICE, you would specify that: ? ? A CUSTOMER may generate many INVOICEs. Each INVOICE is generated by one CUSTOMER.

Because you know both directions of the relationship between CUSTOMER and INVOICE, it is easy to see that this relationship can be classified as 1: M. The relationship classification is difficult to establish if you know only one side of the relationship. For example, if you specify that: A DIVISION is managed by one EMPLOYEE. You don't know if the relationship is 1: 1 or 1: M. Therefore, you should ask the question " Can an employee manage more than one division? " If the answer is yes, the relationship is 1: M, and the second part of the relationship is then written as: An EMPLOYEE may manage many DIVISIONs. If an employee cannot manage more than one division, the relationship is 1: 1, and the second part of the relationship is then written as: An EMPLOYEE may manage only one DIVISION.

E N T I T Y R E L A T I O N S H I P (E R) M O D E L I N G

4. 1. 4 Connectivity and Cardinality

You learned in Chapter 2 that entity relationships may be classified as one-to-one, one-to-many, or many-to-many. You also learned how such relationships were depicted in the Chen and Crow's Foot notations.

The term connectivity is used to describe the relationship classification. Cardinality expresses the minimum and maximum number of entity occurrences associated with one occurrence of the related entity. In the ERD, cardinality is indicated by placing the appropriate numbers beside the entities, using the format (x, y). The first value represents the minimum number of associated entities, while the second value represents the maximum number of associated entities.

Many database designers who use Crow’s Foot modeling notation do not depict the specific cardinalities on the ER diagram itself because the specific limits described by the cardinalities cannot be implemented directly through the database design. Correspondingly, some Crow’s Foot ER modeling tools do not print the numeric cardinality range in the diagram; instead, you can add it as text if you want to have it shown. When the specific cardinalities are not included on the diagram in Crow’s Foot notation, cardinality is implied by the use of the symbols shown in Figure 4. 7, which describe the connectivity and participation (discussed below). The numeric cardinality range has been added using the Visio text drawing tool. FIGURE 4.

7 Connectivity and cardinality in an ERD Knowing the minimum and maximum number of entity occurrences is very useful at the application software level. For example, Tiny College might want to ensure that a class is not taught unless it has at least 10 students enrolled. Similarly, if the classroom can hold only 30 students, the application software should use that cardinality to limit enrollment in the class. However, keep in mind that the DBMS cannot handle the implementation of the cardinalities at the table level—that capability is provided by the application software or by triggers.

You will learn how to create and execute triggers in Chapter 8, Advanced SQL.

As you examine the Crow’s Foot diagram in Figure 4. 7, keep in mind that the cardinalities represent the number of occurrences in the related entity. For example, the cardinality (1, 4) written next to the CLASS entity in the “ PROFESSOR teaches CLASS” relationship indicates that each professor teaches up to four classes, which means that the PROFESSOR table’s primary key value occurs at least once and no more than four times as foreign key values in the CLASS table. If the cardinality had been written as (1, N), there would be no upper limit to the number of classes a professor might teach. Similarly, the cardinality (1, 1) written next to the PROFESSOR entity indicates that each class is taught by one and only one professor. That is, each CLASS entity occurrence is associated with one and only one entity occurrence in PROFESSOR.

Connectivities and cardinalities are established by very concise statements known as business rules, which were introduced in Chapter 2. Such rules, derived from a precise and detailed description of an organization’s data environment, also establish the ERM’s entities, attributes, relationships, connectivities, cardinalities, and constraints. Because business rules define the ERM’s components, making sure that all appropriate business rules are identified is a very important part of a database designer’s job. Note The placement of the cardinalities in the ER diagram is a matter of convention. The Chen notation places the cardinalities on the side of the related entity.

The Crow’s Foot and UML diagrams place the cardinalities next to the entity to which the cardinalities apply. 108 C H A P T E R 4 Online Content Because the careful definition of complete and accurate business rules is crucial to good database design, their derivation is examined in detail in Appendix B, The University Lab: Conceptual Design. The modeling skills you are learning in this chapter are applied in the development of a real database design in Appendix B. The initial design shown in Appendix B is then modified in Appendix C, The University Lab: Conceptual Design Verification, Logical Design, and Implementation. (Both appendixes are found in the Premium Website.

)

4. 1. 5 Existence Dependence

An entity is said to be existence-dependent if it can exist in the database only when it is associated with another related entity occurrence. In implementation terms, an entity is existence-dependent if it has a mandatory foreign key—that is, a foreign key attribute that cannot be null.

For example, if an employee wants to claim one or more dependents for tax-withholding purposes, the relationship “ EMPLOYEE claims DEPENDENT” would be appropriate. In that case, the DEPENDENT entity is clearly existence-dependent on the EMPLOYEE entity because it is impossible for the dependent to exist apart from the EMPLOYEE in the database. If an entity can exist apart from all of its related entities (it is existence-independent), then that entity is referred to as a strong entity or regular entity. For example, suppose that the XYZ Corporation uses parts to produce its products. Furthermore, suppose that some of those parts are produced in-house and other parts are bought from vendors. In that scenario, it is quite possible for a PART to exist independently from a VENDOR in the relationship " PART is supplied by VENDOR," because at least some of the parts are not supplied by a vendor.

Therefore, PART is existence-independent from VENDOR. Note The relationship strength concept is not part of the original ERM. Instead, this concept applies directly to Crow's Foot diagrams. Because Crow's Foot diagrams are used extensively to design relational databases, it is important to understand relationship strength as it affects database implementation. The Chen ERD notation is oriented toward conceptual modeling and therefore does not distinguish between weak and strong relationships.

4. 1. 6 Relationship Strength

The concept of relationship strength is based on how the primary key of a related entity is defined. To implement a relationship, the primary key of one entity appears as a foreign key in the related entity.

For example, the 1: M relationship between VENDOR and PRODUCT in Chapter 3, Figure 3. 3, is implemented by using the VEND_CODE primary key in VENDOR as a foreign key in PRODUCT. There are times when the foreign key also is a primary key component in the related entity. For example, in Figure 4.

5, the CAR entity rimary key (CAR_VIN) appears as both a primary key component and a foreign key in the CAR_COLOR entity. In this section, you will learn how various relationship strength decisions affect primary key arrangement in database design.

E N T I T Y R E L A T I O N S H I P (E R) M O D E L I N G

Weak (Non-identifying) Relationships A weak relationship, also known as a non-identifying relationship, exists if the PK of the related entity does not contain a PK component of the parent entity. By default, relationships are established by having the PK of the parent entity appear as an FK on the related entity. For example, suppose that the COURSE and CLASS entities are defined as: COURSE(CRS_CODE, DEPT_CODE, CRS_DESCRIPTION, CRS_CREDIT) CLASS(CLASS_CODE, CRS_CODE, CLASS_SECTION, CLASS_TIME, ROOM_CODE, PROF_NUM) In this case, a weak relationship exists between COURSE and CLASS because the CLASS_CODE is the CLASS entity’s PK, while the CRS_CODE in CLASS is only an FK.

In this example, the CLASS PK did not inherit the PK component from the COURSE entity. Figure 4. 8 shows how the Crow’s Foot notation depicts a weak relationship by placing a dashed relationship line between the entities. The tables shown below the ERD illustrate how such a relationship is implemented.

FIGURE A weak (non-identifying) relationship between COURSE and CLASS 4. 8 Table name: COURSE Database name: Ch04_TinyCollege Table name: CLASS 110 C H A P T E R 4 Online Content All of the databases used to illustrate the material in this chapter are found in the Premium Website. Note If you are used to looking at relational diagrams such as the ones produced by Microsoft Access, you expect to see the relationship line in the relational diagram drawn from the PK to the FK. However, the relational diagram convention is not necessarily reflected in the ERD.

In an ERD, the focus is on the entities and the relationships between them, rather than on the way those relationships are anchored graphically. You will discover that the placement of the relationship lines in a complex ERD that includes both horizontally and vertically placed entities is largely dictated by the designer's decision to improve the readability of the design. (Remember that the ERD is used for communication between the designer(s) and end users.) Strong (Identifying) RelationshipsA strong relationship, also known as an identifying relationship, exists when the PK of the related entity contains a PK component of the parent entity. For example, the definitions of the COURSE and CLASS entities COURSE(CRS_CODE, DEPT_CODE, CRS_DESCRIPTION, CRS_CREDIT) CLASS(CRS_CODE, CLASS_SECTION , CLASS_TIME, ROOM_CODE, PROF_NUM) indicate that a strong relationship exists between COURSE and CLASS, because the CLASS entity's composite PK is composed of CRS_CODE + CLASS_SECTION. (Note that the CRS_CODE in CLASS is also the FK to the COURSE entity.

The Crow's Foot notation depicts the strong (identifying) relationship with a solid line between the entities, shown in Figure 4. 9. Whether the relationship between COURSE and CLASS is strong or weak depends on how the CLASS entity's primary key is defined. Keep in mind that the order in which the tables are created and loaded is very important. For example, in the " COURSE generates CLASS" relationship, the COURSE table must be created before the CLASS table. After all, it would not be acceptable to have the CLASS table's foreign key reference a COURSE table that did not yet exist.

In fact, you must load the data of the " 1" side first in a 1: M relationship to avoid the possibility of referential integrity errors, regardless of whether the relationships are weak or strong. As you examine Figure 4. 9 you might wonder what the O symbol next to the CLASS entity signifies. You will discover the meaning of this cardinality in Section 4.

1. 8, Relationship Participation. Remember that the nature of the relationship is often determined by the database designer, who must use professional judgment to determine which relationship type and strength best suit the database transaction, efficiency, and information requirements. That point will often be emphasized in detail!

4. 1. 7 Weak Entities

In contrast to the strong or regular entity mentioned in Section 4.

1. 5, a weak entity is one that meets two conditions: 1. 2. The entity is existence-dependent; that is, it cannot exist without the entity with which it has a relationship.

The entity has a primary key that is partially or totally derived from the parent entity in the relationship.

E N T I T Y R E L A T I O N S H I P (E R) M O D E L I N G FIGURE

A strong (identifying) relationship between COURSE and CLASS 4. 9 Table name: COURSE Database name: Ch04_TinyCollege_Alt Table name: CLASSFor example, a company insurance policy insures an employee and his/her dependents. For the purpose of describing an insurance policy, an

EMPLOYEE might or might not have a DEPENDENT, but the DEPENDENT must be associated with an EMPLOYEE.

Moreover, the DEPENDENT cannot exist without the EMPLOYEE; that is, a person cannot get insurance coverage as a dependent unless s(he) happens to be a dependent of an employee. DEPENDENT is the weak entity in the relationship “ EMPLOYEE has DEPENDENT. ” This relationship is shown in Figure 4. 10. Note that the Chen notation in Figure 4. 10 identifies the weak entity by using a double-walled entity rectangle.

The Crow’s Foot notation generated by Visio Professional uses the relationship line and the PK/FK designation to indicate whether the related entity is weak. A strong (identifying) relationship indicates that the related entity is weak. Such a relationship means that both conditions for the weak entity definition have been met—the related entity is existence-dependent, and the PK of the related entity contains a PK component of the parent entity. (Some versions of the Crow’s Foot ERD depict the weak entity by drawing a short line segment in each of the four corners of the weak entity box. Remember that the weak entity inherits part of its primary key from its strong counterpart. For example, at least part of the DEPENDENT entity’s key shown in Figure 4.

10 was inherited from the EMPLOYEE entity: EMPLOYEE (EMP_NUM, EMP_LNAME, EMP_FNAME, EMP_INITIAL, EMP_DOB, EMP_HIREDATE) DEPENDENT (EMP_NUM, DEP_NUM, DEP_FNAME, DEP_DOB) 112 C H A P T E R 4 FIGURE A weak entity in an ERD 4. 10 Chen Model 1 EMPLOYEE has M DEPENDENT (0, N) EMP_NUM EMP_LNAME EMP_FNAME EMP_INITIAL EMP_DOB

EMP_HIREDATE (1, 1) EMP_NUM DEP_NUM DEP_FNAME DEP_DOB Crow's Foot Model Figure 4. 1 illustrates the implementation of the relationship between the weak entity (DEPENDENT) and its parent or strong counterpart (EMPLOYEE). Note that DEPENDENT's primary key is composed of two attributes, EMP_NUM and DEP_NUM, and that EMP_NUM was inherited from EMPLOYEE.

FIGURE A weak entity in a strong relationship 4. 11 Table name: EMPLOYEE Database name: Ch04_ShortCo Table name: DEPENDENT

ENTITYRELATIONSHIP(ER)MODELING

Given this scenario, and with the help of this relationship, you can determine that: Jeanine J. Callifante claims two dependents, Annelise and Jorge. Keep in mind that the database designer usually determines whether an entity can be described as weak based on the business rules. An examination of the relationship between COURSE and CLASS in Figure 4. 8 might cause you to conclude that CLASS is a weak entity to COURSE.

After all, in Figure 4. 8, it seems clear that a CLASS cannot exist without a COURSE; so there is existence dependence. For example, a student cannot enroll in the Accounting I class ACCT-211, Section 3 (CLASS_CODE 10014) unless there is an ACCT-211 course. However, note that the CLASS table's primary key is CLASS_CODE, which is not derived from the COURSE parent entity. That is, CLASS may be represented by: CLASS (CLASS_CODE, CRS_CODE, CLASS_SECTION, CLASS_TIME, ROOM_CODE, PROF_NUM) The second weak entity requirement has not been met; therefore, by definition, the CLASS entity in Figure 4. 8 may not be classified as weak.

On the other hand, if the CLASS entity's primary key had been defined as a composite key, composed of the combination CRS_CODE and CLASS_SECTION, CLASS could be represented by: CLASS (CRS_CODE, CLASS_SECTION, CLASS_TIME, ROOM_CODE, PROF_NUM) In that case, illustrated in Figure 4. , the CLASS primary key is partially derived from COURSE because CRS_CODE is the COURSE table's primary key. Given this decision, CLASS is a weak entity by definition. (In Visio Professional Crow's Foot terms, the relationship between COURSE and CLASS is classified as strong, or identifying.

) In any case, CLASS is always existence-dependent on COURSE, whether or not it is defined as weak.

4. 1. 8 Relationship Participation

Participation in an entity relationship is either optional or mandatory.

Recall that relationships are bidirectional; that is, they operate in both directions. If COURSE is related to CLASS, then by definition, CLASS is related to COURSE. Because of the bidirectional nature of relationships, it is necessary to determine the connectivity of the relationship from COURSE to CLASS and the connectivity of the relationship from CLASS to COURSE. Similarly, the specific maximum and minimum cardinalities must be determined in each direction for the relationship. Once again, you must consider the bidirectional nature of the relationship when determining participation.

Optional participation means that one entity occurrence does not require a corresponding entity occurrence in a particular relationship. For example, in the “ COURSE generates CLASS” relationship, you noted that at least some courses do not generate a class. In other words, an entity occurrence (row) in the COURSE table does not necessarily require the existence of a corresponding entity occurrence in the CLASS table. (Remember that each entity is implemented as a table.

) Therefore, the CLASS entity is considered to be optional to the COURSE entity. In Crow’s Foot notation, an optional relationship between entities is shown by drawing a small circle (O) on the side of the optional entity, as illustrated in Figure 4. 9. The existence of an optional entity indicates that the minimum cardinality is 0 for the optional entity. (The term optionality is used to label any condition in which one or more optional relationships exist.

) Note Remember that the burden of establishing the relationship is always placed on the entity that contains the foreign key. In most cases, that will be the entity on the “ many” side of the relationship. Mandatory participation means that one entity occurrence requires a corresponding entity occurrence in a particular relationship. If no optionality symbol is depicted with the entity, the entity is assumed to exist in a mandatory relationship with the related entity. If the mandatory participation is depicted graphically, it is typically shown as a small 114 C H A P T E R 4 hash mark across the relationship line, similar to the Crow’s Foot depiction of a connectivity of 1. The existence of a mandatory relationship indicates that the minimum cardinality is at least 1 for the mandatory entity.

NoteYou might be tempted to conclude that relationships are weak when they occur between entities in an optional relationship and that relationships are strong when they occur between entities in a mandatory relationship. However, this conclusion is not warranted. Keep in mind that relationship participation and relationship strength do not describe the same thing. You are likely to encounter a strong relationship when one entity is optional to another. For example, the relationship between EMPLOYEE and DEPENDENT is clearly a strong one, but DEPENDENT is clearly optional to EMPLOYEE.

After all, you cannot require employees to have dependents. And it is just as possible for a weak relationship to be established when one entity is mandatory to another. The relationship strength depends on how the PK of the related entity is formulated, while the relationship participation depends on how the business rule is written. For example, the business rules “ Each part must be supplied by a vendor” and “ A part may or may not be supplied by a vendor” create different optionalities for the same entities! Failure to understand this distinction may lead to poor design decisions that cause major problems when table rows are inserted or deleted.

When you create a relationship in MS Visio, the default relationship will be mandatory on the “ 1” side and optional on the “ many” side. Table 4. 3 shows the various connectivity and participation combinations that are supported by the Crow’s Foot notation. Recall that these combinations are often referred to as cardinality in Crow’s Foot notation when specific cardinalities are not used. TABLE 4.

3 Crow’s Foot Symbols CARDINALITY (0, N) (1, N) (1, 1) (0, 1) COMMENT Zero or many. Many side is optional. One or many. Many side is mandatory. One and only one. 1 side is mandatory.

Zero or one. side is optional. CROW'S FOOT SYMBOL Because relationship participation turns out to be a very important component of the database design process, let's examine a few more scenarios. Suppose that Tiny College employs some professors who conduct research without teaching classes.

If you examine the " PROFESSOR teaches CLASS" relationship, it is quite possible for a PROFESSOR not to teach a CLASS. Therefore, CLASS is optional to PROFESSOR. On the other hand, a CLASS must be taught by a PROFESSOR. Therefore, PROFESSOR is mandatory to CLASS. Note that the ERD model in Figure 4.

2 shows the cardinality next to CLASS to be (0, 3), thus indicating that a professor may teach no classes at all or as many as three classes. And each CLASS table row will reference one and only one PROFESSOR row—assuming each class is taught by one and only one professor—represented by the (1, 1) cardinality next to the PROFESSOR table. Failure to understand the distinction between mandatory and optional participation in relationships might yield designs in which awkward (and unnecessary) temporary rows (entity instances) must be created just to accommodate the creation of required entities. Therefore, it is important that you clearly understand the concepts of mandatory and optional participation. It is also important to understand that the semantics of a problem might determine the type of participation in a relationship. For example, suppose that Tiny College offers several courses; each course has several classes.

E N T I T Y R E L A T I O N S H I P (E R) M O D E L I N G FIGURE

An optional CLASS entity in the relationship “ PROFESSOR teaches CLASS” 4. 12 again the distinction between class and course in this discussion: a CLASS constitutes a specific offering (or section) of a COURSE. Typically, courses are listed in the university’s course catalog, while classes are listed in the class schedules that students use to register for their classes.) Analyzing the CLASS entity’s contribution to the “ COURSE generates CLASS” relationship, it is easy to see that a CLASS cannot exist without a COURSE.

Therefore, you can conclude that the COURSE entity is mandatory in the relationship. But two scenarios for the CLASS entity may be written, shown in Figures 4. 13 and 4. 14.

FIGURE CLASS is optional to COURSE 4. 13 FIGURE COURSE and CLASS in a mandatory relationship . 14 The different scenarios are a function of the semantics of the problem; that is, they depend on how the relationship is defined. 1.

CLASS is optional. It is possible for the department to create the entity COURSE first and then create the CLASS entity after making the teaching assignments. In the real world, such a scenario is very likely; there may be courses for which sections (classes) have not yet been defined. In fact, some courses are taught only once a year and do not generate classes each semester. CLASS is mandatory. This condition is created by the constraint that is imposed by the semantics of the statement “ Each COURSE generates one or more CLASSes.

” In ER terms, each COURSE in the “ generates” relationship must have at least one CLASS. Therefore, a CLASS must be created as the COURSE is created, in order to comply with the semantics of the problem. 2. Keep in mind the practical aspects of the scenario presented in Figure 4. 14. Given the semantics of this relationship, the system should not accept a course that is not associated with at least one class section.

Is such a rigid environment 116C H A P T E R 4 desirable from an operational point of view? For example, when a new COURSE is created, the database first updates the COURSE table, thereby inserting a COURSE entity that does not yet have a CLASS associated with it. Naturally, the apparent problem seems to be solved when CLASS entities are inserted into the corresponding CLASS table. However, because of the mandatory relationship, the system will be in temporary violation of the business rule constraint. For practical purposes, it would be desirable to classify the CLASS as optional in order to produce a more flexible design. Finally, as you examine the scenarios presented in Figures 4. 13 and 4.

14, keep in mind the role of the DBMS. To maintain data integrity, the DBMS must ensure that the “ many” side (CLASS) is associated with a COURSE through the foreign key rules.

4. 1. 9 Relationship Degree

A relationship degree indicates the number of entities or participants associated with a relationship.

A unary relationship exists when an association is maintained within a single entity. A binary relationship exists when two entities are associated. A ternary relationship exists when three entities are associated. Although higher degrees exist, they are rare and are not specifically named.

(For example, an association of four entities is described simply as a four-degree relationship.) Figure 4. 15 shows these types of relationship degrees. Unary Relationships In the case of the unary relationship shown in Figure 4. 15, an employee within the EMPLOYEE entity is the manager for one or more employees within that entity.

In this case, the existence of the “ manages” relationship means that EMPLOYEE requires another EMPLOYEE to be the manager—that is, EMPLOYEE has a relationship with itself. Such a relationship is known as a recursive relationship. The various cases of recursive relationships will be explored in Section.

4.

1. 10. Binary Relationships

A binary relationship exists when two entities are associated in a relationship. Binary relationships are most common. In fact, to simplify the conceptual design, whenever possible, most higher-order (ternary and higher) relationships are decomposed into appropriate equivalent binary relationships. In Figure 4.

15, the relationship “ a PROFESSOR teaches one or more CLASSes” represents a binary relationship. Ternary and Higher-Degree RelationshipsAlthough most relationships are binary, the use of ternary and higher-order relationships does allow the designer some latitude regarding the semantics of a problem. A ternary relationship implies an association among three different entities. For example, note the relationships (and their consequences) in Figure 4. 16, which are represented by the following business rules: ? ? ? A DOCTOR writes one or more PRESCRIPTIONs. A PATIENT may receive one or more PRESCRIPTIONs.

A DRUG may appear in one or more PRESCRIPTIONs. (To simplify this example, assume that the business rule states that each prescription contains only one drug. In short, if a doctor prescribes more than one drug, a separate prescription must be written for each drug.) As you examine the table contents in Figure 4.

16, note that it is possible to track all transactions. For instance, you can tell that the first prescription was written by doctor 32445 for patient 102, using the drug DRZ.

E N T I T Y R E L A T I O N S H I P (E R) M O D E L I N G FIGURE

Three types of relationship degree 4. 15 4. 1.

10 Recursive Relationships As was previously mentioned, a recursive relationship is one in which a relationship can exist between occurrences of the same entity set. Naturally, such a condition is found within a unary relationship.) For example, a 1: M unary relationship can be expressed by “ an EMPLOYEE may manage many EMPLOYEEs, and each EMPLOYEE is managed by one EMPLOYEE. ” And as long as polygamy is not legal, a 1: 1 unary relationship may be expressed by “ an EMPLOYEE may be married to one and only one other EMPLOYEE. ” Finally, the M: N unary relationship may

be expressed by “ a COURSE may be a prerequisite to many other COURSEs, and each COURSE may have many other COURSEs as prerequisites.

” Those relationships are shown in Figure 4. 17. The 1: 1 relationship shown in Figure 4. 7 can be implemented in the single table shown in Figure 4. 18.

Note that you can determine that James Ramirez is married to Louise Ramirez, who is married to James Ramirez. And Anne Jones is married to Anton Shapiro, who is married to Anne Jones. 118 C H A P T E R 4 FIGURE The implementation of a ternary relationship 4. 16 Database name: Ch04_Clinic Table name: DRUG Table name: PATIENT Table name: DOCTOR Table name: PRESCRIPTION FIGURE An ER representation of recursive relationships 4. 17 FIGURE 4. 18 The 1: 1 recursive relationship “ EMPLOYEE is married to EMPLOYEE” Database name: CH04_PartCo Table name: EMPLOYEE_V1Unary relationships are common in manufacturing industries.

For example, Figure 4. 19 illustrates that a rotor assembly (C-130) is composed of many parts, but each part is used to create only one rotor assembly. Figure 4. 19 indicates that a rotor assembly is composed of four 2.

5-cm washers, two cotter pins, one 2. 5-cm steel shank, four 10. 25-cm rotor blades, and two 2. 5-cm hex nuts. The relationship implemented in Figure 4. 19 thus enables you to track each part within each rotor assembly.

If a part can be used to assemble several different kinds of other parts and is itself composed of many parts, two tables

E N T I T Y R E L A T I O N S H I P (E R) M O D E L I N G FIGURE

Another unary relationship: “ PART contains PART” 4. 19 Table name: PART_V1 Database name; CH04_PartCo are required to implement the “ PART contains PART” relationship. Figure 4. 20 illustrates such an environment.

Parts tracking is increasingly important as managers become more aware of the legal ramifications of producing more complex output. In fact, in many industries, especially those involving aviation, full parts tracking is required by law. FIGURE Implementation of the M: N recursive relationship “ PART contains PART” 4. 20Table name: COMPONENT Database name: Ch04_PartCo Table name: PART The M: N recursive relationship might be more familiar in a school environment. For instance, note how the M: N “ COURSE requires COURSE” relationship illustrated in Figure 4. 17 is implemented in Figure 4.

21. In this example, MATH-243 is a prerequisite to QM-261 and QM-362, while both MATH-243 and QM-261 are prerequisites to QM-362. Finally, the 1: M recursive relationship “ EMPLOYEE manages EMPLOYEE,” shown in Figure 4. 17, is implemented in Figure 4. 22. One common pitfall when working with unary relationships is to confuse participation with referential integrity.

In theory, participation and referential integrity are very different concepts and are normally easy to distinguish in binary relationships. In practical terms, conversely, participation and referential integrity are very similar because they are both implemented through constraints on the same set of attributes. This similarity often leads to confusion when the concepts are applied within the limited structure of a unary relationship. Consider the unary 1: 1 relationship described in Figure 4. 18 of a spousal relationship between employees. Participation, as described above, is bidirectional, 20 C H A P T E R 4 FIGURE Implementation of the M: N recursive relationship " COURSE requires COURSE" 4. 21 Table name: COURSE Database name: Ch04_TinyCollege Table name: PREREQ FIGURE Implementation of the 1: M recursive relationship " EMPLOYEE manages EMPLOYEE" 4. 22 Database name: Ch04_PartCo Table name: EMPLOYEE_V2 meaning that it must be addressed in both directions along the relationship. Participation in Figure 4. 18 addresses the questions: ? ? Must every employee have a spouse who is an employee? Must every employee be a spouse to another employee? For the data shown in Figure 4. 18, the correct answer to both of those questions is " No. " It is possible to be an employee and not have another employee as a spouse. Also, it is possible to be an employee and not be the spouse of another employee. Referential integrity deals with the correspondence of values in the foreign key with values in the related primary key. Referential integrity is not bidirectional, and therefore has only one question that it answers. ? Must every employee spouse be a valid employee? For the data shown in Figure 4. 18, the correct answer is " Yes. Another way to frame this question is to consider whether or not every value provided for the EMP_SPOUSE attribute must match some value in the EMP_NUM attribute. In practical terms, both participation and referential integrity involve the values used as primary key/foreign key to implement the relationship. Referential integrity requires that the values in the foreign key correspond to values in the primary key. In one direction, participation considers whether or not the foreign key can contain a null. In Figure 4. 18, E N T I T Y R E L A T I O N S H I

P (E R) M O D E L I N G 121 or example, employee Robert Delaney is not required to have a value in EMP_SPOUSE. In the other direction, participation considers whether or not every value in the primary key must appear as a value in the foreign key. In Figure 4. 18, for example, employee Robert Delaney’s value for EMP_NUM (348) is not required to appear as a value in EMP_SPOUSE for any other employee.

4. 1. 11 Associative (Composite) Entities

In the original ERM described by Chen, relationships do not contain attributes. You should recall from Chapter 3 that the relational model generally requires the use of 1: M relationships. Also, recall that the 1: 1 relationship has its place, but it should be used with caution and proper justification.) If M: N relationships are encountered, you must create a bridge between the entities that display such relationships. The associative entity is used to implement a M: N relationship between two or more entities. This associative entity (also known as a composite or bridge entity) is composed of the primary keys of each of the entities to be connected. An example of such a bridge is shown in Figure 4. 23. The Crow’s Foot notation does not identify the composite entity as such. Instead, the composite entity is identified by the solid relationship line between the parent and child entities, thereby indicating the presence of a strong (identifying) relationship. FIGURE Converting the M: N relationship into two 1: M relationships 4. 23 Table name: STUDENT Database name: Ch04_CollegeTry Table name: ENROLL Table name: CLASS Note that the composite ENROLL entity in Figure 4. 23 is existence-dependent on the other two entities; the composition of the ENROLL entity is based on the primary keys of the entities that are connected by the composite entity. The composite entity may also contain additional attributes that play no role in the connective process. For example, although the entity must be composed of at least the STUDENT and CLASS primary keys, it may also include such additional attributes as grades, absences, and other data uniquely identified by the student’s performance in a specific class. Finally, keep in mind that the ENROLL table’s key (CLASS_CODE and STU_NUM) is composed entirely of the primary keys of the CLASS and STUDENT tables. Therefore, no null entries are possible in the ENROLL table’s key attributes. Implementing the small database shown in Figure 4. 3 requires that you define the relationships clearly. Specifically, you must know the “ 1” and the “ M” sides of each relationship, and you must know whether the relationships are mandatory or optional. For example, note the following points: 122 C H A P T E R 4 ? A class may exist (at least at the start of registration) even though it contains no students. Therefore, if you examine Figure 4. 24, an optional symbol should appear on the STUDENT side of the M: N relationship between STUDENT and CLASS. You might argue that to be classified as a STUDENT, a person must be enrolled in at least one CLASS. Therefore, CLASS is mandatory to STUDENT from a purely conceptual point of view. However, when a student is admitted to college, that student has not (yet) signed up for any classes. Therefore, at least initially, CLASS is optional to STUDENT. Note that the practical considerations in the data environment help dictate the use of optionalities. If CLASS is not optional to STUDENT—from a database point of view—a class assignment must be made when the student is admitted. But that’s not how the process actually works, and the database design must reflect this. In short, the optionality reflects practice. FIGURE The M: N relationship between

STUDENT and CLASS 4. 24 Because the M: N relationship between STUDENT and CLASS is decomposed into two 1: M relationships through ENROLL, the optionalities must be transferred to ENROLL. (See Figure 4. 25.) In other words, it now becomes possible for a class not to occur in ENROLL if no student has signed up for that class. Because a class need not occur in ENROLL, the ENROLL entity becomes optional to CLASS. And because the ENROLL entity is created before any students have signed up for a class, the ENROLL entity is also optional to STUDENT, at least initially. FIGUREA composite entity in an ERD 4. 25 ? As students begin to sign up for their classes, they will be entered into the ENROLL entity. Naturally, if a student takes more than one class, that student will occur more than once in ENROLL. For example, note that in the ENROLL table in Figure 4. 23, STU_NUM = 321452 occurs three times. On the other hand, each student occurs only once in the STUDENT entity. (Note that the STUDENT table in Figure 4. 23 has only one STU_NUM = 321452 entry.) Therefore, in Figure 4. 25, the relationship between STUDENT and ENROLL is shown to be 1: M, with the M on the ENROLL side.

As you can see in Figure 4. 23, a class can occur more than once in the ENROLL table. For example, CLASS_CODE = 10014 occurs twice. However, CLASS_CODE = 10014 occurs only once in the CLASS table to reflect that the relationship between CLASS and ENROLL is 1: M. Note that in Figure 4. 25, the M is located on the ENROLL side, while the 1 is located on the CLASS side. 4. 2 DEVELOPING AN ER DIAGRAM The process of database design is an iterative rather than a linear or sequential process. The verb iterate means “

to do again or repeatedly. An iterative process is, thus, one based on repetition of processes and procedures. Building an ERD usually involves the following activities: ? ? ? ? ? ? Create a detailed narrative of the organization’s description of operations. Identify the business rules based on the description of operations. Identify the main entities and relationships from the business rules. Develop the initial ERD. Identify the attributes and primary keys that adequately describe the entities. Revise and review the ERD. During the review process, it is likely that additional objects, attributes, and relationships will be uncovered. Therefore, the basic ERM will be modified to incorporate the newly discovered ER components. Subsequently, another round of reviews might yield additional components or clarification of the existing diagram. The process is repeated until the end users and designers agree that the ERD is a fair representation of the organization’s activities and functions. During the design process, the database designer does not depend simply on interviews to help define entities, attributes, and relationships. A surprising amount of information can be gathered by examining the business forms and reports that an organization uses in its daily operations. To illustrate the use of the iterative process that ultimately yields a workable ERD, let’s start with an initial interview with the Tiny College administrators. The interview process yields the following business rules: 1. Tiny College (TC) is divided into several schools: a school of business, a school of arts and sciences, a school of education, and a school of applied sciences. Each school is administered by a dean who is a professor. Each professor can be the dean of only one school, and a professor is not required to be the dean of any school. Therefore, a 1: 1 relationship exists between PROFESSOR and SCHOOL. Note that the cardinality can be expressed by writing (1, 1) next to the entity PROFESSOR and (0, 1) next to the entity SCHOOL. Each school comprises several departments. For example, the school of business has an accounting department, a management/marketing department, an economics/finance department, and a computer information systems department. Note again the cardinality rules: The smallest number of departments operated by a school is one, and the largest number of departments is indeterminate (N). On the other hand, each department belongs to only a single school; thus, the cardinality is expressed by (1, 1). That is, the minimum number of schools that a department belongs to is one, as is the maximum number. Figure 4. 26 illustrates these first two business rules. 2. 124 C H A P T E R 4 FIGURE The first Tiny College ERD segment 4. 26 Note It is again appropriate to evaluate the reason for maintaining the 1: 1 relationship between PROFESSOR and SCHOOL in the PROFESSOR is dean of SCHOOL relationship. It is worth repeating that the existence of 1: 1 relations