

3. Terminology

“When I use a word,” Humpty Dumpty said in rather a scornful tone, “it means just what I choose it to mean—neither more nor less.”

—LEWIS CARROLL *THROUGH THE LOOKING GLASS*

TOPICS COVERED IN THIS CHAPTER

Why This Terminology Is Important

Value-Related Terms

Structure-Related Terms

Relationship-Related Terms

Integrity-Related Terms

Summary

Review Questions

The terms in this chapter are important for you to understand before you embark upon learning the design process. Indeed, there are other terms that you’ll need to learn, and I’ll cover them as you work through the process. There’s also a glossary in the back of the book that you can use to refresh your memory on any term you learn here or in the following chapters.

WHY THIS TERMINOLOGY IS IMPORTANT

Relational database design has its own unique set of terms, just as any other profession, trade, or discipline. Here are three good reasons why it’s important for you to learn these terms.

- 1.** *They are used to express and define the special ideas and concepts of the relational database model.* Much of the terminology is derived from the mathematical branches of set theory and first-order predicate logic, which form the basis of the relational database model.
- 2.** *They are used to express and define the database design process itself.* The design process becomes clearer and much easier to understand once you know these terms.
- 3.** *They are used anywhere a relational database or RDBMS is discussed.* You’ll see these terms in publications such as trade magazines, software manuals, educational course materials, commercial database software books, and database-related web sites. This chapter covers a majority of the terms that define the ideas and concepts of the design process, including definitions and somewhat detailed discussions for each term. (I provide pertinent details or necessary further discussion for a given term at the point where the term is expressly used within a specific technique in the design process.) There are several other terms that I introduce and discuss later in the book because I think you’ll more easily understand them within the context of the specific idea or concept to which they relate.

Note

The glossary contains concise definitions for all of the terms in this chapter and throughout the book.

There are four categories of terms defined in this chapter: value-related, structure-related, relationship-related, and integrity-related.

VALUE-RELATED TERMS

Data

The values you store in the database are *data*. Data is static in the sense that it remains in the same state until you modify it by some manual or automated process. Figure 3.1 shows some sample data.

George Edleman 92883 05/16/96 95.00


Figure 3.1. An example of basic data

This data is meaningless at this point. For example, there is no easy way for you to determine what “92883” represents. Is it a zip code? Is it a part number? Even if you know it represents a customer identification number, is it one that is associated with George Edleman? There’s just no way of knowing until you process the data.

Information

Information is data that you process in a manner that makes it meaningful and useful to you when you work with it or view it. It is dynamic in the sense that it constantly changes relative to the data stored in the database, and also in the sense that you can process it and present it in an unlimited number of ways. You can show information as the result of a SQL `SELECT` statement, display it in a form on your computer screen, or print it as a report. The point to remember is that *you must process your data in some manner so that you can turn it into meaningful information*.

Figure 3.2 demonstrates how you might process and transform the data from the previous example into meaningful information. It has been manipulated in such a way—in this case as part of a patient invoice report—that it is now meaningful to anyone who views it.

 <p>Eastside Medical Clinic 7743 Kingman Dr. Seattle, WA 98032 (206) 555-9982</p>	<p>Patient Name: <u>George Edelman</u> Patient ID: <u>10884</u> Visit Date: <u>02/16/12</u> Physician: <u>Daniel Chavez</u></p>
--	--

Doctors Services	Service Code	Fee	Nursing Services	Service Code	Fee
<input checked="" type="checkbox"/> Consultation	<u>92883</u>	119.00	<input type="checkbox"/> R.N. Exam	89327	
<input checked="" type="checkbox"/> EKG	92773	<u>95.00</u>	<input type="checkbox"/> Supplies	82372	
<input type="checkbox"/> Physical	98377		<input type="checkbox"/> Nurse Instruction	88332	
<input type="checkbox"/> Ultrasound	97399		<input type="checkbox"/> Insurance Report	81368	

Figure 3.2. An example of data transformed into information

It is very important that you understand the difference between *data* and *information*. You design a database to provide meaningful information to someone within a business or organization. This information is available only if the appropriate *data* exists in the database and the database is structured in such a way as to support that *information*. If you ever forget the difference between data and information, just remember this little axiom:

Data is what you *store*; *information* is what you *retrieve*.

When you fully understand this single, simple concept, the logic behind the database design process will become crystal clear.

Note

Unfortunately, *data* and *information* are two terms that are *still* frequently used interchangeably (and, therefore, erroneously) throughout the database industry. You'll encounter this error in numerous trade magazines, commercial database books, and web sites, and you'll even see the terms misused by authors who should know better.

Null

A *null* represents a missing or unknown value. You must understand from the outset that a null *does not* represent a zero or a text string of one or more blank spaces. The reasons are quite simple.

- A zero can have a very wide variety of meanings. It can represent the state of an account balance, the current number of available first-class ticket upgrades, or the current stock level of a particular product.

- Although a text string of one or more blank spaces is guaranteed to be meaningless to most of us, it is definitely meaningful to a query language like SQL. A blank space is a valid character as far as SQL is concerned, and a character string composed of three blank spaces (' ') is just as legitimate as a character string composed of three letters ('abc'). In Figure 3.3, a blank represents the fact that Washington, D.C., is not located in any county whatsoever.

Clients

Client ID	Client First Name	Client Last Name	Client City	Client County	State	<< other fields >>
9001	Stewart	Jameson	Seattle	King	WA
9002	Susan	Black	Poulsbo		WA
9003	Estela	Rosales	Fremont	Alameda	CA
9004	Timothy	Ennis	Bellevue	King	WA
9005	Marvin	Russo	Washington		DC
9006	Kira	Bently	Portland		OR

Figure 3.3. An example of a table containing null values

- A *zero-length* string—two consecutive single quotes with no space in between ('')—is also an acceptable value to languages such as SQL, and can be meaningful under certain circumstances. In an EMPLOYEES table, for example, a zero-length string value in a field called MIDDLE INITIAL may represent the fact that a particular employee does not have a middle initial in his name.

Note

Due to space restrictions, I cannot always show all of the fields for a given sample table. I will, however, show the fields that are most relevant to the discussion at hand and use <<other fields>> to represent fields that are unessential to the example. You'll see this convention in many examples throughout the remainder of the book.

The Value of Nulls

A null is quite useful when you use it for its stated purpose, and the CLIENTS table in Figure 3.3 clearly illustrates this. Each null in the CLIENT COUNTY field represents a missing or unknown county name for the record in which it appears. In order for you to use nulls correctly, you must first understand why they occur at all.

Missing values are commonly the result of human error. For example, consider the record for Shannon Black. If you're entering the data for Ms. Black and you fail to ask her for the name of the county she lives in, that data is considered missing and is represented in the record as a null. Once you recognize the error, however, you can correct it by calling Ms. Black and asking her for the county name.

Unknown values appear in a table for a variety of reasons. One reason may be that a specific value you need for a field is as yet undefined. For instance, you could have a CATEGORIES table in a *School Scheduling* database that doesn't currently contain a category for a new set of classes that you want to offer beginning in the fall session.

Another reason a table might contain unknown values is that they are truly unknown. Refer to the CLIENTS table in Figure 3.3 once again and consider the record for Marvin Russo. Say that you're entering the data for Mr. Russo and you ask him for the name of the county he lives in. If he doesn't know the county name and you don't happen to know the county that includes the city in which he lives, then the value for the county field in his record is truly unknown and is represented within the record as a null. Obviously, you can correct the problem once either of you determines the correct county name.

A field value may also be null if none of its values applies to a particular record. Assume for a moment that you're working with an EMPLOYEES table that contains a SALARY field and an HOURLY RATE field. The value for one of these two columns is always going to be null because an employee cannot be paid both a fixed salary and an hourly rate.

It's important to note that there is a very slim difference between "does not apply" and "is not applicable." In the previous example, the value of one of the two fields literally does not apply. Now assume you're working with a PATIENTS table that contains a field called HAIR COLOR and you're currently updating a record for an existing male patient. If that patient recently became bald, then the value for that field is definitely "not applicable." Although you could just use a null to represent a value that is not applicable, I always recommend that you use a true value such as "N/A" or "Not Applicable." This will make the information clearer in the long run.

As you can see, whether you allow nulls in a table depends on the manner in which you're using the data. Now that you've seen the positive side of using nulls, let's take a look at the negative implication of using them.

The Problem with Nulls

The major disadvantage of nulls is that they have an adverse effect on mathematical operations. An operation involving a null evaluates to null. This is logically reasonable—if a number is unknown then the result of the operation is necessarily unknown. Note how a null alters the outcome of the operation in the following example:

$$(25 \times 3) + 4 = 79$$

$$(\text{Null} \times 3) + 4 = \text{Null}$$

$$(25 \times \text{Null}) + 4 = \text{Null}$$

$$(25 \times 3) + \text{Null} = \text{Null}$$

The PRODUCTS table in Figure 3.4 helps to illustrate the effects nulls have on mathematical expressions that incorporate fields from a table. In this case, the value for the TOTAL VALUE field is derived from the mathematical expression "[SRP] × [QTY ON HAND]." As you inspect the records in this table, note that the value for the TOTAL VALUE field is missing where the QTY ON HAND value is null, resulting in a null value for the TOTAL VALUE field as well. This leads to a serious *undetected* error that occurs when all the values in the TOTAL VALUE field are added together: an inaccurate total. This error is "undetected" because an RDBMS program will not inherently alert

you of the error. The only way to avoid this problem is to ensure that the values for the QTY ON HAND field cannot be null.

Products

Product ID	Product Description	Category	SRP	Qty On Hand	Total Value
70001	Shur-Lok U-Lock	Accessories	75.00		
70002	SpeedRite Cyclecomputer		65.00	20	1,300.00
70003	SteelHead Microshell Helmet	Accessories	36.00	33	1,118.00
70004	SureStop 133-MB Brakes	Components	23.50	16	376.00
70005	Diablo ATM Mountain Bike	Bikes	1,200.00		
70006	UltraVision Helmet Mount Mirrors		7.45	10	74.50

Figure 3.4. The nulls in this table will have an effect on mathematical operations involving the table's fields.

Figure 3.5 helps to illustrate the effect nulls have on aggregate functions that incorporate the values of a given field in a table. The result of an aggregate function, such as COUNT(<fieldname>), will be null if it is based on a field that contains null values. The table in Figure 3.5 shows the results of a summary query that counts the total number of occurrences of each category in the PRODUCTS table in Figure 3.4. The value of the TOTAL OCCURRENCES field is the result of the function expression COUNT([CATEGORY]). Notice that the summary query shows “0” occurrences of an unspecified category, implying that each product has been assigned a category. This information is clearly inaccurate because there are two products in the PRODUCTS table that have not been assigned a category.

Category Summary

Category	Total Occurrences
	0
Accessories	2
Bikes	1
Components	1

Figure 3.5. Nulls affect the results of an aggregate function.

The issues of missing values, unknown values, and whether a value will be used in a mathematical expression or aggregate function are all taken into consideration in the database design process, and we will revisit and discuss these issues further in later chapters.

STRUCTURE-RELATED TERMS

Table

According to the relational model, data in a relational database is stored in *relations*, which are perceived by the user as *tables*. Each relation is composed of *tuples* (records) and *attributes* (fields). Figure 3.6 shows a typical table structure.

Clients

Client ID	Client First Name	Client Last Name	Client City	<< other fields >>
9001	Stewart	Jameson	Seattle
9002	Susan	Black	Poulsbo
9003	Estela	Rosales	Tacoma
9004	Timothy	Ennis	Seattle
9005	Marvin	Russo	Bellingham
9006	Kira	Bently	Tacoma

Records

Fields

Figure 3.6. A typical table structure

Tables are the chief structures in the database and each table always represents a single, specific subject. The logical order of records and fields within a table is of absolutely no importance, and every table contains at least one field—known as a *primary key*—that uniquely identifies each of its records. (In Figure 3.6, for example, CLIENT ID is the primary key of the CLIENTS table.) In fact, data in a relational database can exist independently of the way it is physically stored in the computer because of these last two table characteristics. This is great news for the user because he or she isn't required to know the physical location of a record in order to retrieve its data.

The subject that a given table represents can either be an *object* or an *event*. When the subject is an object, it means that the table represents something that is tangible, such as a person, place, or thing. Regardless of its type, every object has characteristics that you can store as data and then process as information in an almost infinite number of ways. Pilots, products, machines, students, buildings, and equipment are all examples of objects that a table can represent, and Figure 3.6 illustrates one of the most common examples of this type of table.

When the subject of a table is an event, it means that the table represents something that occurs at a given point in time having characteristics you wish to record. You can store these characteristics as data and then process the data as information in exactly the same manner as a table that represents some specific object. Examples of events you may need to record include judicial hearings, distributions of funds, lab test

results, and geological surveys. Figure 3.7 shows an example of a table representing an event that we all have experienced at one time or another—a doctor’s appointment.

Patient Visit

Patient ID	Visit Date	Visit Time	Physician	Blood Pressure	<< other fields >>
92001	02/14/12	10:30	Hernandez	120/80
97002	02/14/12	13:00	Black	112/74
99014	02/15/12	09:30	Rolson	120/80
96105	02/15/12	11:00	Hernandez	160/90
96203	02/15/12	14:00	Hernandez	110/75
98003	02/16/12	09:30	Rolson	120/80

Figure 3.7. A table representing an event

A table that stores data used to supply information is called a *data table*, and it is the most common type of table in a relational database. Data in this type of table is dynamic because you can manipulate it (modify, delete, and so forth) and process it into information in some form or fashion. You’ll constantly interact with these types of tables as you work with your database.

A *validation table* (also known as a *lookup table*), on the other hand, stores data that you specifically use to implement data integrity. A validation table usually represents subjects, such as city names, skill categories, product codes, and project identification numbers. Data in this type of table is static because it will very rarely change at all. Although you have very little *direct* interaction with these tables, you’ll frequently use them *indirectly* to validate values that you enter into a data table. Figure 3.8 shows an example of a validation table.

Categories

Category ID	Category Name
10000	Accessories
20000	Bikes
30000	Clothing
40000	Components

Figure 3.8. An example of a validation table

I’ll discuss validation tables in more detail in Chapter 11, “Business Rules.”

Field

A *field* (known as an *attribute* in relational database theory) is the smallest structure in the database and it represents a characteristic of the subject of the table to which it belongs. Fields are the structures that actually store data. The data in these fields can then be retrieved and presented as information in almost any configuration that you can imagine. The quality of the information you get from your data is in direct proportion to the amount of time you’ve dedicated to ensuring the structural integrity and data integrity of the fields themselves. There is just no way to underestimate the importance of fields.

Every field in a *properly designed* database contains one and only one value, and its name will identify the type of value it holds. This makes entering data into a field very intuitive. If you see fields with names such as FIRSTNAME, LASTNAME, CITY, STATE, and ZIPCODE, you know exactly what type of values go into each field. You’ll also find it very easy to sort the data by state or look for everyone whose last name is “Hernandez.” You’ll typically encounter three other types of fields in an improperly or poorly designed database:

1. A *multipart* field (also known as a *composite* field), which contains two or more *distinct* items within its value
2. A *multivalued* field, which contains multiple instances of the *same* type of value
3. And a *calculated* field, which contains a concatenated text value or the result of a mathematical expression

Figure 3.9 shows a table with an example of each of these types of fields.

Client ID	Client First Name	Client Last Name	Client Full Name	Address	Client City, State, Zip	Account Rep
9001	Stewart	Jameson	Stewart Jameson	Seattle, WA 98125	John, Sandi
9002	Susan	Black	Susan Black	Poulsbo, WA 98370	Frits
9003	Estela	Rosales	Estela Rosales	Bellevue, WA 98005	John
9004	Timothy	Ennis	Timothy Ennis	Seattle, WA 98115	Frits, Sandi
9005	Marvin	Russo	Marvin Russo	Bellingham, WA 98225	Frits, John
9006	Kira	Bently	Kira Bently	Olympia, WA 98504	Sandi

Figure 3.9. A table containing regular, calculated, multipart, and multivalued fields

I’ll cover calculated, multipart, and multivalued fields in greater detail in Chapter 7, “Establishing Table Structures.”

Record

A *record* (known as a *tuple* in relational database theory) represents a unique instance of the subject of a table. It is composed of the entire set of fields in a table, regardless of whether the fields contain values.

Because of the manner in which a table is defined, each record is identified throughout the database by a unique value in the primary key field of that record.

In Figure 3.9, each record represents a unique client within the table, and the CLIENT ID field will identify a given client throughout the database. In turn, each record includes all of the fields within the table, and each field describes some aspect of the client represented by the record. Consider the record for Timothy Ennis, for example. His record represents a unique instance of the table's subject ("Clients") and includes the total collection of fields in the table, treated as a unit. The values of those fields represent relevant facts about Mr. Ennis that are important to someone in the organization.

Records are a key factor in understanding table relationships because you'll need to know how a record in one table relates to other records in another table.

View

A *view* is a "virtual" table composed of fields from one or more tables in the database; the tables that comprise the view are known as *base tables*. The relational model refers to a view as being "virtual" because it draws data from base tables rather than storing data on its own. In fact, the only information about a view that is stored in the database is its structure. Many major RDBMS programs support views, but some (such as Microsoft Access) refer to them as *saved queries*. Your specific RDBMS program will determine whether you refer to this object as a query or a view.

Views enable you to see the information in your database from many different aspects, providing you with a great amount of flexibility when you work with your data. You can create views in a variety of ways and they are especially useful when you base them on multiple related tables. In a school scheduling database, for example, you could create a view that consolidates data from the STUDENTS, CLASSES, and CLASS SCHEDULES tables.

Figure 3.10 shows a view called INSTRUMENT ASSIGNMENTS that is composed of fields taken from the STUDENTS, INSTRUMENTS, and STUDENT INSTRUMENTS tables. The view displays data that it draws from all of these tables simultaneously, based on matching values between the STUDENT ID fields in the STUDENTS and STUDENT INSTRUMENTS tables, and the INSTRUMENT ID fields in the INSTRUMENTS and STUDENT INSTRUMENTS tables.

Students

Student ID	Student First Name	Student Last Name	Student Phone	<< other fields >>
60001	Zachary	Erlich	553-3992
60002	Susan	Black	790-3992
60003	Joe	Rosales	551-4993

Student Instruments

Student ID	Instrument ID	Checkout Date
60002	1003	02/02/12
60001	1002	02/06/12
60003	1000	02/06/12

Instruments

Instrument ID	Instrument Description	Category	<< other fields >>
1000	Stratocaster	Guitar
1001	Player 2100 Multieffects	Multieffect Unit
1002	JCM 2000 Tube Super Lead	Amplifier
1003	Twin Reverb Reissue	Amplifier

Instrument Assignments (View)

Student First Name	Student Last Name	Instrument Description	Checkout Date
Zachary	Erlich	JCM 2000 Tube Super Lead	02/06/12
Susan	Black	Twin Reverb Reissue	02/03/12
Joe	Rosales	Stratocaster	02/06/12

Figure 3.10. An example of a typical view

There are three major reasons that views are important.

1. They enable you to work with data from multiple tables simultaneously. (In order for a view to do this, the tables must have connections, or *relationships*, to one another.)
2. They enable you to prevent certain users from viewing or manipulating specific fields within a table or group of tables. This capability can be very advantageous in terms of security.
3. You can use them to implement data integrity. A view you use for this purpose is known as *avalidation view*.

You'll learn more about designing and using views in Chapter 12, "Views."

Note

Although every major database vendor supports the type of view I've described in this section, several vendors are now supporting what is known as an *indexed* (or *materialized*) view. An indexed view is different from a "regular" view in that it does store data, and you can index its fields in order to improve the speed at which the RDBMS processes the view's data. A full discussion of indexed views is beyond the scope of this book because it is a vendor-specific implementation issue. However, you should research this topic further if you are working with RDBMS software such as Oracle, Microsoft SQL Server, IBM DB2, or Sybase SQL, or if you are working within a data warehouse scenario.

Keys

Keys are special fields that play very specific roles within a table, and the type of key determines its purpose within the table. There are several types of keys a table may contain, but the two most significant ones are the *primary key* and the *foreign key*.

A primary key is a field or group of fields that uniquely identifies each record within a table; a primary key composed of two or more fields is known as a *composite* primary key. The primary key is absolutely the most important key in the entire table.

- A primary key **value** identifies *a specific record* throughout the entire database.
- The primary key **field** identifies *a given table* throughout the entire database.
- The primary key enforces table-level integrity and helps establish relationships with other tables in the database. (You'll learn more about relationships in the next section.)

Every table in your database should have a primary key!

The AGENT ID field in Figure 3.11 is a good example of a primary key. It uniquely identifies each agent within the AGENTS table and helps to guarantee table-level integrity by ensuring nonduplicate records. It also establishes relationships between the AGENTS table and other tables in the database, as in the case with the ENTERTAINERS table shown in the example.

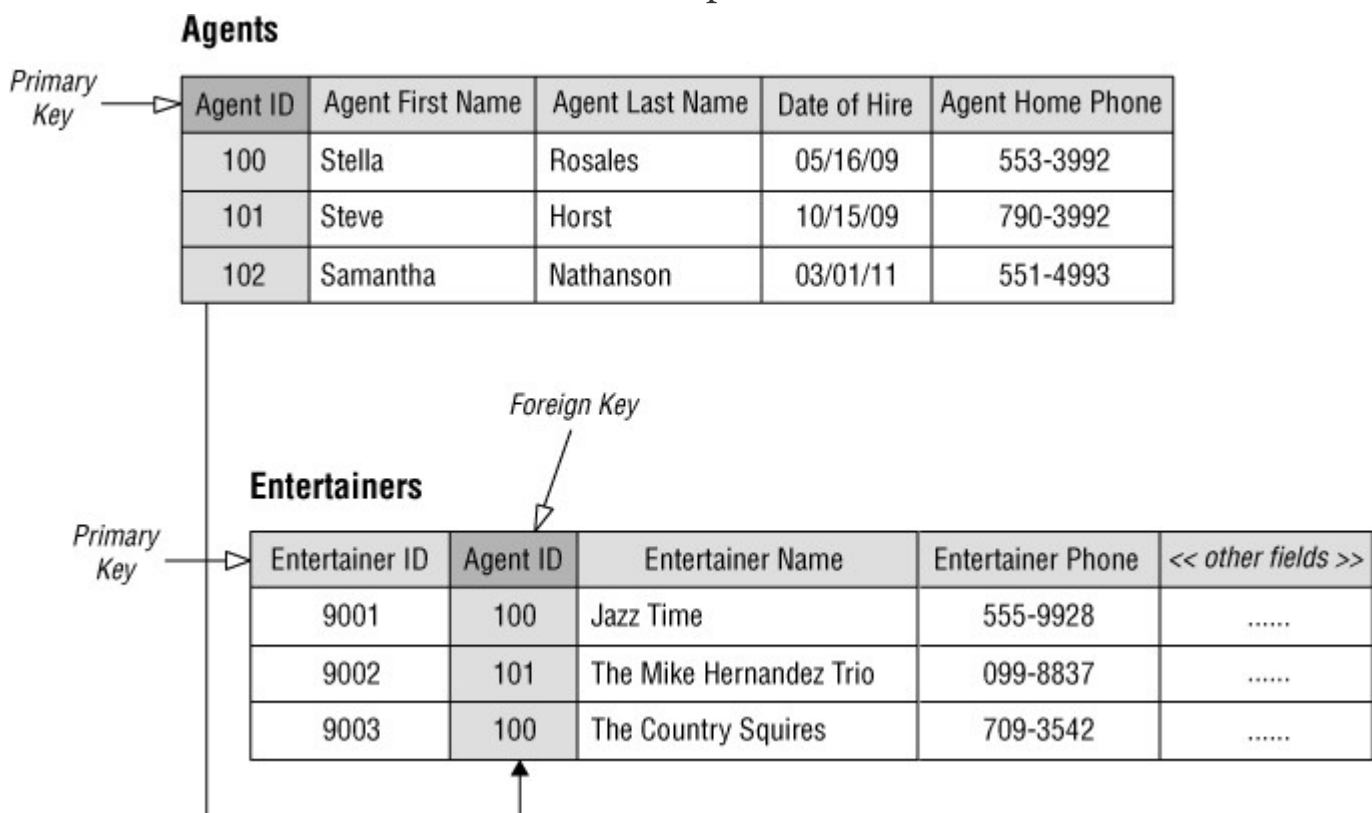


Figure 3.11. An example of primary and foreign key fields

When you determine that two tables bear a relationship to each other, you typically establish the relationship by taking a copy of the primary key from the first table and incorporating it into the structure of the second table, where it becomes a *foreign key*. The name “foreign key” is derived from the fact that the second table already has a

primary key of its own, and the primary key you are introducing from the first table is “foreign” to the second table.

Figure 3.11 also shows a good example of a foreign key. Note that AGENT ID is the primary key of the AGENTS table and a foreign key in the ENTERTAINERS table. AGENT ID assumes this role because the ENTERTAINERS table already has a primary key—ENTERTAINER ID. As such, AGENT ID establishes the relationship between both of the tables.

Besides helping to establish relationships between pairs of tables, foreign keys also help implement and ensure relationship-level integrity. This means that the records in both tables will always be properly related because the values of a foreign key *must match* existing values of the primary key to which it refers. Relationship-level integrity also helps you avoid the dreaded “orphaned” record, a classic example of which is an order record without an associated customer. If you don’t know who made the order, you can’t process it, and you obviously can’t invoice it. That’ll throw your quarterly sales off!

Key fields play an important part in a relational database, and you must learn how to create and use them. You’ll learn more about primary keys in Chapter 8, “Keys,” and Chapter 10, “Table Relationships.”

Index

An *index* is a structure that an RDBMS provides to improve data processing. Your particular RDBMS program will determine how the index works and how you use it. However, *an index has absolutely nothing to do with the logical database structure!* The only reason I include the term *index* in this chapter is that people often confuse it with the term *key*.

Index and *key* are just two more terms that are widely and frequently misused throughout the database industry and in numerous database-related publications and web sites. (Remember my comments on *data* and *information*?) You’ll always know the difference between the two if you remember that keys are *logical structures* you use to identify records within a table, and indexes are *physical structures* you use to optimize data processing.

RELATIONSHIP-RELATED TERMS

Relationships

A relationship exists between two tables when you can in some way associate the records of the first table with those of the second. You can establish the relationship via a set of primary and foreign keys (as you learned in the previous section) or through a third table known as a *linking table* (also known as an *associative table*). The manner in which you establish the relationship really depends on the type of relationship that exists between the tables. (You’ll learn more about that in a moment.) While Figure 3.11 illustrated a relationship established via primary/foreign keys, Figure 3.12 illustrates a relationship established with a linking table.

Students

Student ID	Student First Name	Student Last Name	Student Phone	<< other fields >>
60001	Zachary	Erlich	553-3992
60002	Susan	Black	790-3992
60003	Joe	Rosales	551-4993

Student Schedule (Linking Table)

Student ID	Class ID
60003	900001
60001	900003
60003	900003
60002	900002
60001	900001

Classes

Class ID	Class Name	Instructor ID	<< other fields >>
900001	Intro. to Political Science	220087
900002	Adv. Music Theory	220039
900003	American History	220148

Figure 3.12. A relationship established between two tables with the help of a linking table

A relationship is an important component of a relational database.

- It enables you to create multitable views.
- It is crucial to data integrity because it helps reduce redundant data and eliminate duplicate data.

You can characterize every relationship in three ways: by the type of relationship that exists between the tables, the manner in which each table participates, and the degree to which each table participates.

Types of Relationships

There are three specific types of relationship (traditionally known as a *cardinality*) that can exist between a pair of tables: *one-to-one*, *one-to-many*, and *many-to-many*.

One-to-One Relationships

A pair of tables bears a *one-to-one* relationship when a single record in the first table is related to only one record in the second table, and a single record in the second table is related to only one record in the first table. In this type of relationship, one table serves as a “parent” table and the other serves as a “child” table. You establish the relationship by taking a copy of the parent table’s primary key and incorporating it within the structure of the child table, where it becomes a foreign key. This is a special type of

relationship because it is the only one in which both tables may actually share the same primary key.

Figure 3.13 shows an example of a typical one-to-one relationship. In this case, EMPLOYEES is the parent table and COMPENSATION is the child table. The relationship between these tables is such that a single record in the EMPLOYEES table can be related to only one record in the COMPENSATION table, and a single record in the COMPENSATION table can be related to only one record in the EMPLOYEES table. Note that EMPLOYEE ID is indeed the primary key in both tables. However, it will also serve the role of a foreign key in the child table.

Employees

Employee ID	Employee First Name	Employee Last Name	Home Phone	<< other fields >>
100	Zachary	Erllich	553-3992
101	Susan	Black	790-3992
102	Joe	Rosales	551-4993

Compensation

Employee ID	Hourly Rate	Commission Rate	<< other fields >>
100	19.75	3.5%
101	25.00	5.0%
102	22.50	5.0%

Figure 3.13. An example of a one-to-one relationship

One-to-Many Relationships

A *one-to-many* relationship exists between a pair of tables when a single record in the first table can be related to many records in the second table, but a single record in the second table can be related to *only one* record in the first table. (The parent/child model I used to describe a one-to-one relationship works here as well. In this case, the table on the “one” side of the relationship is the parent table, and the table on the “many” side is the child table.) You establish a one-to-many relationship by taking a copy of the parent table’s primary key and incorporating it within the structure of the child table, where it becomes a foreign key.

The example in Figure 3.14 illustrates a typical one-to-many relationship. A single record in the AGENTS table can be related to one or more records in the ENTERTAINERS table, but a single record in the ENTERTAINERS table is related to only one record in the AGENTS table. As you probably have already guessed, AGENT ID is a foreign key in the ENTERTAINERS table.

Agents

Agent ID	Agent First Name	Agent Last Name	Date of Hire	Agent Home Phone
100	Stella	Rosales	05/16/09	553-3992
101	Steve	Horst	10/15/09	790-3992
102	Samantha	Nathanson	03/01/11	551-4993

Entertainers

Entertainer ID	Agent ID	Entertainer Name	Entertainer Phone	<< other fields >>
9001	101	Jazz Time	555-9928
9002	100	The Mike Hernandez Trio	959-8837
9003	100	The Country Squires	709-3542

Figure 3.14. An example of a one-to-many relationship

This is by far the most common relationship that exists between a pair of tables in a database. It is crucial from a data-integrity standpoint because it helps to eliminate duplicate data and to keep redundant data to an absolute minimum.

Many-to-Many Relationships

A pair of tables bears a *many-to-many* relationship when a single record in the first table can be related to *many* records in the second table and a single record in the second table can be related to *many* records in the first table. You establish this relationship with a *linking table*. (You learned a little bit about this type of table at the beginning of this section.) A linking table makes it easy for you to associate records from one table with those of the other and will help to ensure you have no problems adding, deleting, or modifying related data. You define a linking table by taking copies of the primary key of each table in the relationship and using them to form the structure of the new table. These fields actually serve two distinct roles: Together, they form the *composite primary key* of the linking table; separately, they each serve as a foreign key.

A many-to-many relationship that is not properly established is “unresolved.” Figure 3.15 shows a classic and clear example of an unresolved many-to-many relationship. In this instance, a single record in the STUDENTS table can be related to *many* records in the CLASSES table and a single record in the CLASSES table can be related to *many* records in the STUDENTS table.

Students

Student ID	Student First Name	Student Last Name	Student Phone	<< other fields >>
60001	Zachary	Erlich	553-3992
60002	Susan	Black	790-3992
60003	Joe	Rosales	551-4993

Classes

Class ID	Class Name	Instructor ID	<< other fields >>
900001	Intro. to Political Science	220087
900002	Adv. Music Theory	220039
900003	American History	220148

Figure 3.15. An example of an unresolved many-to-many relationship

This relationship is unresolved due to the inherent peculiarity of the many-to-many relationship. The main issue is this: How do you *easily* associate records from the first table with records in the second table? To reframe the question in terms of the tables shown in Figure 3.15, how do you associate a single student with several classes or a specific class with several students? Do you insert a few STUDENT fields into the CLASSES table? Or do you add several CLASS fields to the STUDENTS table? Either of these approaches will make it difficult for you to work with the data in those tables and will affect data integrity adversely. The best approach for you to take is to create and use a linking table, which will resolve the many-to-many relationship in the most appropriate and effective manner. Figure 3.16 shows this solution in practice.

Students

Student ID	Student First Name	Student Last Name	Student Phone	<< other fields >>
60001	Zachary	Erlich	553-3992
60002	Susan	Black	790-3992
60003	Joe	Rosales	551-4993

Student Schedule (Linking Table)

Student ID	Class ID
60003	900001
60001	900003
60003	900003
60002	900002
60001	900001

Classes

Class ID	Class Name	Instructor ID	<< other fields >>
900001	Intro. to Political Science	220087
900002	Adv. Music Theory	220039
900003	American History	220148

Figure 3.16. Resolving the many-to-many relationship with a linking table

It's important for you to know the type of relationship that exists between a pair of tables because it determines how the tables are related, whether or not records between the tables are interdependent, and the minimum and maximum number of related records that can exist within the relationship. You'll learn much more about relationships in Chapter 10, "Table Relationships."

Types of Participation

A table's participation within a relationship can be either *mandatory* or *optional*. Say there is a relationship between two tables called TABLE_A and TABLE_B.

- TABLE_A's participation is *mandatory* if you must enter *at least one* record into TABLE_A before you can enter records into TABLE_B.
- TABLE_A's participation is *optional* if you are not required to enter *any* records into TABLE_A before you can enter records into TABLE_B.

Let's take a look at an example using the AGENTS and CLIENTS tables in Figure 3.17. The AGENTS table has a *mandatory* participation within the relationship *if an agent must exist* before you can enter a new client into the CLIENTS table. However, the AGENTS table's participation is *optional if there is no requirement for an agent to exist* in the table before you enter a new client into the CLIENTS table. You can identify the appropriate type of participation for the AGENTS table by determining how you're going to use its data in relation to the data in the CLIENTS table. For example, when

you want to ensure that each client is assigned to an available agent, you make the AGENTS table's participation within the relationship mandatory.

Agents

Agent ID	Agent First Name	Agent Last Name	Date of Hire	Agent Home Phone
100	Stella	Rosales	05/16/09	553-3992
101	Steve	Horst	10/15/09	790-3992
102	Samantha	Nathanson	03/01/11	551-4993

Clients

Client ID	Agent ID	Client First Name	Client Last Name	Client Home Phone
9001	100	Stewart	Jameson	553-3992
9002	101	Susan	Black	790-3992
9003	102	Scott	Baker	551-4993

Figure 3.17. The AGENTS and CLIENTS tables

Degree of Participation

The *degree of participation* determines the minimum number of records that a given table must have associated with a single record in the related table and the maximum number of records that a given table is allowed to have associated with a single record in the related table.

Consider, once again, a relationship between two tables called TABLE_A and TABLE_B. You establish the degree of participation for TABLE_B by indicating a minimum and maximum number of records in TABLE_B that can be related to a *single* record in TABLE_A. If a single record in TABLE_A can be related to no less than one but no more than ten records in TABLE_B, then the degree of participation for TABLE_B is **1,10**. (The notation for the degree of participation shows the minimum number on the left and the maximum number on the right, separated by a comma.) You can establish the degree of participation for TABLE_A in the same manner. You can identify the degree of participation for each table in a relationship by determining the way the data in each table is related and how you're using the data.

Consider the AGENTS and CLIENTS tables in Figure 3.17 once more. If you require an agent to handle *at least one* client, but certainly no more than eight, then the degree of participation for the CLIENTS table is **1,8**. When you want to ensure that a client can only be assigned to one agent, then you indicate the degree of participation for the AGENTS table as **1,1**. You'll learn how to indicate the degree of participation for a given relationship in Chapter 10.

INTEGRITY-RELATED TERMS

Field Specification

A *field specification* (traditionally known as a *domain*) represents all the elements of a field. Each field specification incorporates three types of elements: *general*, *physical*, and *logical*.

- *General* elements constitute the most fundamental information about the field and include items such as Field Name, Description, and Parent Table.
- *Physical* elements determine how a field is built and how it is represented to the person using it. This category includes items such as Data Type, Length, and Display Format.
- *Logical* elements describe the values stored in a field and include items such as Required Value, Range of Values, and Default Value.

You'll learn all of the elements associated with a field specification, including those mentioned here, in Chapter 9, "Field Specifications."

Data Integrity

Data integrity refers to the validity, consistency, and accuracy of the data in a database. I cannot overstate the fact that the level of accuracy of the information you retrieve from the database is in direct proportion to the level of data integrity you impose upon the database. Data integrity is one of the most important aspects of the database design process, and you cannot underestimate, overlook, or even partially neglect it. To do so would put you at risk of being plagued by errors that are very hard to detect or identify. As a result, you would be making important decisions on information that is inaccurate at best, or totally invalid at worst.

There are four types of data integrity that you'll implement during the database design process. Three types of data integrity are based on various aspects of the database structure and are labeled according to the area (level) in which they operate. The fourth type of data integrity is based on the way an organization perceives and uses its data. The following is a brief description of each.

1. *Table-level integrity* (traditionally known as *entity integrity*) ensures that there are no duplicate records within the table and that the field that identifies each record within the table is unique and never null.
2. *Field-level integrity* (traditionally known as *domain integrity*) ensures that the structure of every field is sound; that the values in each field are valid, consistent, and accurate; and that fields of the same type (such as CITY fields) are consistently defined throughout the database.
3. *Relationship-level integrity* (traditionally known as *referential integrity*) ensures that the relationship between a pair of tables is sound and that the records in the tables are synchronized whenever data is entered into, updated in, or deleted from either table.
4. *Business rules* impose restrictions or limitations on certain aspects of a database based on the ways an organization perceives and uses its data. These restrictions can

affect aspects of database design, such as the range and types of values stored in a field, the type of participation and the degree of participation of each table within a relationship, and the type of synchronization used for relationship-level integrity in certain relationships. All of these restrictions are discussed in more detail in Chapter 11. Because business rules affect integrity, they must be considered along with the other three types of data integrity during the design process.

SUMMARY

This chapter began with an explanation of why terminology is important for defining, discussing, or reading about the relational database model and the database design process.

The section on *value-related terms* showed you that there is a distinct difference between data and information, and that understanding this difference is crucial to understanding the database design process. You now know quite a bit about *nulls* and how they affect information you retrieve from the database.

Structure-related terms were covered next, and you learned that the core structures of every relational database are *fields*, *records*, and *tables*. You now know that *views* are virtual tables that are used, in part, to work with data from two or more tables simultaneously. We then looked at *key fields*, which are used to identify records uniquely within a table and to establish a relationship between a pair of tables. Finally, you learned the difference between a *key field* and an *index*. Now you know that an index is strictly a software device used to optimize data processing.

In the section on *relationship-related terms*, you learned that a connection between a pair of tables is known as a *relationship*. A relationship is used to help ensure various aspects of data integrity, and it is the mechanism used by a view to draw data from multiple tables. You then learned about the three characteristics of table relationships: the *type of relationship* (one-to-one, one-to-many, many-to-many), the *type of participation* (optional or mandatory), and the *degree of participation* (minimum/maximum number of related records).

The chapter ended with a discussion of *integrity-related terms*. Here you learned that a *field specification* establishes the general, physical, and logical characteristics of a field—characteristics that are an integral part of every field in the database. You then learned that *data integrity* is one of the most important aspects of the database design process because of its positive effect on the data in the database. Also, you now know that there are four types of data integrity—three based on database structure and one based on the way the organization interprets and uses its data. These levels of integrity ensure the quality of your database's design and the accuracy of the information you retrieve from it.

REVIEW QUESTIONS

1. Why is terminology important?
2. Name the four categories of terms.
3. What is the difference between *data* and *information*?

- 4.** What does a *null* represent?
- 5.** What is a null's major disadvantage?
- 6.** What are the chief structures in the database?
- 7.** Name the three types of tables.
- 8.** What is a *view*?
- 9.** State the difference between a *key* and an *index*.
- 10.** What are the three types of *relationships* that can exist between a pair of tables?
- 11.** What are the three ways in which you can characterize a relationship?
- 12.** What is a *field specification*?
- 13.** What three types of elements does a field specification incorporate?
- 14.** What is *data integrity*?
- 15.** Name the four types of data integrity.