4.1 Logical View of Data

Database Systems, you learned that database stores and manages both data and metadata. You also learned that the DBMS manages and controls access to the data and the database structure. Such an arrangement—placing the DBMS between the application and the database—eliminates most of the file system's inherent limitations.

The result of such flexibility, however, is a far more complex physical structure. In fact, the database structures required by both the hierarchical and network database models often become complicated enough to diminish efficient database design. The relational data model changed all of that by allowing the designer to focus on the logical representation of the data and its relationships, rather than on the physical storage details. To use an automotive analogy, the relational database uses an automatic transmission to relieve you of the need to manipulate clutch pedals and gearshifts. In short, the relational model enables you to view data logically rather than physically.

The practical significance of taking the logical view is that it serves as a reminder of the simple file concept of data storage. Although the use of a table, quite unlike that of a file, has the advantages of structural and data independence, a table does resemble a file from a conceptual point of view. Because you can think of related records as being stored in independent tables, the relational database model is much easier to understand than the hierarchical and network models. Logical simplicity tends to yield simple and effective database design methodologies.

Because the table plays such a prominent role in the relational model, it deserves a closer look. Therefore, our discussion begins by exploring the details of table structure and contents.

Tables and Their Characteristics

The logical view of the relational database is facilitated by the creation of data relationships based on a logical construct known as a relation. Because a relation is a mathematical construct, end users find it much easier to think of a relation as a table. A table is perceived as a two-dimensional structure composed of rows and columns. A table is also called a relation because the relational model's creator, E. F. Codd, used the two terms as synonyms. You can think of a table as a persistent representation of a logical relation—that is, a relation whose contents can be permanently saved for future use. As far as the table's user is concerned, a table contains a group of related entity occurrences—that is, an entity set. For example, a STUDENT table contains a collection of entity occurrences, each representing a student. For that reason, the terms entity set and table are often used interchangeably.

You will discover that the table view of data makes it easy to spot and define entity relationships, thereby greatly simplifying the task of database design. The characteristics of a relational table are summarized in Table 3.1.

TABLE 3.1

CHARACTERISTICS OF A RELATIONAL TABLE

1	A table is perceived as a two-dimensional structure composed of rows and columns.
2	Each table row (**tuple**) represents a single entity occurrence within the entity set.
3	Each table column represents an attribute, and each column has a distinct name.
4	Each intersection of a row and column represents a single data value.
5	All values in a column must conform to the same data format.
6	Each column has a specific range of values known as the **attribute domain**.
7	The order of the rows and columns is immaterial to the DBMS.
8	Each table must have an attribute or combination of attributes that uniquely identifies each row.

The database table shown in Figure 3.1 illustrates the characteristics listed in Table 3.1.

FIGURE 3.1 STUDENT TABLE ATTRIBUTE VALUES

Table name: STUDENT **Database name: Ch03_TinyCollege**

STU_NUM	STU_LNAME	STU_FNAME	STU_INIT	STU_DOB	STU_HRS	STU_CLASS	STU_GPA	STU_TRANSFER	DEPT_CODE	STU_PHONE	PROF_NUM
321452	Bowser	William	C	12-Feb-1985	42	So	2.84	No	BIOL	2134	205
324257	Smithson	Anne	K	15-Nov-1991	81	Jr	3.27	Yes	CIS	2256	222
324258	Brewer	Juliette		23-Aug-1979	36	So	2.26	Yes	ACCT	2256	228
324269	Oblonski	Walter	H	16-Sep-1986	66	Jr	3.09	No	CIS	2114	222
324273	Smith	John	D	30-Dec-1968	102	Sr	2.11	Yes	ENGL	2231	199
324274	Katinga	Raphael	P	21-Oct-1989	114	Sr	3.15	No	ACCT	2267	228
324291	Robertson	Gerald	T	08-Apr-1983	120	Sr	3.87	No	EDU	2267	311
324299	Smith	John	B	30-Nov-1996	15	Fr	2.92	No	ACCT	2315	230

STU_NUM = Student number
STU_LNAME = Student last name
STU_FNAME = Student first name
STU_INIT = Student middle initial
STU_DOB = Student date of birth
STU_HRS = Credit hours earned
STU_CLASS = Student classification
STU_GPA = Grade point average
STU_TRANSFER = Student transferred from another institution
DEPT_CODE = Department code
STU_PHONE = 4-digit campus phone extension
PROF_NUM = Number of the professor who is the student's advisor

Using the STUDENT table shown in Figure 3.1, you can draw the following conclusions corresponding to the points in Table 3.1:

1. The STUDENT table is perceived to be a two-dimensional structure composed of 8 rows (tuples) and 12 columns (attributes).

2. Each row in the STUDENT table describes a single entity occurrence within the entity set. (The entity set is represented by the STUDENT table.) For example, row 4 in Figure 3.1 describes a student named Walter H. Oblonski. Given the table contents, the STUDENT entity set includes eight distinct entities (rows) or students.

3. Each column represents an attribute, and each column has a distinct name.

4. All of the values in a column match the attribute's characteristics. For example, the grade point average (STU_GPA) column contains only STU_GPA entries for each of the table rows. Data must be classified according to its format and function. Although various DBMSs can support different data types, most support at least the following:

a. Numeric. You can use numeric data to perform meaningful arithmetic procedures. For example, in Figure 3.1, STU_HRS and STU_GPA are numeric attributes.

b. Character. Character data, also known as text data or string data, can contain any character or symbol not intended for mathematical manipulation. In Figure 3.1, STU_CLASS and STU_PHONE are examples of character attributes.

c. Date. Date attributes contain calendar dates stored in a special format known as the Julian date format. In Figure 3.1, STU_DOB is a date attribute.

d. Logical. Logical data can only have true or false (yes or no) values. In Figure 3.1, the STU_TRANSFER attribute uses a logical data format.

5. The column's range of permissible values is known as its domain. Because the STU_GPA values are limited to the range 0–4, inclusive, the domain is [0,4].

6. The order of rows and columns is immaterial to the user.

7. Each table must have a primary key. In general terms, the primary key (PK) is an attribute or combination of attributes that uniquely identifies any given row. In this case, STU_NUM (the student number) is the primary key. Using the data in Figure 3.1, observe that a student's last name

(STU_LNAME) would not be a good primary key because several students have the last name of Smith. Even the combination of the last name and first name (STU_FNAME) would not be an appropriate primary key because more than one student is named John Smith.

4.2 Keys

In the relational model, keys are important because they are used to ensure that each row in a table is uniquely identifiable. They are also used to establish relationships among tables and to ensure the integrity of the data. A key consists of one or more attributes that determine other attributes.

Dependencies

The role of a key is based on the concept of determination. Determination is the state in which knowing the value of one attribute makes it possible to determine the value of another. The idea of determination is not unique to the database environment. You are familiar with the formula revenue − cost = profit. This is a form of determination, because if you are given the revenue and the cost, you can determine the profit. Given profit and revenue, you can determine the cost. Given any two values, you can determine the third. Determination in a database environment, however, is not normally based on a formula but on the relationships among the attributes.

If you consider what the attributes of the STUDENT table in Figure 3.1 actually represent, you will see a relationship among the attributes. If you are given a value for STU_NUM, then you can determine the value for STU_LNAME because one and only one value of STU_LNAME is associated with any given value of STU_NUM. A specific terminology and notation is used to describe relationships based on determination. The relationship is called functional dependence, which means that the value of one or more attributes determines the value of one or more other attributes. The standard notation for representing the relationship between STU_NUM and STU_LNAME is as follows:

STU_NUM --> STU_LNAME

In this functional dependency, the attribute whose value determines another is called the determinant or the key. The attribute whose value is determined by the other attribute is called the dependent. Using this terminology, it would be correct to say that STU_NUM is the determinant and STU_LNAME is the dependent. STU_NUM functionally determines STU_LNAME, and STU_LNAME is functionally dependent on STU_NUM. As stated earlier, functional dependence can involve a determinant that comprises more than one attribute and multiple dependent attributes. Refer to the STUDENT table for the following example:

STU_NUM --> (STU_LNAME, STU_FNAME, STU_GPA)

and

(STU_FNAME, STU_LNAME, STU_INIT, STU_PHONE) --> (STU_DOB, STU_HRS, STU_GPA)

Determinants made of more than one attribute require special consideration. It is possible to have a functional dependency in which the determinant contains attributes that are not necessary for the relationship. Consider the following two functional dependencies:

STU_NUM --> STU_GPA

(STU_NUM, STU_LNAME) --> STU_GPA

In the second functional dependency, the determinant includes STU_LNAME, but this attribute is not necessary for the relationship. The functional dependency is valid because given a pair of values for STU_NUM and STU_LNAME, only one value would occur for STU_GPA. A more specific term, full functional dependence, is used to refer to functional dependencies in which the entire collection of attributes in the determinant is necessary for the relationship. Therefore, the dependency shown in the preceding example is a functional dependency, but not a full functional dependency.

Types of Keys

Recall that a key is an attribute or group of attributes that can determine the values of other attributes. Therefore, keys are determinants in functional dependencies. Several different types of keys are used in the relational model, and you need to be familiar with them.

A composite key is a key that is composed of more than one attribute. An attribute that is a part of a key is called a key attribute. For example,

STU_NUM --> STU_GPA

(STU_LNAME, STU_FNAME, STU_INIT, STU_PHONE) --> STU_HRS

In the first functional dependency, STU_NUM is an example of a key composed of only one key attribute. In the second functional dependency, (STU_LNAME, STU_FNAME, STU_INIT, STU_PHONE) is a composite key composed of four key attributes.

A superkey is a key that can uniquely identify any row in the table. In other words, a superkey functionally determines every attribute in the row. In the STUDENT table, STU_NUM is a superkey, as are the composite keys (STU_NUM, STU_LNAME), (STU_NUM, STU_LNAME, STU_INIT) and (STU_LNAME, STU_FNAME, STU_INIT, STU_PHONE). In fact, because STU_NUM alone is a superkey, any composite key that has STU_NUM as a key attribute will also be a superkey. Be careful, however, because not all keys are superkeys. For example, Gigantic State University determines its student classification based on hours completed, as shown in Table 3.2.

Therefore, you can write STU_HRS --> STU_CLASS.

However, the specific number of hours is not dependent on the classification. It is quite possible to find a junior with 62 completed hours or one with 84 completed hours. In other words, the classification (STU_CLASS) does not determine one and only one value for completed hours (STU_HRS).

TABLE 3.2

STUDENT CLASSIFICATION

HOURS COMPLETED	CLASSIFICATION
Less than 30	Fr
30–59	So
60–89	Jr
90 or more	Sr

A candidate key is a minimal superkey—that is, a superkey without any unnecessary attributes. A candidate key is based on a full functional dependency. For example, STU_NUM would be a candidate key, as would (STU_LNAME, STU_FNAME, STU_INIT, STU_PHONE). On the other hand, (STU_NUM,

STU_LNAME) is a superkey, but it is not a candidate key because STU_LNAME could be removed and the key would still be a superkey.

Entity integrity is the condition in which each row (entity instance) in the table has its own unique identity. To ensure entity integrity, the primary key has two requirements: (1) all of the values in the primary key must be unique and (2) no key attribute in the primary key can contain a null.

Null values are problematic in the relational model. A null is the absence of any data value, and it is never allowed in any part of the primary key. In fact, an abundance of nulls is often a sign of a poor design. Also, nulls should be avoided in the database because their meaning is not always identifiable. For example, a null could represent any of the following:

- An unknown attribute value

- A known, but missing, attribute value

- A “not applicable” condition

Depending on the sophistication of the application development software, nulls can create problems when functions such as COUNT, AVERAGE, and SUM are used. In addition, nulls can create logical problems when relational tables are linked.

A foreign key (FK) is the primary key of one table that has been placed into another table to create a common attribute. In Figure 3.2, the primary key of VENDOR, VEND_CODE, was placed in the PRODUCT table; therefore, VEND_CODE is a foreign key in PRODUCT.

FIGURE 3.2 AN EXAMPLE OF A SIMPLE RELATIONAL DATABASE

Table name: PRODUCT
Primary key: PROD_CODE
Foreign key: VEND_CODE

Database name: Ch03_SaleCo

PROD_CODE	PROD_DESCRIPT	PROD_PRICE	PROD_ON_HAND	VEND_CODE
001278-AB	Claw hammer	12.95	23	232
123-21UUY	Houselite chain saw, 16-in. bar	189.99	4	235
QER-34256	Sledge hammer, 16-lb. head	18.63	6	231
SRE-657UG	Rat-tail file	2.99	15	232
ZZX/3245Q	Steel tape, 12-ft. length	6.79	8	235

link

Table name: VENDOR
Primary key: VEND_CODE
Foreign key: none

VEND_CODE	VEND_CONTACT	VEND_AREACODE	VEND_PHONE
230	Shelly K. Smithson	608	555-1234
231	James Johnson	615	123-4536
232	Annelise Crystall	608	224-2134
233	Candice Wallace	904	342-6567
234	Arthur Jones	615	123-3324
235	Henry Ortozo	615	899-3425

Just as the primary key has a role in ensuring the integrity of the database, so does the foreign key. Foreign keys are used to ensure referential integrity, the condition in which every reference to an entity instance by another entity instance is valid.

A secondary key is defined as a key that is used strictly for data retrieval purposes. Suppose that customer data is stored in a CUSTOMER table in which the customer number is the primary key. Do you think that most customers will remember their numbers? Data retrieval for a customer is easier when the customer's last name and phone number are used.

TABLE 3.3

RELATIONAL DATABASE KEYS

KEY TYPE	DEFINITION
Superkey	An attribute or combination of attributes that uniquely identifies each row in a table
Candidate key	A minimal (irreducible) superkey; a superkey that does not contain a subset of attributes that is itself a superkey
Primary key	A candidate key selected to uniquely identify all other attribute values in any given row; cannot contain null entries
Foreign key	An attribute or combination of attributes in one table whose values must either match the primary key in another table or be null
Secondary key	An attribute or combination of attributes used strictly for data retrieval purposes

4.3. Integrity Rules

Relational database integrity rules are very important to good database design. Relational database management systems (RDBMSs) enforce integrity rules automatically, but it is much safer to make sure your application design conforms to the entity and referential integrity rules mentioned in this chapter. Those rules are summarized in Table 3.4.

TABLE 3.4

INTEGRITY RULES

ENTITY INTEGRITY	DESCRIPTION
Requirement	All primary key entries are unique, and no part of a primary key may be null.
Purpose	Each row will have a unique identity, and foreign key values can properly reference primary key values.
Example	No invoice can have a duplicate number, nor can it be null; in short, all invoices are uniquely identified by their invoice number.
REFERENTIAL INTEGRITY	**DESCRIPTION**
Requirement	A foreign key may have either a null entry, as long as it is not a part of its table's primary key, or an entry that matches the primary key value in a table to which it is related (every non-null foreign key value *must* reference an *existing* primary key value).
Purpose	It is possible for an attribute *not* to have a corresponding value, but it will be impossible to have an invalid entry; the enforcement of the referential integrity rule makes it impossible to delete a row in one table whose primary key has mandatory matching foreign key values in another table.
Example	A customer might not yet have an assigned sales representative (number), but it will be impossible to have an invalid sales representative (number).

The integrity rules summarized in Table 3.4 are illustrated in Figure 3.3.

FIGURE 3.3 AN ILLUSTRATION OF INTEGRITY RULES

Table name: CUSTOMER **Database name: Ch03_InsureCo**
Primary key: CUS_CODE
Foreign key: AGENT_CODE

CUS_CODE	CUS_LNAME	CUS_FNAME	CUS_INITIAL	CUS_RENEW_DATE	AGENT_CODE
10010	Ramas	Alfred	A	05-Apr-2018	502
10011	Dunne	Leona	K	16-Jun-2018	501
10012	Smith	Kathy	W	29-Jan-2019	502
10013	Olowski	Paul	F	14-Oct-2018	
10014	Orlando	Myron		28-Dec-2018	501
10015	O'Brian	Amy	B	22-Sep-2018	503
10016	Brown	James	G	25-Mar-2019	502
10017	Williams	George		17-Jul-2018	503
10018	Farriss	Anne	G	03-Dec-2018	501
10019	Smith	Olette	K	14-Mar-2019	503

Table name: AGENT (only five selected fields are shown)
Primary key: AGENT_CODE
Foreign key: none

AGENT_CODE	AGENT_AREACODE	AGENT_PHONE	AGENT_LNAME	AGENT_YTD_SLS
501	713	228-1249	Alby	132735.75
502	615	882-1244	Hahn	138967.35
503	615	123-5589	Okon	127093.45

Note the following features of Figure 3.3.

• Entity integrity. The CUSTOMER table's primary key is CUS_CODE. The CUSTOMER primary key column has no null entries, and all entries are unique. Similarly, the AGENT table's primary key is AGENT_CODE, and this primary key column is also free of null entries.

• Referential integrity. The CUSTOMER table contains a foreign key, AGENT_CODE, that links entries in the CUSTOMER table to the AGENT table. The CUS_CODE row identified by the (primary key) number 10013 contains a null entry in its AGENT_ CODE foreign key because Paul F. Olowski does not yet have a sales representative assigned to him. The remaining AGENT_CODE entries in the CUSTOMER table all match the AGENT_CODE entries in the AGENT table.

To avoid nulls, some designers use special codes, known as flags, to indicate the absence of some value.

4.4. The Data Dictionary and the System Catalog

The data dictionary provides a detailed description of all tables in the database created by the user and designer. Thus, the data dictionary contains at least all of the attribute names and characteristics for each table in the system. In short, the data dictionary contains metadata—data about data. Using the small database presented in Figure 3.3, you might picture its data dictionary as shown in Table 3.6.

TABLE 3.6

A SAMPLE DATA DICTIONARY

TABLE NAME	ATTRIBUTE NAME	CONTENTS	TYPE	FORMAT	RANGE	REQUIRED	PK OR FK	FK REFERENCED TABLE
CUSTOMER	CUS_CODE	Customer account code	CHAR(5)	99999	10000–99999	Y	PK	
	CUS_LNAME	Customer last name	VARCHAR(20)	Xxxxxxxx		Y		
	CUS_FNAME	Customer first name	VARCHAR(20)	Xxxxxxxx		Y		
	CUS_INITIAL	Customer initial	CHAR(1)	X				
	CUS_RENEW_DATE	Customer insurance renewal date	DATE	dd-mmm-yyyy				
	AGENT_CODE	Agent code	CHAR(3)	999			FK	AGENT
AGENT	AGENT_CODE	Agent code	CHAR(3)	999		Y	PK	
	AGENT_AREACODE	Agent area code	CHAR(3)	999		Y		
	AGENT_PHONE	Agent telephone number	CHAR(8)	999–9999		Y		
	AGENT_LNAME	Agent last name	VARCHAR(20)	Xxxxxxxx		Y		
	AGENT_YTD_SLS	Agent year-to-date sales	NUMBER(9,2)	9,999,999.99				

FK	= Foreign key
PK	= Primary key
CHAR	= Fixed character length data (1 – 255 characters)
VARCHAR	= Variable character length data (1 – 2,000 characters)
NUMBER	= Numeric data. NUMBER (9,2) is used to specify numbers with up to nine digits, including two digits to the right of the decimal place. Some RDBMS permit the use of a MONEY or CURRENCY data type.

The data dictionary is sometimes described as “the database designer’s database”because it records the design decisions about tables and their structures.

Like the data dictionary, the system catalog contains metadata. The system catalog can be described as a detailed system data dictionary that describes all objects within the database, including data about table names, table’s creator and creation date, number of columns in each table, data type corresponding to each column, index filenames, index creators, authorized users, and access privileges. Because the system catalog contains all required data dictionary information, the terms system catalog and data dictionary are often used interchangeably. In fact, current relational database software generally provides only a system catalog, from which the designer’s data dictionary information may be derived. The system catalog is actually a system-created database whose tables store the user/designer-created database characteristics and contents. Therefore, the system catalog tables can be queried just like any user/designer-created table.

4.5. Relationships within the Relational Database

You already know that relationships are classified as one-to-one (1:1), one-to-many (1:M), and many-to-many (M:N or M:M). This section explores those relationships further to help you apply them properly when you start developing database designs. This section focuses on the following points:

• The 1:M relationship is the relational modeling ideal. Therefore, this relationship type should be the norm in any relational database design.

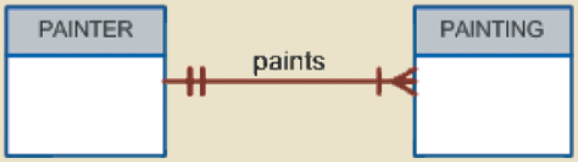
• The 1:1 relationship should be rare in any relational database design.

• M:N relationships cannot be implemented as such in the relational model. Later in this section, you will see how any M:N relationship can be changed into two 1:M relationships.

The 1:M Relationship

The 1:M relationship is the norm for relational databases. To see how such a relationship is modeled and implemented, consider the PAINTER and PAINTING example shown in Figure 3.17.

FIGURE 3.17 THE 1:M RELATIONSHIP BETWEEN PAINTER AND PAINTING

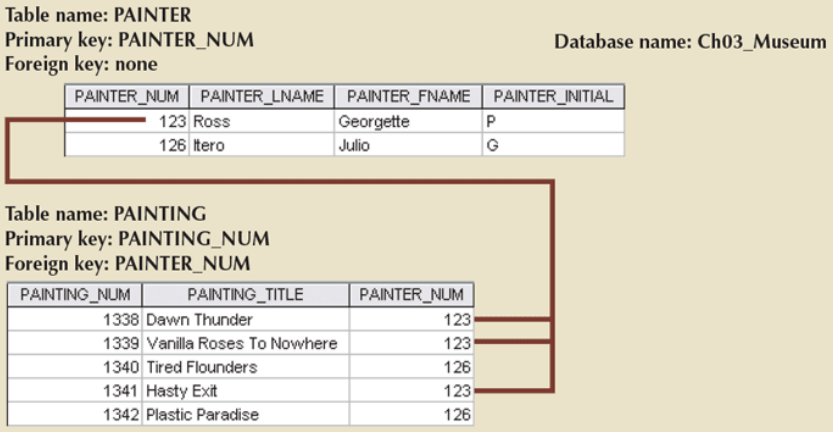


Compare the data model in Figure 3.17 with its implementation in Figure 3.18. As you examine the PAINTER and PAINTING table contents in Figure 3.18, note the following features:

• Each painting was created by one and only one painter, but each painter could have created many paintings. Note that painter 123 (Georgette P. Ross) has three works stored in the PAINTING table.

• There is only one row in the PAINTER table for any given row in the PAINTING table, but there may be many rows in the PAINTING table for any given row in the PAINTER table.

FIGURE 3.18 THE IMPLEMENTED 1:M RELATIONSHIP BETWEEN PAINTER AND PAINTING



The 1:1 Relationship

As the 1:1 label implies, one entity in a 1:1 relationship can be related to only one other entity, and vice versa. For example, one department chair—a professor—can chair only one department, and one department can have only one department chair. The entities PROFESSOR and DEPARTMENT thus exhibit a 1:1 relationship. (You might argue that not all professors chair a department and professors cannot be required to chair a department. That is, the relationship between the two entities is optional. However, at this stage of the discussion, you should focus your attention on the basic 1:1 relationship. (Optional relationships will be addressed in Chapter 4.) The basic 1:1 relationship is modeled in Figure 3.21, and its implementation is shown in Figure 3.22.

FIGURE 3.21 THE 1:1 RELATIONSHIP BETWEEN PROFESSOR AND DEPARTMENT

As you examine the tables in Figure 3.22, note several important features:

• Each professor is a Tiny College employee. Therefore, the professor identification is through the EMP_NUM. (However, note that not all employees are professors—there's another optional relationship.)

• The 1:1 "PROFESSOR chairs DEPARTMENT" relationship is implemented by having the EMP_NUM foreign key in the DEPARTMENT table. Note that the 1:1 relationship is treated as a special case of the

1:M relationship in which the "many" side is restricted to a single occurrence. In this case, DEPARTMENT contains the EMP_NUM as a foreign key to indicate that it is the department that has a chair.

FIGURE 3.22 THE IMPLEMENTED 1:1 RELATIONSHIP BETWEEN PROFESSOR AND DEPARTMENT

Table name: PROFESSOR
Primary key: EMP_NUM
Foreign key: DEPT_CODE

Database name: Ch03_TinyCollege

EMP_NUM	DEPT_CODE	PROF_OFFICE	PROF_EXTENSION	PROF_HIGH_DEGREE
103	HIST	DRE 156	6783	Ph.D.
104	ENG	DRE 102	5561	MA
105	ACCT	KLR 229D	8665	Ph.D.
106	MKT/MGT	KLR 126	3899	Ph.D.
110	BIOL	AAK 160	3412	Ph.D.
114	ACCT	KLR 211	4436	Ph.D.
155	MATH	AAK 201	4440	Ph.D.
160	ENG	DRE 102	2248	Ph.D.
162	CIS	KLR 203E	2359	Ph.D.
191	MKT/MGT	KLR 409B	4016	DBA
195	PSYCH	AAK 297	3550	Ph.D.
209	CIS	KLR 333	3421	Ph.D.
228	CIS	KLR 300	3000	Ph.D.
297	MATH	AAK 194	1145	Ph.D.
299	ECON/FIN	KLR 284	2851	Ph.D.
301	ACCT	KLR 244	4683	Ph.D.
335	ENG	DRE 208	2000	Ph.D.
342	SOC	BBG 208	5514	Ph.D.
387	BIOL	AAK 230	8665	Ph.D.
401	HIST	DRE 156	6783	MA
425	ECON/FIN	KLR 284	2851	MBA
435	ART	BBG 185	2278	Ph.D.

The 1:M DEPARTMENT employs PROFESSOR relationship is implemented through the placement of the DEPT_CODE foreign key in the PROFESSOR table.

The 1:1 PROFESSOR chairs DEPARTMENT relationship is implemented through the placement of the EMP_NUM foreign key in the DEPARTMENT table.

Table name: DEPARTMENT
Primary key: DEPT_CODE
Foreign key: EMP_NUM

DEPT_CODE	DEPT_NAME	SCHOOL_CODE	EMP_NUM	DEPT_ADDRESS	DEPT_EXTENSION
ACCT	Accounting	BUS	114	KLR 211, Box 52	3119
ART	Fine Arts	A&SCI	435	BBG 185, Box 128	2278
BIOL	Biology	A&SCI	387	AAK 230, Box 415	4117
CIS	Computer Info. Systems	BUS	209	KLR 333, Box 56	3245
ECON/FIN	Economics/Finance	BUS	299	KLR 284, Box 63	3126
ENG	English	A&SCI	160	DRE 102, Box 223	1004
HIST	History	A&SCI	103	DRE 156, Box 284	1867
MATH	Mathematics	A&SCI	297	AAK 194, Box 422	4234
MKT/MGT	Marketing/Management	BUS	106	KLR 126, Box 55	3342
PSYCH	Psychology	A&SCI	195	AAK 297, Box 438	4110
SOC	Sociology	A&SCI	342	BBG 208, Box 132	2008

Also note that the PROFESSOR table contains the DEPT_CODE foreign key to implement the 1:M "DEPARTMENT employs PROFESSOR" relationship. This is a good example of how two entities can participate in two (or even more) relationships simultaneously.

The preceding "PROFESSOR chairs DEPARTMENT" example illustrates a proper 1:1 relationship. In fact, the use of a 1:1 relationship ensures that two entity sets are not placed in the same table when they should not be. However, the existence of a 1:1 relationship sometimes means that the entity

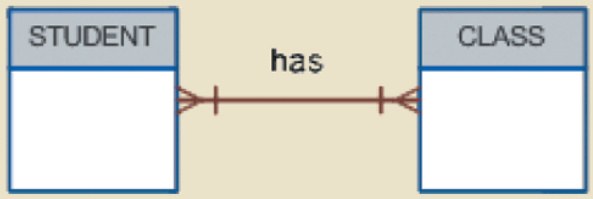
components were not defined properly. It could indicate that the two entities actually belong in the same table!

Although 1:1 relationships should be rare, certain conditions absolutely require their use.

The M:N Relationship

A many-to-many (M:N) relationship is not supported directly in the relational environment. However, M:N relationships can be implemented by creating a new entity in 1:M relationships with the original entities. To explore the M:N relationship, consider a typical college environment. The ER model in Figure 3.23 shows this M:N relationship.

FIGURE 3.23 THE ERM'S M:N RELATIONSHIP BETWEEN STUDENT AND CLASS



Note the features of the ERM in Figure 3.23.

• Each CLASS can have many STUDENTs, and each STUDENT can take many CLASSes.

• There can be many rows in the CLASS table for any given row in the STUDENT table, and there can be many rows in the STUDENT table for any given row in the CLASS table.

To examine the M:N relationship more closely, imagine a small college with two students, each of whom takes three classes. Table 3.7 shows the enrollment data for the two students.

TABLE 3.7

SAMPLE STUDENT ENROLLMENT DATA

STUDENT'S LAST NAME	SELECTED CLASSES
Bowser	Accounting 1, ACCT-211, code 10014 Intro to Microcomputing, CIS-220, code 10018 Intro to Statistics, QM-261, code 10021
Smithson	Accounting 1, ACCT-211, code 10014 Intro to Microcomputing, CIS-220, code 10018 Intro to Statistics, QM-261, code 10021

Given such a data relationship and the sample data in Table 3.7, you could wrongly assume that you could implement this M:N relationship simply by adding a foreign key in the "many" side of the relationship that points to the primary key of the related table, as shown in Figure 3.24.

FIGURE 3.24 THE WRONG IMPLEMENTATION OF THE M:N RELATIONSHIP BETWEEN STUDENT AND CLASS

Table name: STUDENT
Primary key: STU_NUM
Foreign key: none

Database name: Ch03_CollegeTry

STU_NUM	STU_LNAME	CLASS_CODE
321452	Bowser	10014
321452	Bowser	10018
321452	Bowser	10021
324257	Smithson	10014
324257	Smithson	10018
324257	Smithson	10021

Table name: CLASS
Primary key: CLASS_CODE
Foreign key: STU_NUM

CLASS_CODE	STU_NUM	CRS_CODE	CLASS_SECTION	CLASS_TIME	CLASS_ROOM	PROF_NUM
10014	321452	ACCT-211	3	TTh 2:30-3:45 p.m.	BUS252	342
10014	324257	ACCT-211	3	TTh 2:30-3:45 p.m.	BUS252	342
10018	321452	CIS-220	2	MWF 9:00-9:50 a.m.	KLR211	114
10018	324257	CIS-220	2	MWF 9:00-9:50 a.m.	KLR211	114
10021	321452	QM-261	1	MWF 8:00-8:50 a.m.	KLR200	114
10021	324257	QM-261	1	MWF 8:00-8:50 a.m.	KLR200	114

However, the M:N relationship should not be implemented as shown in Figure 3.24 for two good reasons:

• The tables create many redundancies. For example, note that the STU_NUM values occur many times in the STUDENT table. In a real-world situation, additional student attributes such as address, classification, major, and home phone would also be contained in the STUDENT table, and each of those attribute values would be repeated in each of the records shown here. Similarly, the CLASS table contains much duplication: each student taking the class generates a CLASS record. The problem would be even worse if the CLASS table included such attributes as credit hours and course description. Those redundancies lead to the anomalies discussed in Database Systems.

• Given the structure and contents of the two tables, the relational operations become very complex and are likely to lead to system efficiency errors and output errors.

Fortunately, the problems inherent in the M:N relationship can easily be avoided by creating a composite entity (also referred to as a bridge entity or an associative entity). Because such a table is used to link the tables that were originally related in an M:N relationship, the composite entity structure includes—as foreign keys—at least the primary keys of the tables that are to be linked. The database designer has two main options when defining a composite table's primary key: use the combination of those foreign keys or create a new primary key.

Remember that each entity in the ERM is represented by a table. Therefore, you can create the composite ENROLL table shown in Figure 3.25 to link the tables CLASS and STUDENT. In this example, the ENROLL table's primary key is the combination of its foreign keys CLASS_CODE and STU_NUM. However, the designer could have decided to create a single-attribute new primary key such as ENROLL_LINE, using a different line value to identify each ENROLL table row uniquely. (Microsoft Access users might use the Autonumber data type to generate such line values automatically.)

FIGURE 3.25 CONVERTING THE M:N RELATIONSHIP INTO TWO 1:M RELATIONSHIPS

Table name: STUDENT
Primary key: STU_NUM
Foreign key: none

Database name: Ch03_CollegeTry2

STU_NUM	STU_LNAME
321452	Bowser
324257	Smithson

Table name: ENROLL
Primary key: CLASS_CODE + STU_NUM
Foreign key: CLASS_CODE, STU_NUM

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

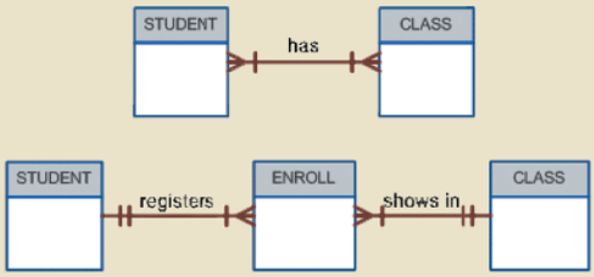
Table name: CLASS
Primary key: CLASS_CODE
Foreign key: CRS_CODE

CLASS_CODE	CRS_CODE	CLASS_SECTION	CLASS_TIME	CLASS_ROOM	PROF_NUM
10014	ACCT-211	3	TTh 2:30-3:45 p.m.	BUS252	342
10018	CIS-220	2	MWF 9:00-9:50 a.m.	KLR211	114
10021	QM-261	1	MWF 8:00-8:50 a.m.	KLR200	114

Because the ENROLL table in Figure 3.25 links two tables, STUDENT and CLASS, it is also called a linking table. In other words, a linking table is the implementation of a composite entity.

As you examine Figure 3.26, note that the composite entity named ENROLL represents the linking table between STUDENT and CLASS.

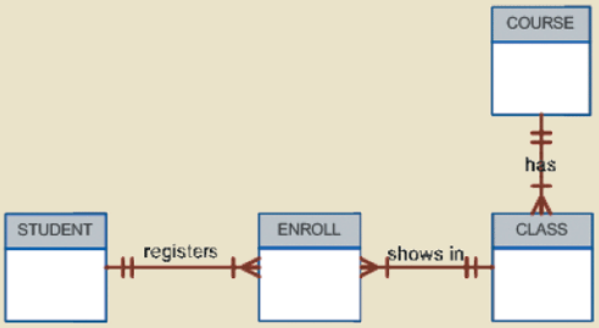
FIGURE 3.26 CHANGING THE M:N RELATIONSHIPS TO TWO 1:M RELATIONSHIPS



Because the ENROLL table in Figure 3.25 links two tables, STUDENT and CLASS, it is also called a linking table. In other words, a linking table is the implementation of a composite entity.

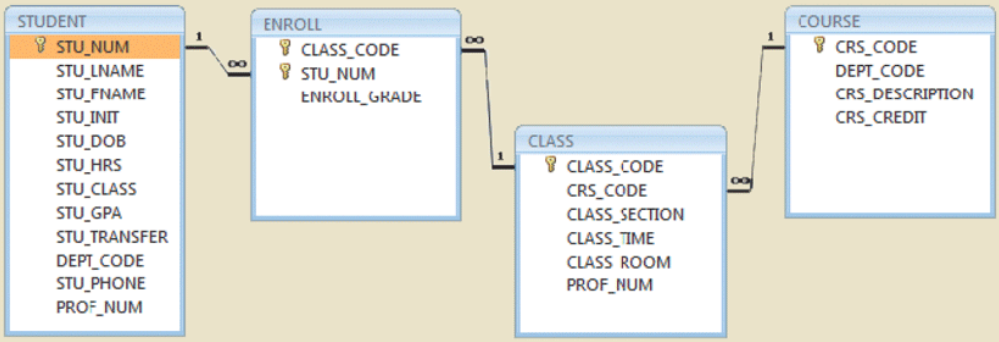
As you examine Figure 3.26, note that the composite entity named ENROLL represents the linking table between STUDENT and CLASS.

FIGURE 3.27 THE EXPANDED ER MODEL



The relational diagram that corresponds to the ERM in Figure 3.27 is shown in Figure 3.28.

FIGURE 3.28 THE RELATIONAL DIAGRAM FOR THE CH03_TINYCOLLEGE DATABASE



4.6. Data Redundancy Revisited

In Database Systems, you learned that data redundancy leads to data anomalies, which can destroy the effectiveness of the database. You also learned that the relational database makes it possible to control data redundancies by using common attributes that are shared by tables, called foreign keys.

The proper use of foreign keys is crucial to controlling data redundancy, although they do not totally eliminate the problem because the foreign key values can be repeated many times. However, the

proper use of foreign keys minimizes data redundancies and the chances that destructive data anomalies will develop.

As important as it is to control data redundancy, sometimes the level of data redundancy must actually be increased to make the database serve crucial information purposes.

Data redundancies sometimes seem to exist to preserve the historical accuracy of the data. For example, consider a small invoicing system. The system includes the CUSTOMER, who may buy one or more PRODUCTs, thus generating an INVOICE. Because a customer may buy more than one product at a time, an invoice may contain several invoice LINEs, each providing details about the purchased product. The PRODUCT table should contain the product price to provide a consistent pricing input for each product that appears on the invoice. The tables that are part of such a system are shown in Figure 3.29. The system's relational diagram is shown in Figure 3.30.

As you examine the tables and relationships in the two figures, note that you can keep track of typical sales information. For example, by tracing the relationships among the four tables, you discover that customer 10014 (Myron Orlando) bought two items on March 8, 2018, that were written to invoice number 1001: one Houselite chain saw with a 16-inch bar and three rat-tail files. In other words, trace the CUS_CODE number 10014 in the CUSTOMER table to the matching CUS_CODE value in the INVOICE table. Next, trace the INV_NUMBER 1001 to the first two rows in the LINE table. Finally, match the two PROD_CODE values in LINE with the PROD_CODE values in PRODUCT. Application software will be used to write the correct bill by multiplying each invoice line item's LINE_UNITS by its LINE_PRICE, adding the results, and applying appropriate taxes. Later, other application software might use the same technique to write sales reports that track and compare sales by week, month, or year.

FIGURE 3.29 A SMALL INVOICING SYSTEM

Table name: CUSTOMER
Primary key: CUS_CODE
Foreign key: none

Database name: Ch03_SaleCo

CUS_CODE	CUS_LNAME	CUS_FNAME	CUS_INITIAL	CUS_AREACODE	CUS_PHONE
10010	Ramas	Alfred	A	615	844-2573
10011	Dunne	Leona	K	713	894-1238
10012	Smith	Kathy	W	615	894-2285
10013	Olowski	Paul	F	615	894-2180
10014	Orlando	Myron		615	222-1672
10015	O'Brian	Amy	B	713	442-3381
10016	Brown	James	G	615	297-1228
10017	Williams	George		615	290-2556
10018	Farriss	Anne	G	713	382-7185
10019	Smith	Olette	K	615	297-3809

Table name: INVOICE
Primary key: INV_NUMBER
Foreign key: CUS_CODE

INV_NUMBER	CUS_CODE	INV_DATE
1001	10014	08-Mar-18
1002	10011	08-Mar-18
1003	10012	08-Mar-18
1004	10011	09-Mar-18

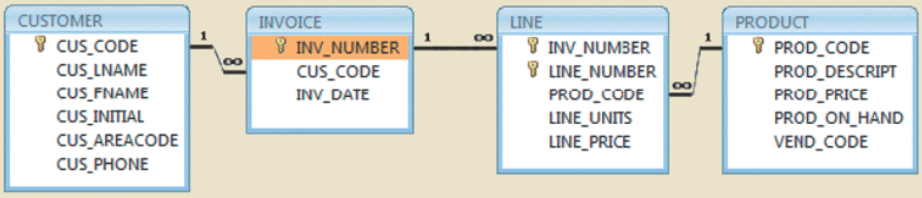
Table name: LINE
Primary key: INV_NUMBER + LINE_NUMBER
Foreign key: INV_NUMBER, PROD_CODE

INV_NUMBER	LINE_NUMBER	PROD_CODE	LINE_UNITS	LINE_PRICE
1001	1	123-21UUY	1	189.99
1001	2	SRE-657UG	3	2.99
1002	1	QER-34256	2	18.63
1003	1	ZZX/3245Q	1	6.79
1003	2	SRE-657UG	1	2.99
1003	3	001278-AB	1	12.95
1004	1	001278-AB	1	12.95
1004	2	SRE-657UG	2	2.99

Table name: PRODUCT
Primary key: PROD_CODE
Foreign key: none

PROD_CODE	PROD_DESCRIPT	PROD_PRICE	PROD_ON_HAND	VEND_CODE
001278-AB	Claw hammer	12.95	23	232
123-21UUY	Houselite chain saw, 16-in. bar	189.99	4	235
QER-34256	Sledge hammer, 16-lb. head	18.63	6	231
SRE-657UG	Rat-tail file	2.99	15	232
ZZX/3245Q	Steel tape, 12-ft. length	6.79	8	235

FIGURE 3.30 THE RELATIONAL DIAGRAM FOR THE INVOICING SYSTEM



As you examine the sales transactions in Figure 3.29, you might reasonably suppose that the product price billed to the customer is derived from the PRODUCT table because the product data is stored there. But why does that same product price occur again in the LINE table? Is that not a data redundancy? It certainly appears to be, but this time, the apparent redundancy is crucial to the system's success. Copying the product price from the PRODUCT table to the LINE table maintains the historical accuracy of the transactions. Suppose, for instance, that you fail to write the LINE_PRICE in the LINE table and that you use the PROD_PRICE from the PRODUCT table to calculate the sales revenue. Now suppose that the PRODUCT table's PROD_PRICE changes, as prices frequently do. This price change will be properly reflected in all subsequent sales revenue calculations. However, the calculations of past sales revenues will also reflect the new product price, which was not in effect when the transaction took place! As a result, the revenue calculations for all past transactions will be incorrect, thus eliminating the possibility of making proper sales comparisons over time. On the other hand, if the price data is copied from the PRODUCT table and stored with the transaction in the LINE table, that price will always accurately reflect the transaction that took place at that time. You will discover that such planned "redundancies" are common in good database design.

Finally, you might wonder why the LINE_NUMBER attribute was used in the LINE table in Figure 3.29. Wouldn't the combination of INV_NUMBER and PROD_CODE be a sufficient composite primary key—and, therefore, isn't the LINE_NUMBER redundant? Yes, it is, but this redundancy is common practice on invoicing software that typically generates such line numbers automatically. In this case, the redundancy is not necessary, but given its automatic generation, the redundancy is not a source of anomalies. The inclusion of LINE_NUMBER also adds another benefit: the order of the retrieved invoicing data will always match the order in which the data was entered. If product codes are used as part of the primary key, indexing will arrange those product codes as soon as the invoice is completed and the data is stored. You can imagine the potential confusion when a customer calls and says, "The second item on my invoice has an incorrect price," and you are looking at an invoice whose lines show a different order from those on the customer's copy!

4.7. Indexes

Suppose you want to locate a book in a library. Does it make sense to look through every book until you find the one you want? Of course not; you use the library's catalog, which is indexed by title, topic, and author. The index (in either a manual or computer library catalog) points you to the book's location, making retrieval a quick and simple matter. An index is an orderly arrangement used to logically access rows in a table.

Or, suppose you want to find a topic in this book, such as ER model. Does it make sense to read through every page until you stumble across the topic? Of course not; it is much simpler to go to the book's index, look up the phrase ER model, and read the references that point you to the appropriate page(s). In each case, an index is used to locate a needed item quickly.

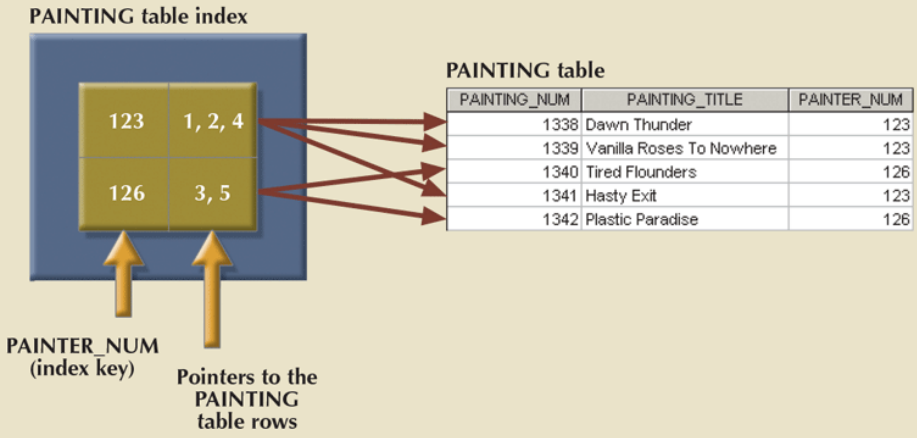
Indexes in the relational database environment work like the indexes described in the preceding paragraphs. From a conceptual point of view, an index is composed of an index key and a set of pointers. The index key is, in effect, the index's reference point. More formally, an index is an ordered arrangement of keys and pointers. Each key points to the location of the data identified by the key.

For example, suppose you want to look up all of the paintings created by a given painter in the Ch03_Museum database in Figure 3.18. Without an index, you must read each row in the PAINTING table and see if the PAINTER_NUM matches the requested painter. However, if you index the PAINTER table and use the index key PAINTER_NUM, you merely need to look up the appropriate PAINTER_NUM in the index and find the matching pointers. Conceptually speaking, the index would resemble the presentation in Figure 3.31.

As you examine Figure 3.31, note that the first PAINTER_NUM index key value (123) is found in records 1, 2, and 4 of the PAINTING table. The second PAINTER_NUM index key value (126) is found in records 3 and 5 of the PAINTING table.

DBMSs use indexes for many different purposes. You just learned that an index can be used to retrieve data more efficiently, but indexes can also be used by a DBMS to retrieve data ordered by a specific attribute or attributes. For example, creating an index on a customer's last name will allow you to retrieve the customer data alphabetically by the customer's last name. Also, an index key can be composed of one or more attributes. For example, in Figure 3.29, you can create an index on VEND_CODE and PROD_CODE to retrieve all rows in the PRODUCT table ordered by vendor and, within vendor, ordered by product.

FIGURE 3.31 COMPONENTS OF AN INDEX



Indexes play an important role in DBMSs for the implementation of primary keys. When you define a table's primary key, the DBMS automatically creates a unique index on the primary key column(s) you declared.

4.8. Codd's Relational Database Rules

In 1985, Dr. E. F. Codd published a list of 12 rules to define a relational database system. He published the list out of concern that many vendors were marketing products as "relational" even though those products did not meet minimum relational standards. Dr. Codd's list, shown in Table 3.8, is a frame of reference for what a truly relational database should be. Bear in mind that even the dominant database vendors do not fully support all 12 rules.

TABLE 13.8

DR. CODD'S 12 RELATIONAL DATABASE RULES

RULE	RULE NAME	DESCRIPTION
1	Information	All information in a relational database must be logically represented as column values in rows within tables.
2	Guaranteed access	Every value in a table is guaranteed to be accessible through a combination of table name, primary key value, and column name.
3	Systematic treatment of nulls	Nulls must be represented and treated in a systematic way, independent of data type.
4	Dynamic online catalog based on the relational model	The metadata must be stored and managed as ordinary data—that is, in tables within the database; such data must be available to authorized users using the standard database relational language.
5	Comprehensive data sublanguage	The relational database may support many languages; however, it must support one well-defined, declarative language as well as data definition, view definition, data manipulation (interactive and by program), integrity constraints, authorization, and transaction management (begin, commit, and rollback).
6	View updating	Any view that is theoretically updatable must be updatable through the system.
7	High-level insert, update, and delete	The database must support set-level inserts, updates, and deletes.
8	Physical data independence	Application programs and ad hoc facilities are logically unaffected when physical access methods or storage structures are changed.
9	Logical data independence	Application programs and ad hoc facilities are logically unaffected when changes are made to the table structures that preserve the original table values (changing order of columns or inserting columns).
10	Integrity independence	All relational integrity constraints must be definable in the relational language and stored in the system catalog, not at the application level.
11	Distribution independence	The end users and application programs are unaware of and unaffected by the data location (distributed vs. local databases).
12	Nonsubversion	If the system supports low-level access to the data, users must not be allowed to bypass the integrity rules of the database.
13	Rule zero	All preceding rules are based on the notion that to be considered relational, a database must use its relational facilities exclusively for management.