

Appendix A1

Designing Databases with Visio Professional 2010: A Tutorial

Microsoft Visio Professional is a powerful database design and modeling tool. The Visio software has so many features that it is impossible to demonstrate all of them in this short tutorial. However, you will learn how to:

- Start Visio Professional.
- Select the Crow's Foot entity relationship diagram (ERD) option.
- Create the entities and define their components.
- Create the relationships between the entities and define the nature of those relationships.
- Edit the Crow's Foot ERDs.
- Insert text into the design grid and format the text.

Preview

Once you have learned how to create a Visio Crow's Foot ERD, you will be sufficiently familiar with the basic Visio Professional software features to experiment on your own with other modeling and diagramming options. You will also learn how to insert text into the Visio diagram to document features you consider especially important or to simply provide an explanation of some segment of the ERD.

Note: The screens and instructions in this tutorial are for Microsoft Visio Professional 2010. If you are using an earlier version of Visio, your screen will vary slightly.

Data Files and Available Formats

[MS Access](#) [Oracle](#) [MS SQL](#) [My SQL](#)

[MS Access](#) [Oracle](#) [MS SQL](#) [My SQL](#)

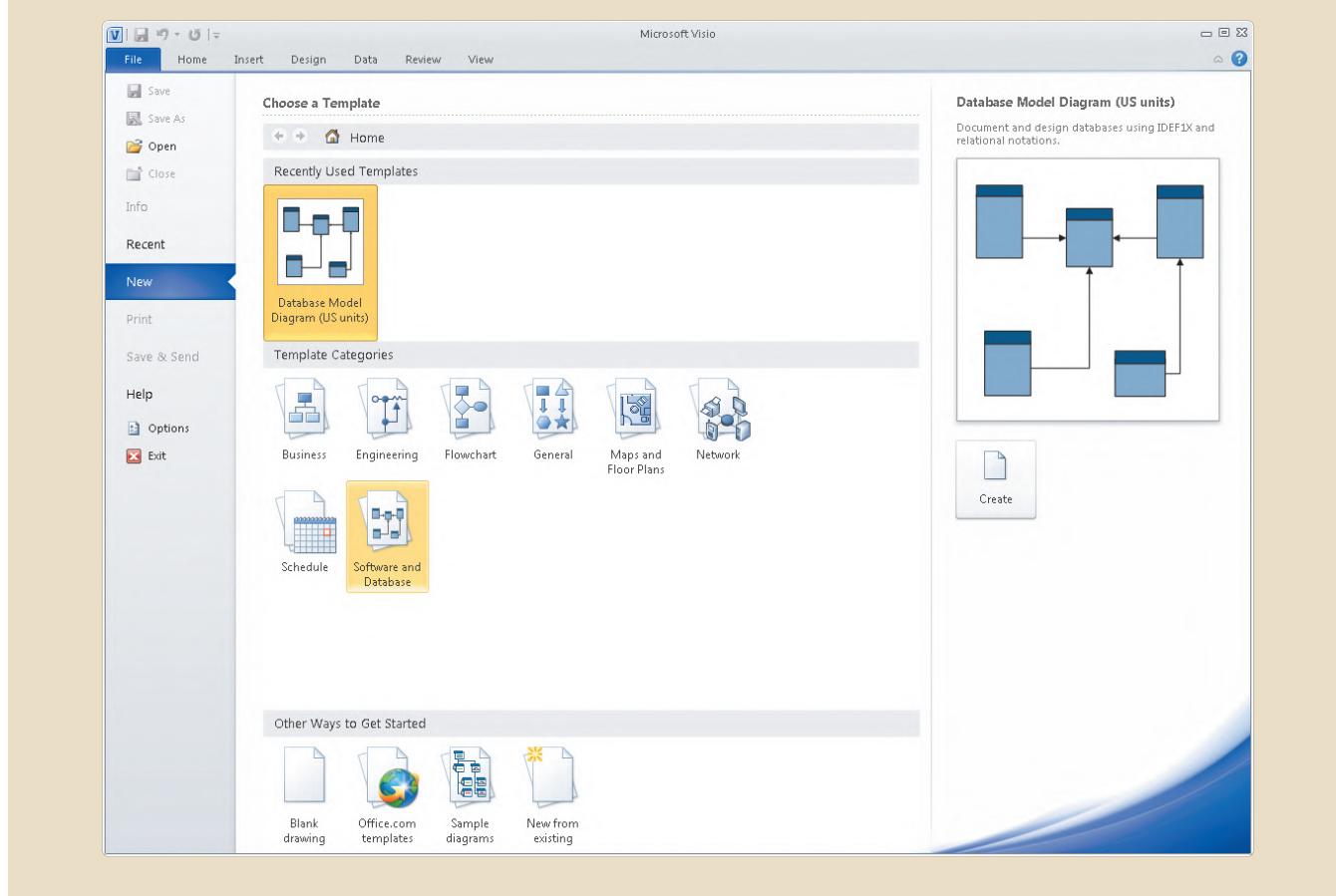
There are no data files for this appendix.

Data Files Available on cengagebrain.com

A1-1 Starting Visio Professional

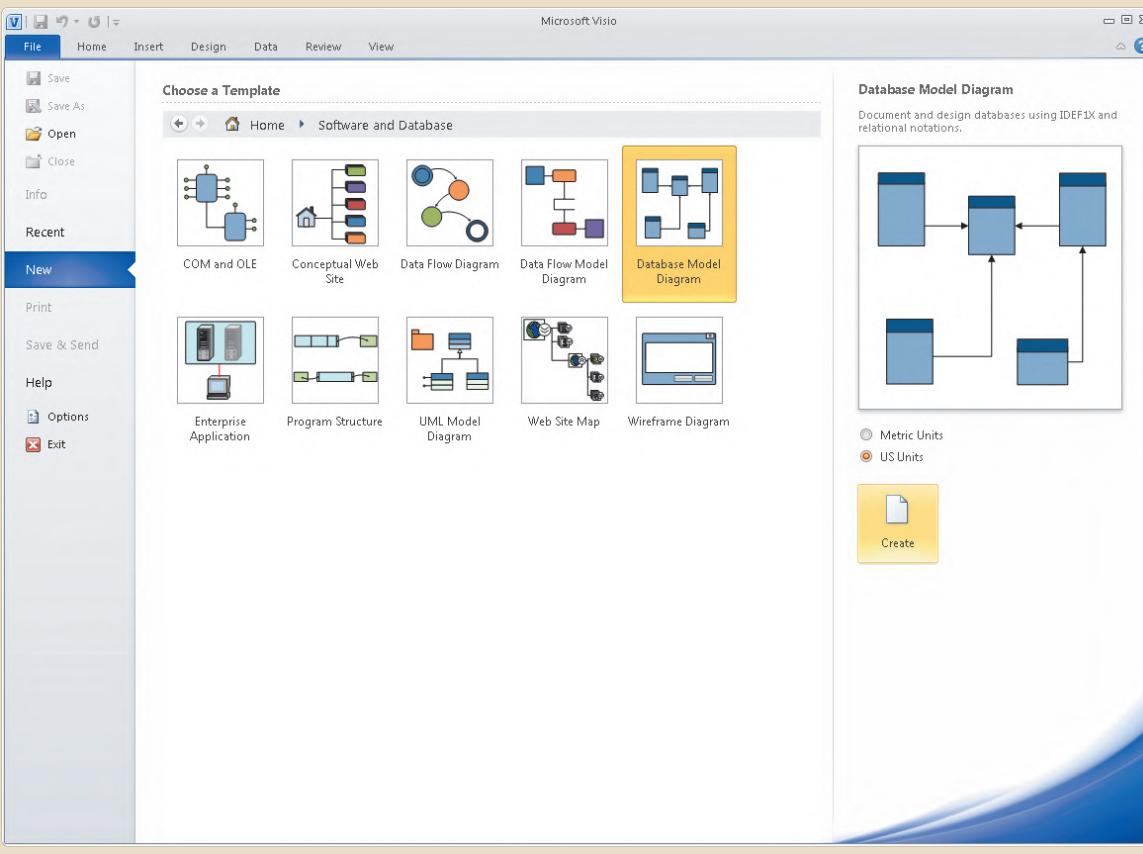
The typical Visio Professional software installation lets you select Visio through the **Start**, **(All) Programs**, **Microsoft Visio** sequence. After the Visio software has been activated, click the **Software and Database** option to match the screen shown in Figure A1.1. (Previously created Visio files show up in the **Recent Documents** header on the right side of the screen.)

FIGURE A1.1 THE VISIO PROFESSIONAL OPENING SCREEN



With the **Software and Database** selection shown in Figure A1.1, select the **Database Model Diagram** object. Note that your selection results in a highlight around the object. As shown in Figure A1.2, you will see the Database Modeling Template description in the right side of your screen.

FIGURE A1.2 THE DATABASE MODEL OBJECT SELECTION



Click the **Create** button on the right side of the screen as shown in Figure A1.2 to begin a new diagram, producing the screen shown in Figure A1.3. Because the preference here is for a larger grid than the one shown in Figure A1.3, start by selecting the **View** ribbon at the top of the screen. Click the **Zoom** option to generate the list of size options. Figure A1.4 shows that the **100%** option has been selected. When you click the **100%** selection and click **OK**, the grid expands to fill the screen.

A1-4 Appendix A1

FIGURE A1.3 THE DRAWING BOARD

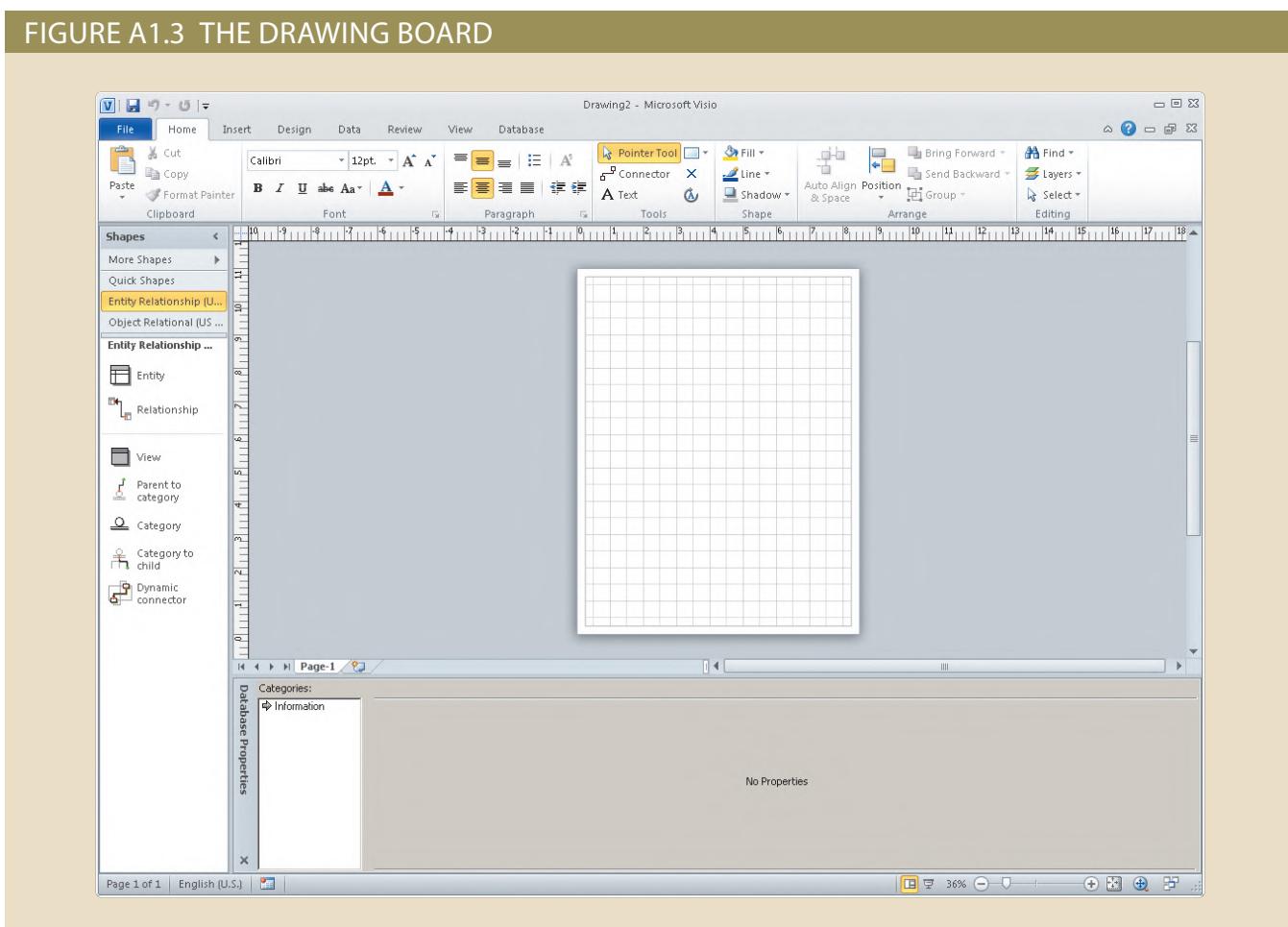
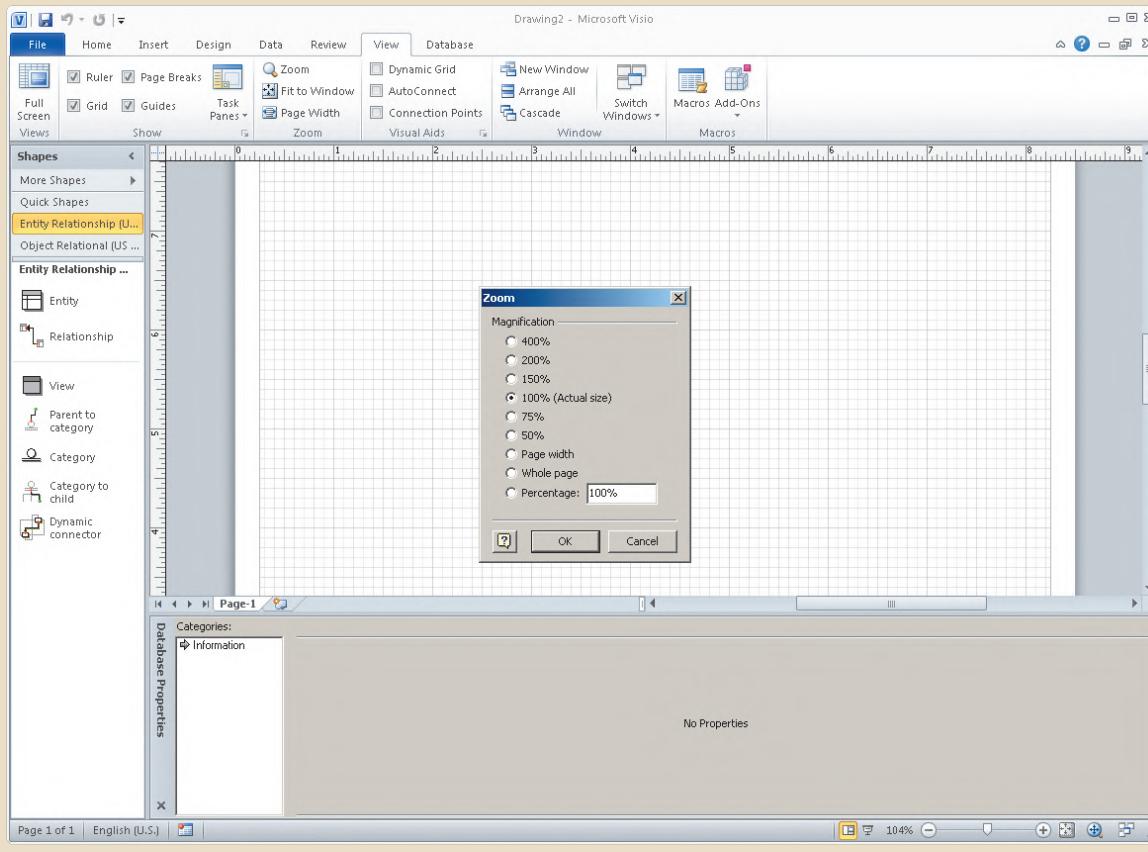


FIGURE A1.4 THE DRAWING BOARD SIZE OPTION



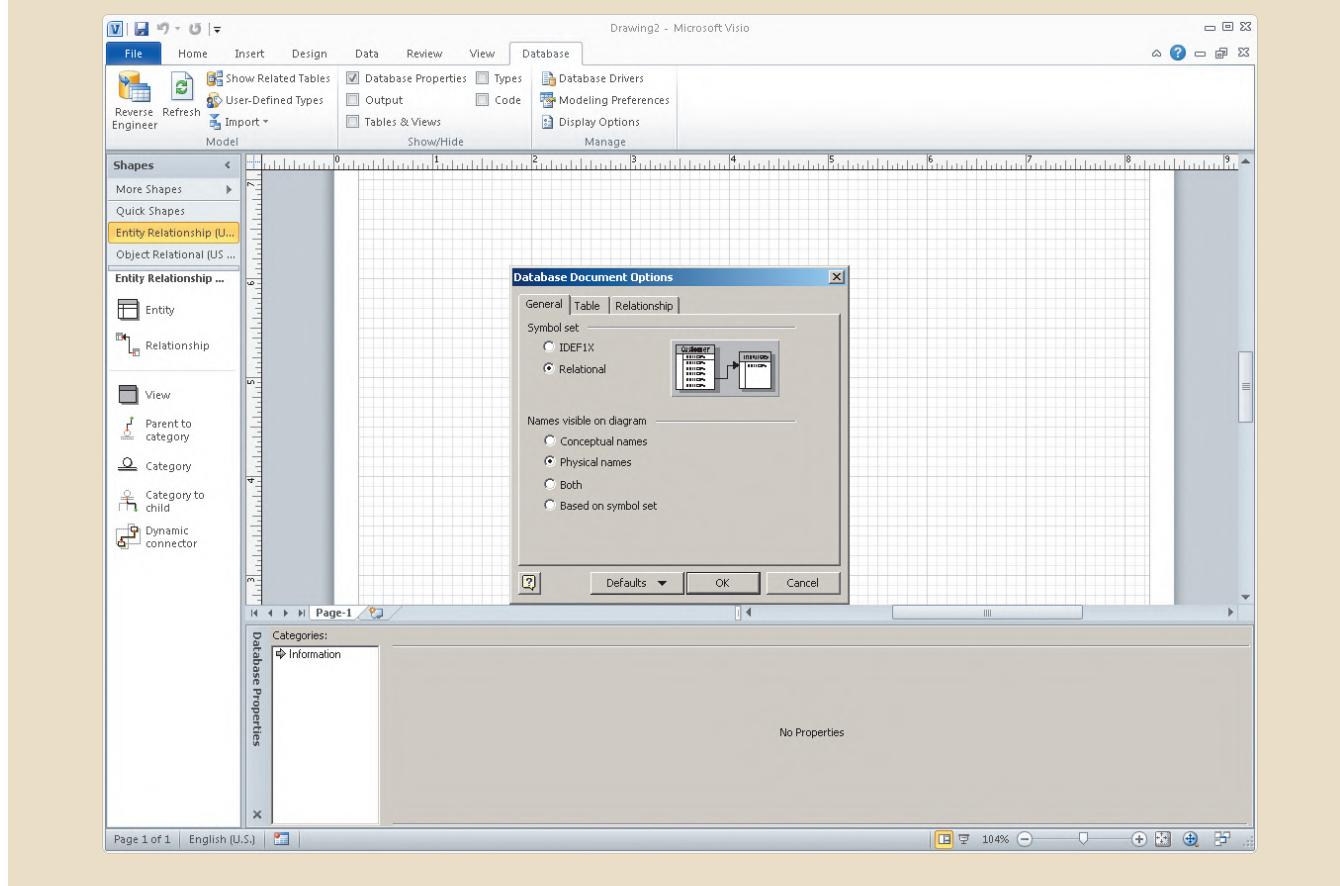
By selecting the Visio Professional database option and its drawing board, you have completed the preliminary work required to create ERDs. You are now ready to draw the ERDs on the drawing board. You will use the Crow's Foot option, the same one used to create all of the ERDs in this text.

A1-2 Setting the Stage for Creating a Crow's Foot ERD

To select the Crow's Foot option, select the **Database** ribbon, and then click **Display Options** to display the Database Document Options window shown in Figure A1.5. The default selection is the *General* tab. Note that the default selections in the *General* tab are *Relational* and *Physical names*. Ensure that you have these default options selected.

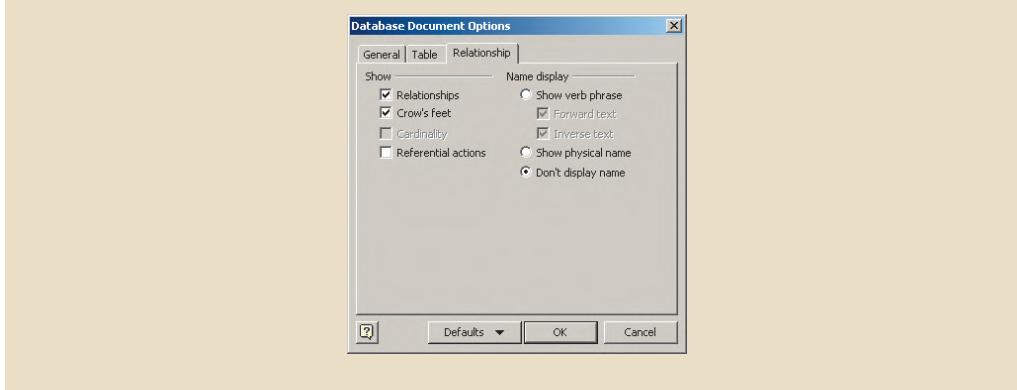
A1-6 Appendix A1

FIGURE A1.5 THE DATABASE DOCUMENT OPTIONS WINDOW



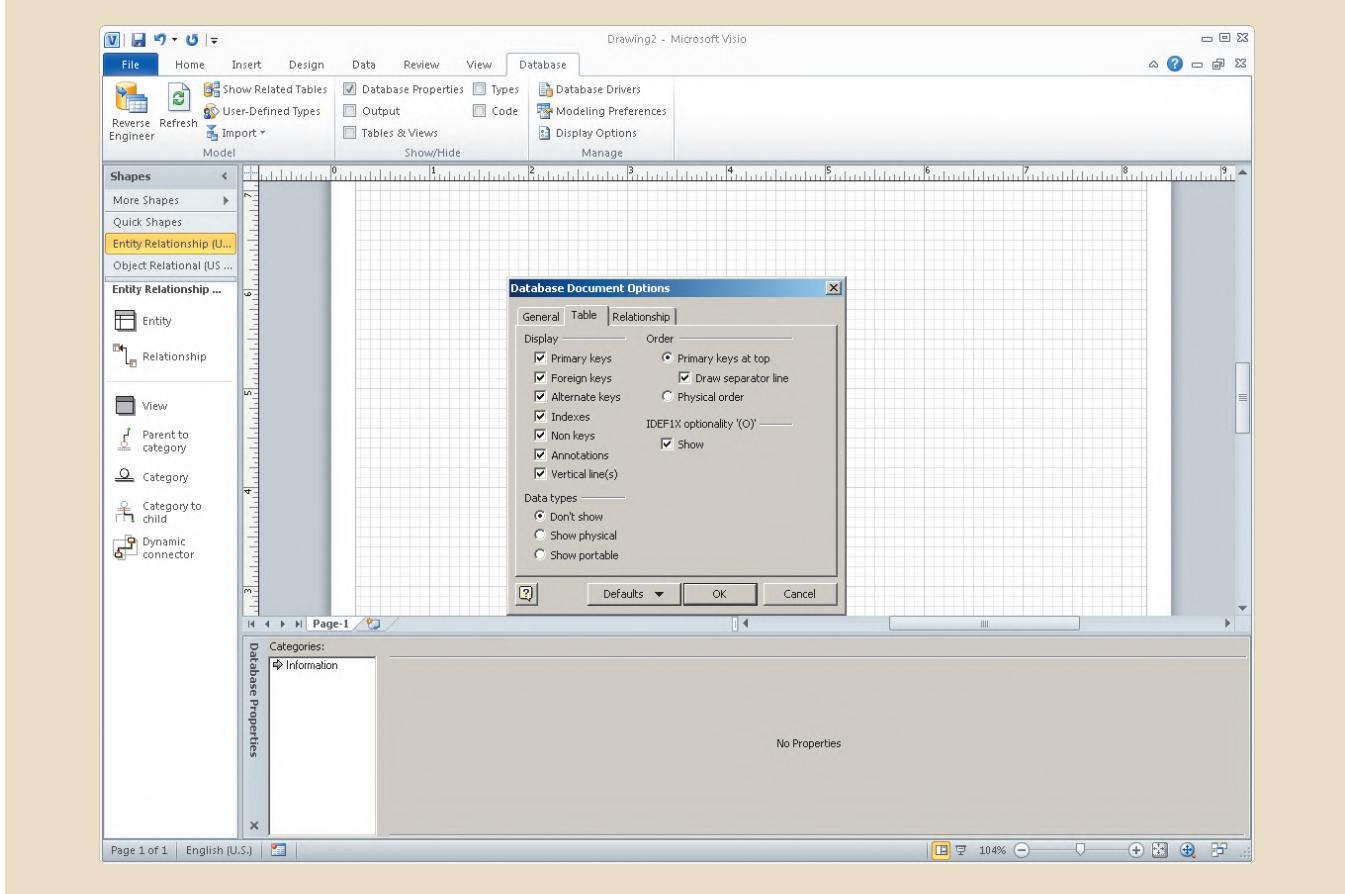
Select the **Relationship** tab and click to select the **Crow's feet** option as shown in Figure A1.6. As you examine the tab, keep in mind that we have not created and named any relationships yet. Therefore, we have not determined *how* the names will be displayed. You will return to this dialog box later to see the effect of name display options and to demonstrate that you can edit the displays when you are working on the relationships.

FIGURE A1.6 THE DATABASE DOCUMENT OPTIONS, RELATIONSHIP TAB



Next, select the **Table** tab in the Database Document Options dialog box, as shown in Figure A1.7. Make sure that the check boxes are marked as shown here, and then click the **OK** button to begin creating Crow's Foot ERDs.

FIGURE A1.7 THE DATABASE DOCUMENT OPTIONS, TABLE TAB



A1-2a The Business Rules

To illustrate the development of the Visio Professional's Crow's ERD, you will create a simple design based on the following business rules:

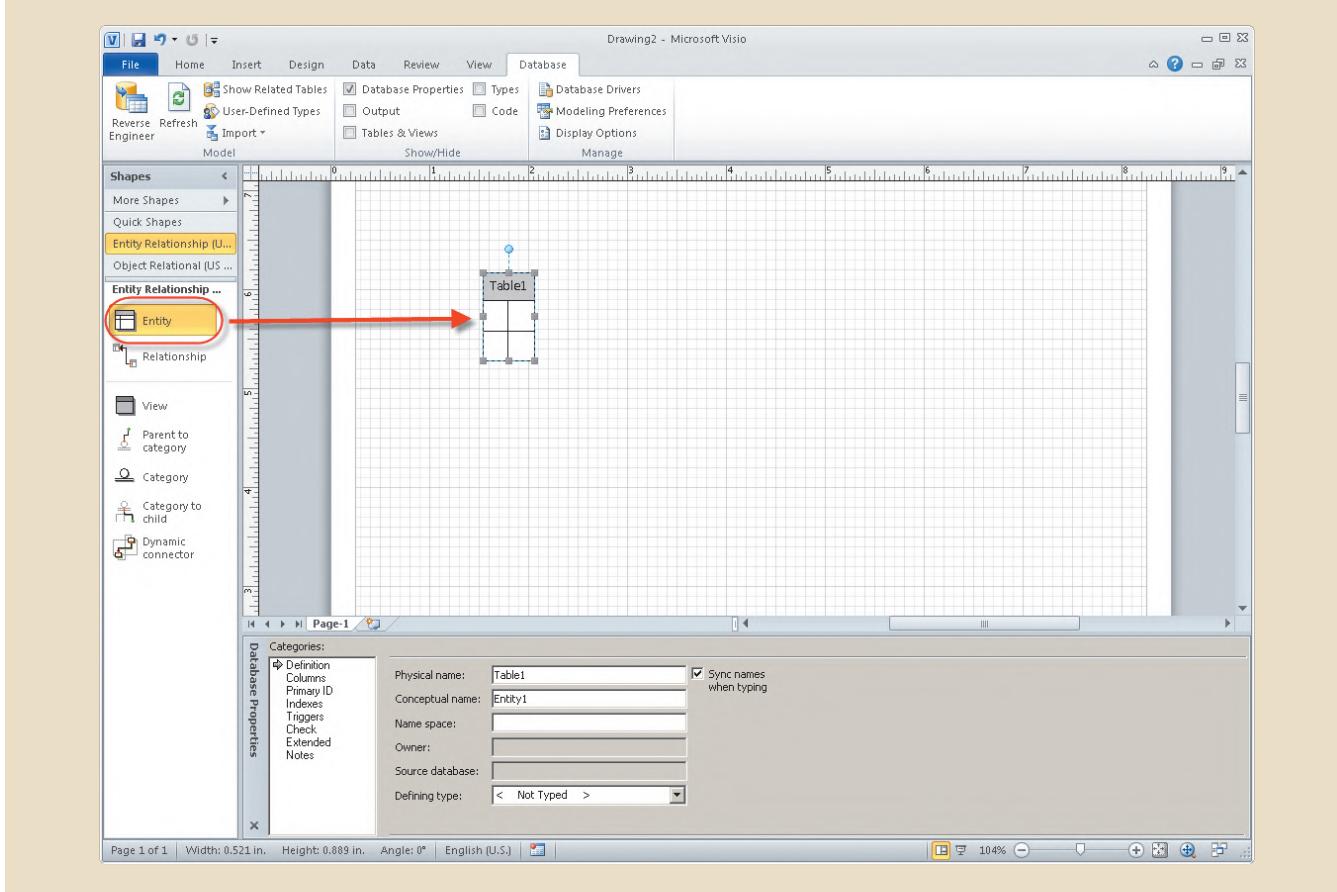
1. A course can generate many classes.
2. Each class is generated by a course.
3. A course may or may not generate a class.

Note that a class has been defined as a section of a course. That definition reflects the real world's use of the labels *class* and *course*. Students have a *class* schedule rather than a section schedule. The catalog that lists all of the courses offered by a department is called a *course catalog*. Some courses are not taught each semester, so they may not generate a class during any given semester.

A1-3 Creating an Entity

Now that you have some idea of the proposed design components, let's create the first entity for the design. Click the **Entity** object shown in Figure A1.8. (It is circled in the figure.) Drag the **Entity** object to the grid and then drop it. That action will produce the **Table1** object shown in the grid in Figure A1.8. (The **1** in the **Table1** label indicates that this is the first entity object to be placed on the grid.) Note that the entity object is shown as a table.

FIGURE A1.8 PLACING THE ENTITY OBJECT IN THE GRID



As you examine Figure A1.8, note that the small “locks” around the **Table1** object perimeter indicate that the **Table1** object has been selected. You can deselect the object by clicking an empty portion of the grid. If the **Table1** object has not been selected, click it to select it.

A1-3a The Database Properties Window

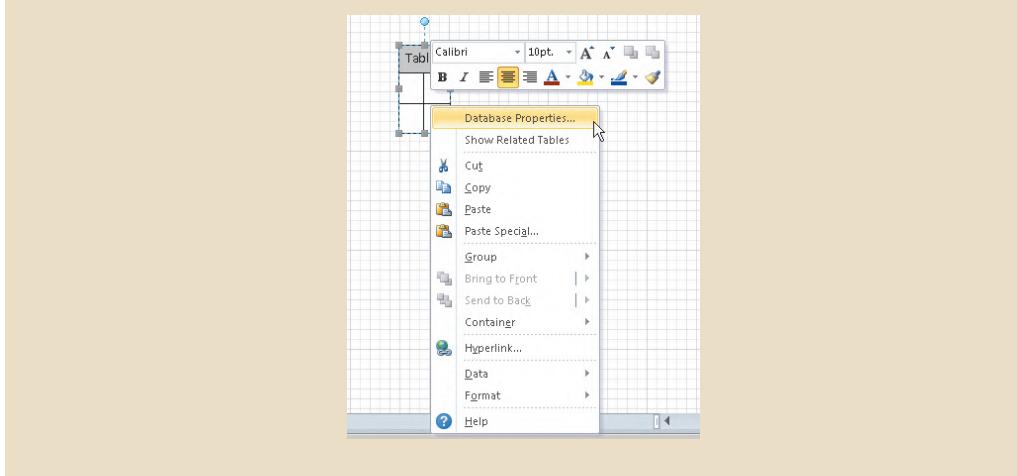
If the **Table1** object is selected, you will see the default **Database Properties** window at the bottom of the screen. (You will see later in this section that the window’s location and format may be changed to become the new default. However, you will start by using the standard default window shown in Figure A1.8.)

As you examine the **Database Properties** window in Figure A1.8, note the selection of the **Definition** option in the **Categories:** listing. (To select any option in the list, click it. The selection is indicated by the arrow to the left of the option. In this case, the arrow appears next to the **Definition** option.) At this point, the default **Table1** label shows up in the **Physical name:** slot.

A1-3b Creating the Default Database Properties Window

Depending on how you configured the Visio Professional software and/or on what operating system you use, you may not see the Database Properties window. If your screen does not show a default Database Properties window, right-click the **Table1** object in the grid to generate the **Database Properties...** option shown in Figure A1.9.

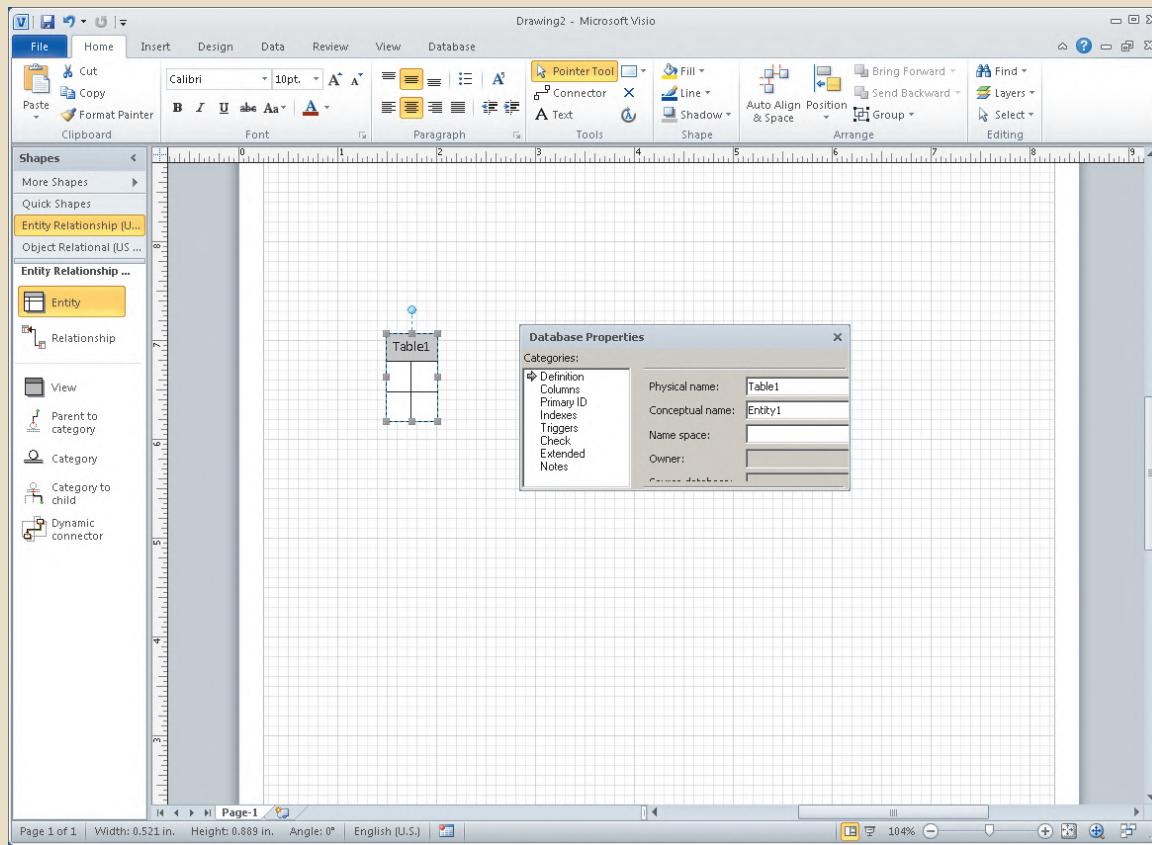
FIGURE A1.9 SELECTING THE DATABASE PROPERTIES... OPTION



Click the **Database Properties...** option shown in Figure A1.9 to display the **Database Properties** window. Figure A1.10 shows you a typical placement of the window. In that example, the **Database Properties** window is located on the grid, next to the **Table1** object. You will learn how to change the window’s location and format.

A1-10 Appendix A1

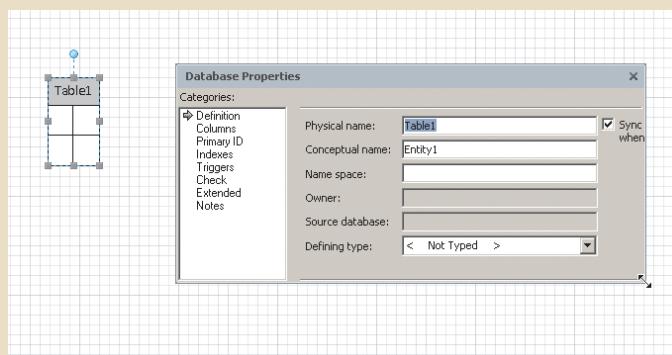
FIGURE A1.10 THE DATABASE PROPERTIES WINDOW



A1-3c Sizing the Database Properties Window

You can size the **Database Properties** window as you would size any Windows object. For example, note that placing the cursor in the lower right margin (see Figure A1.11) changes the cursor shape to a double-sided arrow in preparation for resizing the window by dragging its lower and right limits.

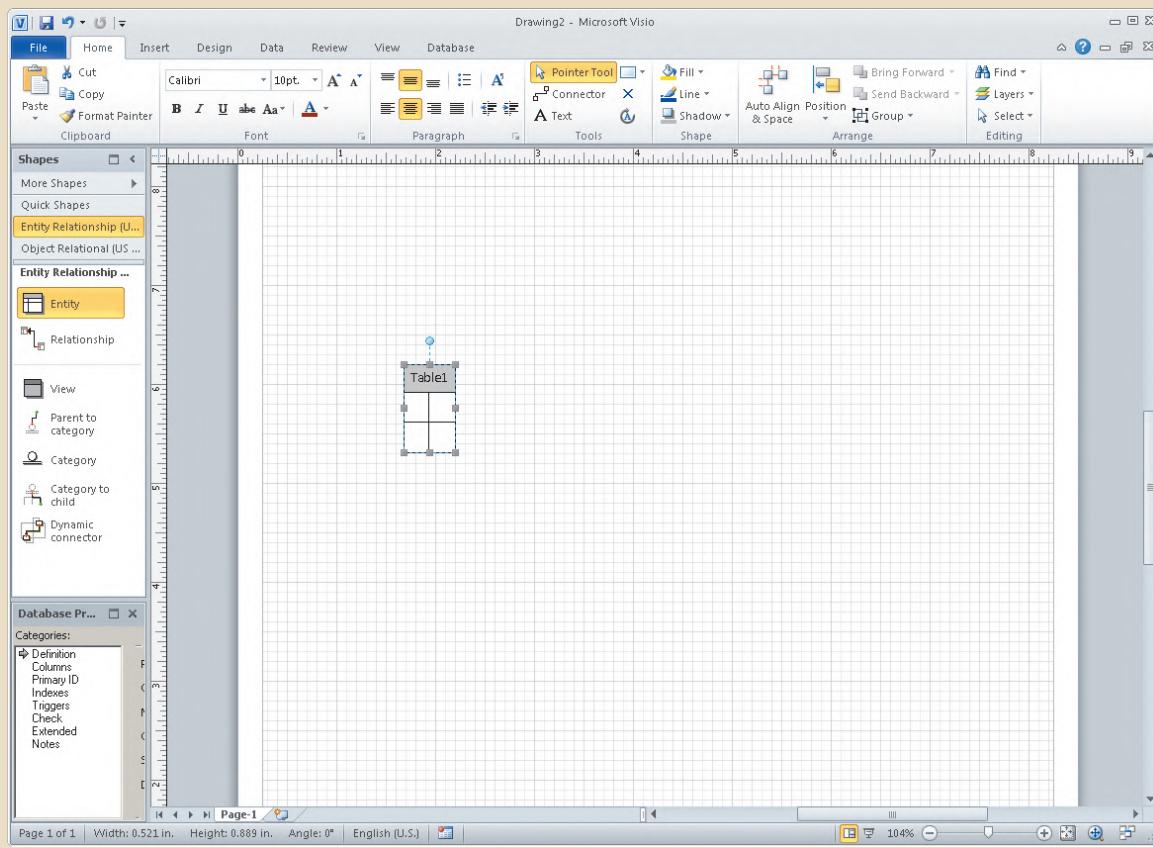
FIGURE A1.11 SIZING THE DATABASE PROPERTIES WINDOW



A1-3d Moving the Database Properties Window

You can also drag and drop the entire **Database Properties** window to the screen's lower-left corner. (See Figure A1.12.)

FIGURE A1.12 THE DATABASE PROPERTIES WINDOW IN THE LOWER-LEFT CORNER

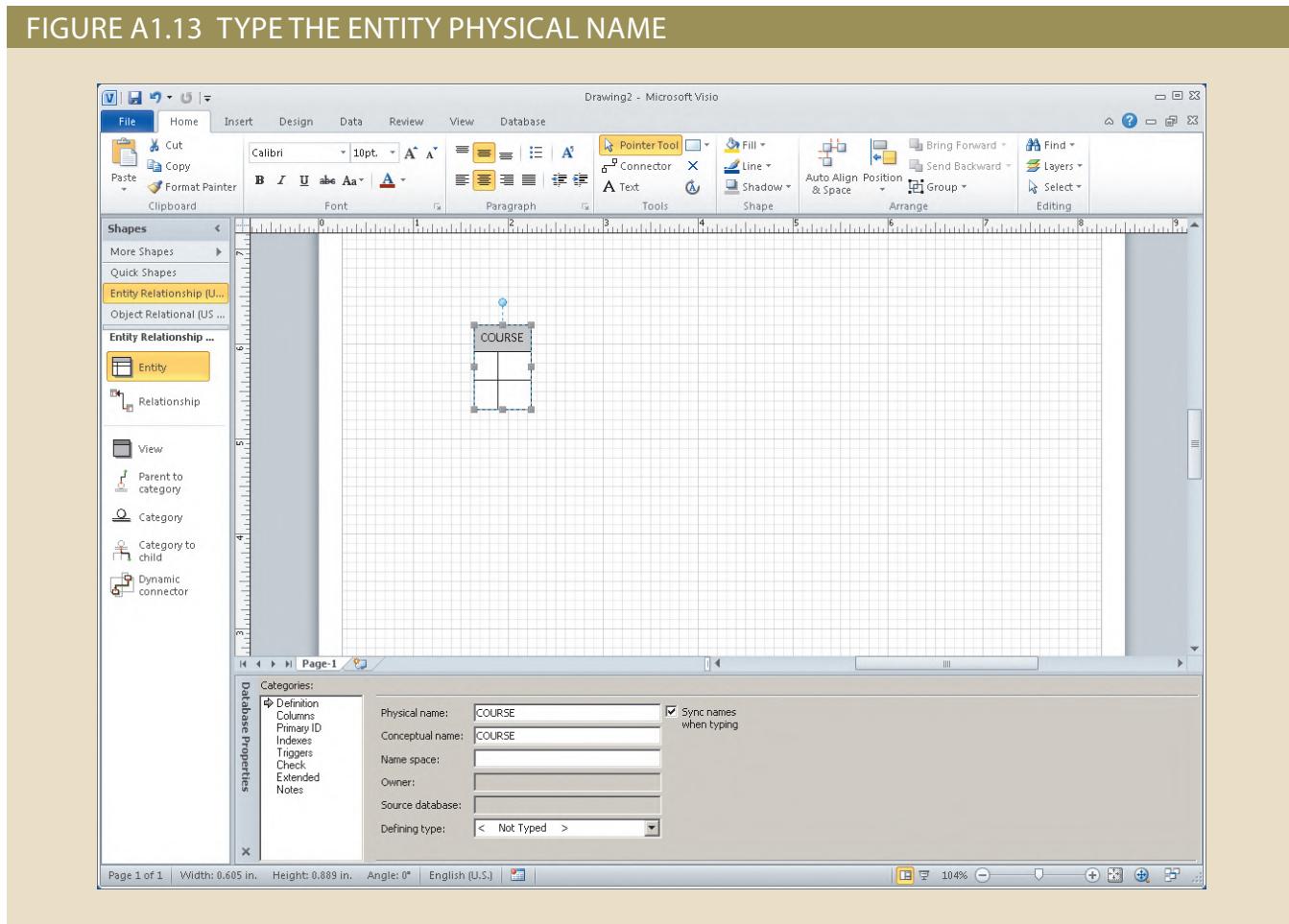


Naturally, you can also drag and drop the **Database Properties** window back to its original position depicted in Figure A1.8. (Just drag and drop to the screen's bottom margin.) Because that location allows you to see more of the database properties without blocking part of the entities you draw on the screen, that's the position you'll use.

A1-3e Creating the Entity Name

First, create a COURSE entity by placing the cursor in the **Physical name:** slot and typing COURSE, as shown in Figure A1.13. Because the **Sync names when typing** (default) option was selected in Figure A1.13, the **Physical name:** and **Conceptual name:** entries are the same.

FIGURE A1.13 TYPE THE ENTITY PHYSICAL NAME



When you have finished typing the COURSE label in the **Physical name:** slot as shown in Figure A1.13, note that the conceptual table in the grid automatically inherits the COURSE label. You are now ready to start defining the table columns.

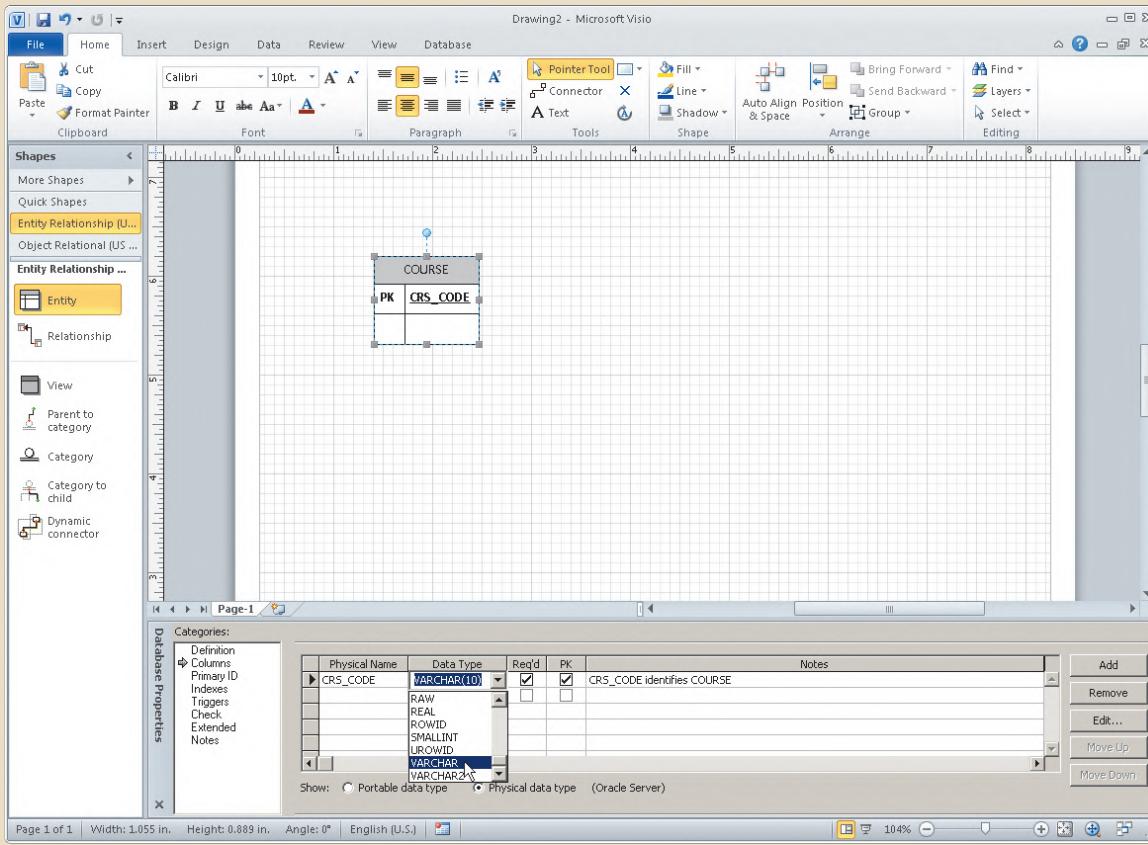
A1-3f Defining the Entity Attributes (Columns)

Each table column represents one of the characteristics (attributes or fields) of the entity. For example, if the COURSE entity, represented by the COURSE table, is described by the course code, the course description, and the course credits, you can expect to define three columns in the COURSE table. Table A1.1 provides a preview of the expected COURSE table structure. (A few sample records are entered to give you an idea of the COURSE table contents.)

SOME SAMPLE COURSE RECORDS			
CRS_CODE	CRS_TITLE	CRS_DESCRIPTION	CRS_CREDITS
ACCT-345	Managerial Accounting	Accounting as a management tool. Prerequisites: Junior standing and ACCT-234 and 245.	3
CIS-456	Database Systems Design	Creation of conceptual models, logical models, and design implementation. Includes basic database applications development and the role of the database administrator. Prerequisites: Senior standing and at least 12 credit hours in computer information systems, including CIS-234 and CIS-345.	4
ECON-101	Introduction to Economics	An introduction to economic history and basic economic principles. Not available for credit to economics and finance majors.	3

To define the columns of the COURSE table, you must assign column names and characteristics. The first column in the COURSE table will be the CRS_CODE, which serves as the table's primary key (PK). Because typical course code entries might be values such as CIS-456 or ACCT-234, each data entry involves a *character string*. Note that not all entries will have the same number of characters, so the course code is a variable length character string. In structured query language (SQL) terms, the CRS_CODE data is best defined as VARCHAR() data. Figure A1.14 shows you how the CRS_CODE name and data characteristics were specified.

FIGURE A1.14 THE COLUMN PK SELECTION

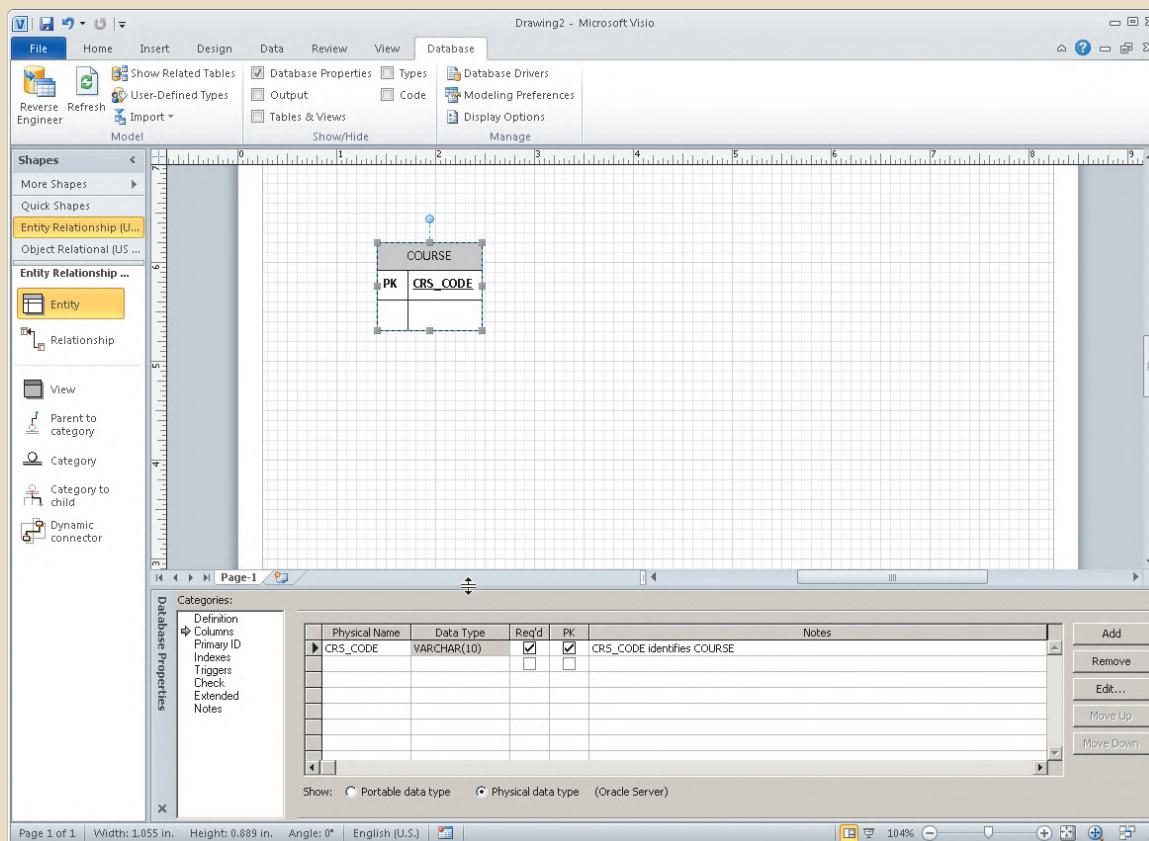


To generate the appropriate input for the column characteristics shown in Figure A1.14, follow these steps:

1. Make sure that the **COURSE** table object—shown in the grid—is selected. (The handles around the perimeter show that the selection was made properly.)
2. Select the **Columns** option in the **Database Properties** window at the bottom of the screen. (Note that the selection was marked with an arrow.)
3. Step 2 generates the column-specific dialog box. Type **CRS_CODE** in the first line under the **Physical Name** header. Moving along the line for the **CRS_CODE** entry:
 - a. Select the **VARCHAR** option from the drop-down list under the **Data Type** header. (Click the **down arrow** to generate the list.)
 - b. Because a course code is required to define the course offering, place a check mark—by clicking the check box—under the **Req'd** header.
 - c. Because the **CRS_CODE** is the PK, place a check mark—by clicking the check box—under the **PK** header.

Before you enter the remaining attribute names and characteristics, you may need to enlarge the **Database Properties** window by dragging its upper limit (see Figure A1.15) to increase the desired space. That action lets you see sufficient space for all of the remaining attributes in the **COURSE** table. Now place the cursor on the second **Columns** line and get ready to enter the remaining attributes.

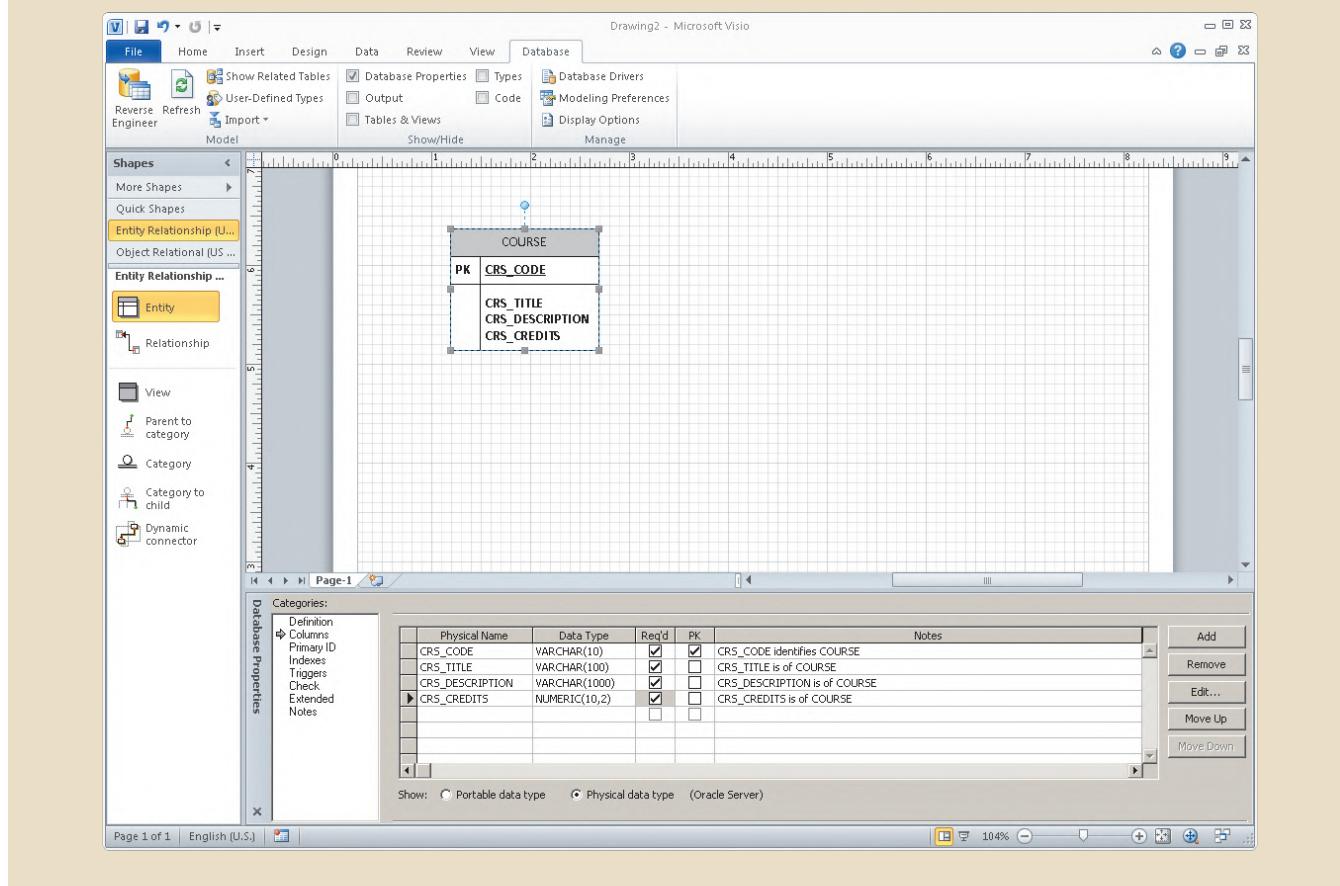
FIGURE A1.15 DRAG THE DATABASE PROPERTIES BOX LIMIT TO SHOW MORE COLUMNS



You are now ready to make the entries for the second COURSE attribute. Name this attribute **CRS_TITLE**. Typical entries are *Database Design and Implementation* or *Intermediate Accounting*. (Check the sample entries in Table A1.1.) Therefore, the CRS_TITLE is a character field. Double-click the selected data type to allow editing of the length of the character string allowed. Set the CRS_TITLE to a maximum length of 100 characters. The course title is required, but it is not a PK. Similarly, enter the CRS_DESCRIPTION entries with a maximum length of 1000 characters, and make it required. The CRS_CREDITS entries are numeric, and they are required; they will be used at some point to help compute grade point averages for the students taking a section of this course. When the appropriate entries are made, the screen will look like Figure A1.16. Note that the attribute names become boldfaced when the **Req'd** (required) option is checked for the **Column** property. Selecting that option means that the design should be implemented such that the end user will be required to provide a value for the checked attributes during data entry activities.

A1-16 Appendix A1

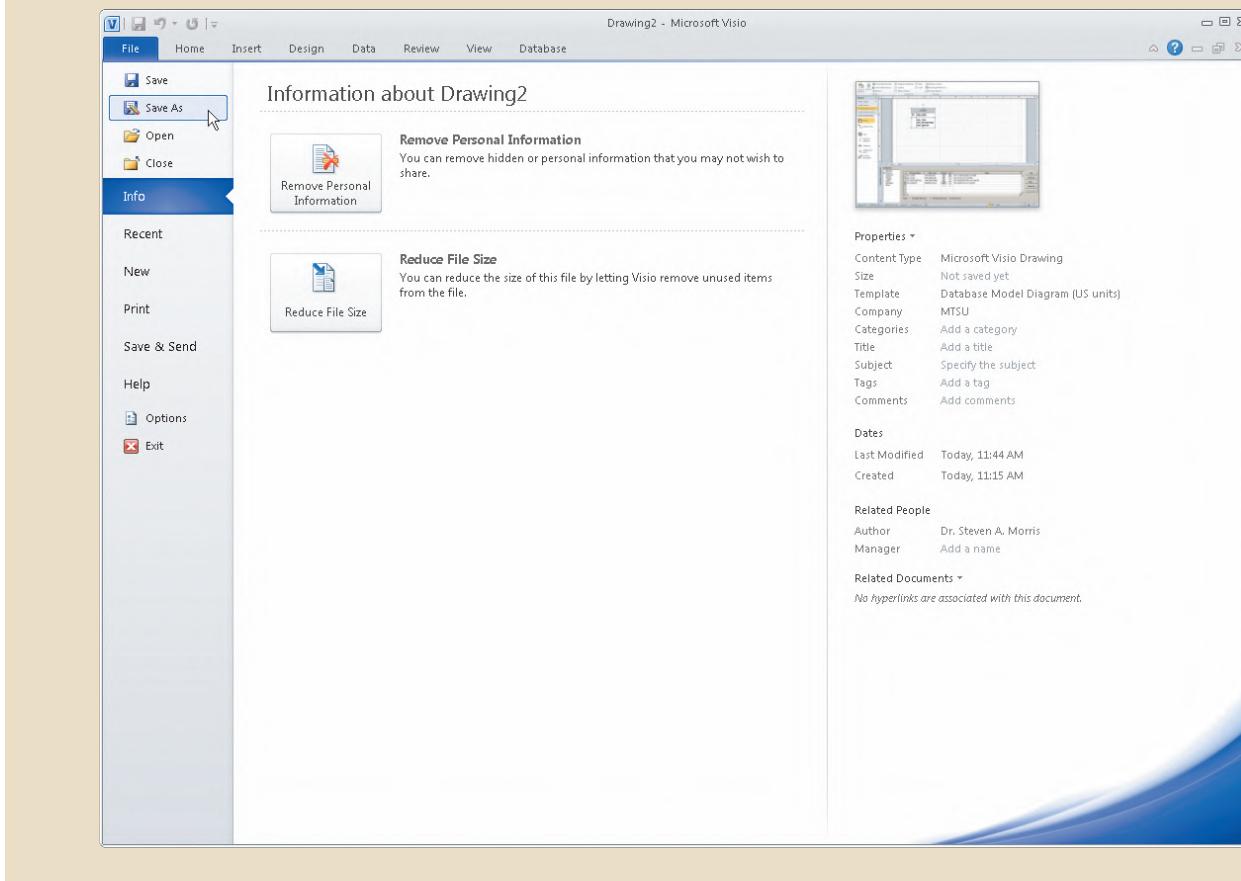
FIGURE A1.16 ENTER THE REMAINING COLUMNS



A1-4 Saving and Opening the Visio ERD

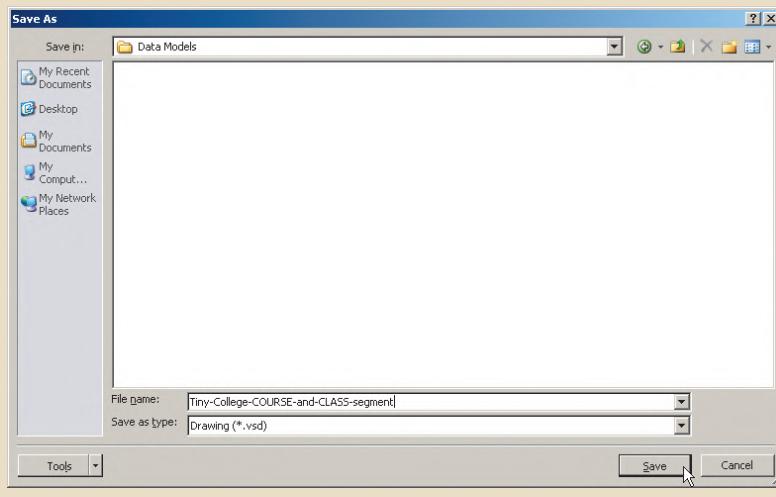
To ensure that you don't lose this first Visio Professional ERD segment, save it in an appropriate folder. Use the **File, Save As** option to select the folder location and the file-name, as shown in Figures A1.17 and A1.18.

FIGURE A1.17 SELECT THE SAVE AS OPTION TO SAVE THE FILE



A1-18 Appendix A1

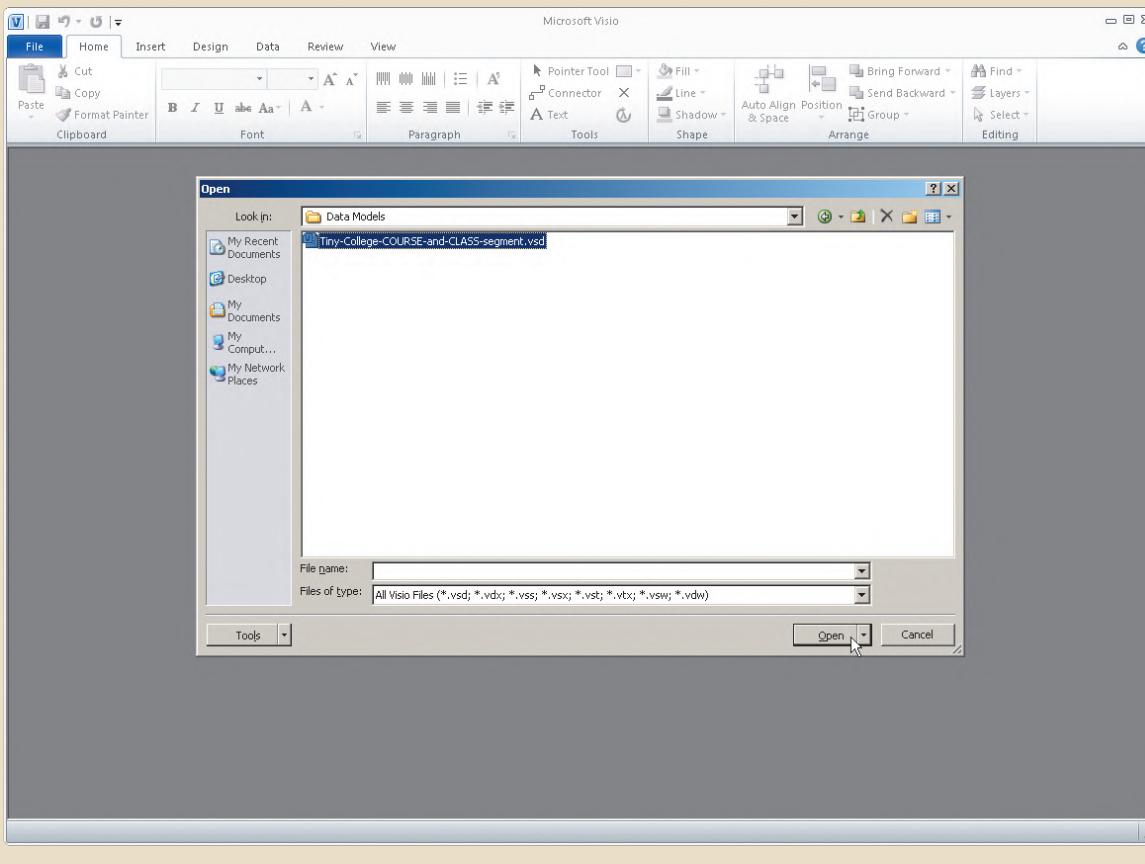
FIGURE A1.18 SELECT THE FOLDER, TYPE THE FILE NAME, AND SPECIFY THE FILE TYPE



As you examine Figure A1.18, note that the filename describes its origin and purpose. In this example, the ERD is named **Tiny-College-COURSE-and-CLASS-segment**. The naming convention serves the important purpose of self-documentation. Note also that the file is saved as a Visio Drawing (*.vsd).

You can now go ahead and close the file—and, of course, make a backup copy! The next time you want to use the file, after you start Visio Professional, use the standard Windows **File, Open** option to retrieve and open the file, the process used to generate the screen shown in Figure A1.19.

FIGURE A1.19 OPEN THE PREVIOUSLY SAVED FILE

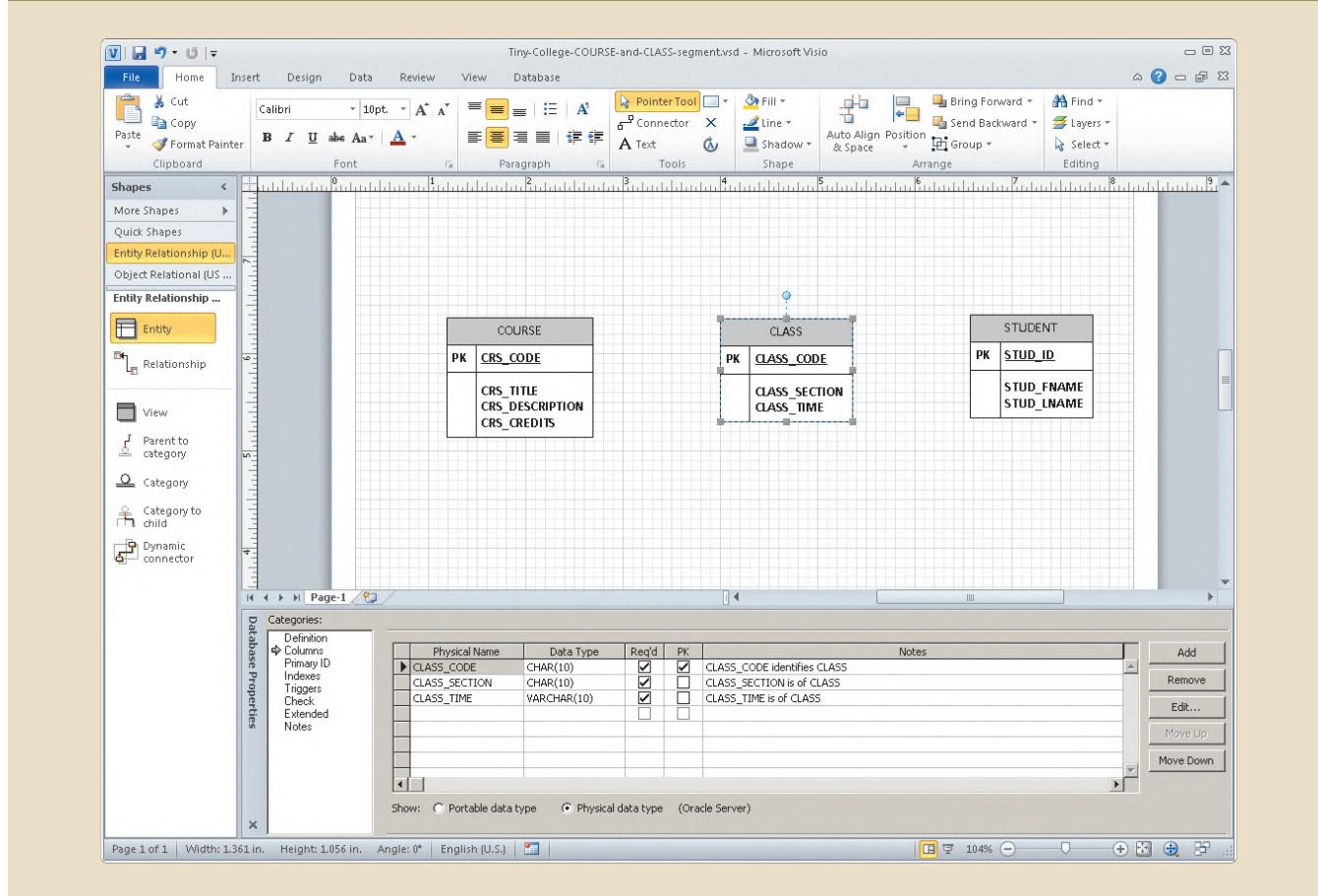


Note that the just-opened file does not show any entity properties. If you want to see this entity's properties, click the COURSE table to display its characteristics in the **Database Properties** window again.

You are now ready to define the CLASS and STUDENT entities, using the same techniques you used to create the COURSE entity. When you are done, the screen will look like Figure A1.20.

A1-20 Appendix A1

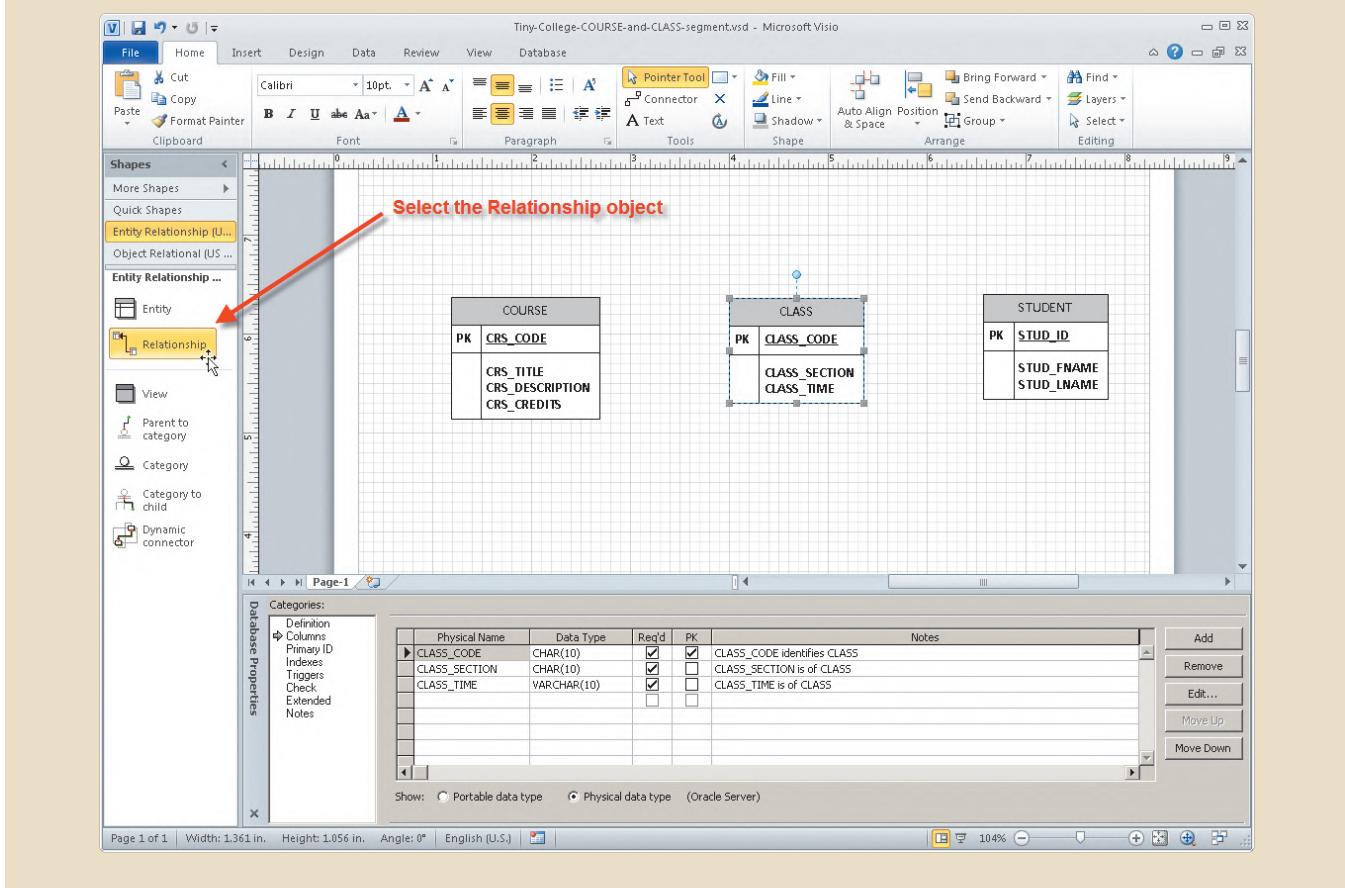
FIGURE A1.20 ADDING THE CLASS AND STUDENT ENTITIES



A1-5 Defining Relationships

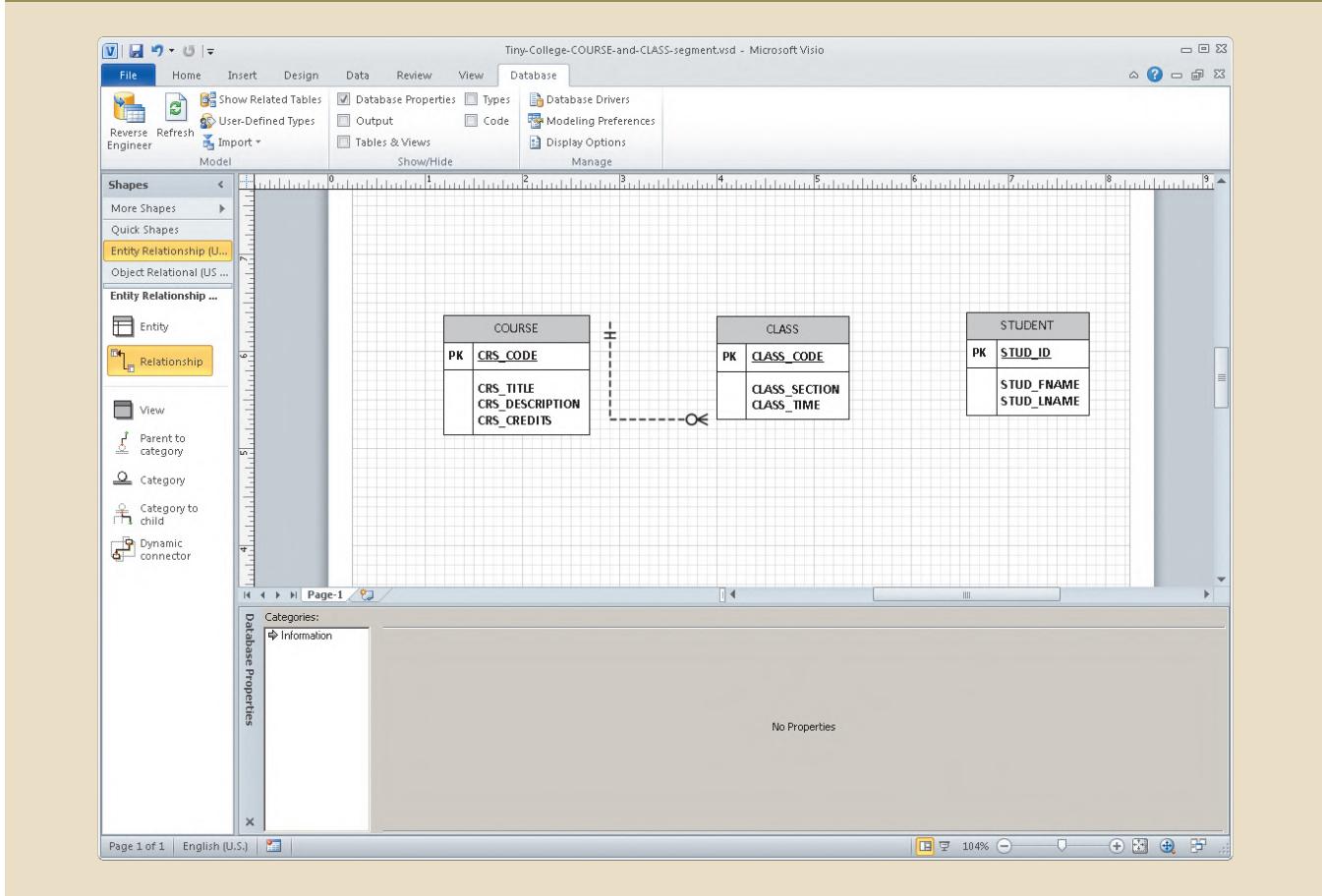
As you examine Figure A1.20, note that a foreign key (FK) has not been defined in CLASS to relate CLASS to COURSE. Instead, *Visio Professional will define the FK field when you specify the relationship between the two entities. Do not enter your own FK fields!* (Visio Professional tells you what the relationship option will do for you—read the relationship text in Figure A1.21).

FIGURE A1.21 SELECT THE RELATIONSHIP OBJECT



To create a relationship between the entities, click the **Relationship** object, drag it to the grid, and drop it between the COURSE and CLASS entities to produce the results shown in Figure A1.22.

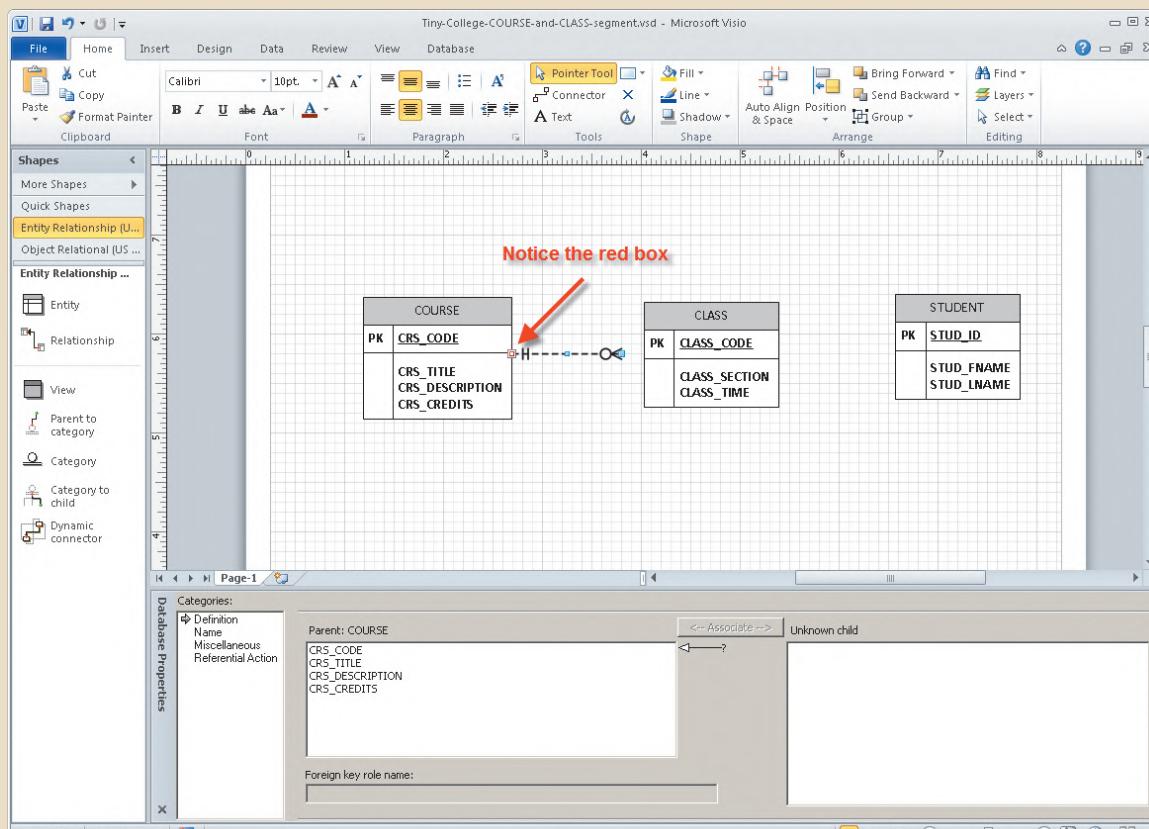
FIGURE A1.22 DRAG AND DROP THE RELATIONSHIP OBJECT



Dropping the **Relationship** object on the grid produces the relationship line. Further note that the symbols at the two ends of the relationship line reflect default cardinalities of (1,1) and (0,N). Finally, remember that the relationship to be established between COURSE and CLASS reflects the business rule “One COURSE may generate many CLASSES.” Therefore, the COURSE represents the “1” side of the relationship and the CLASS represents the “many” side of the relationship.

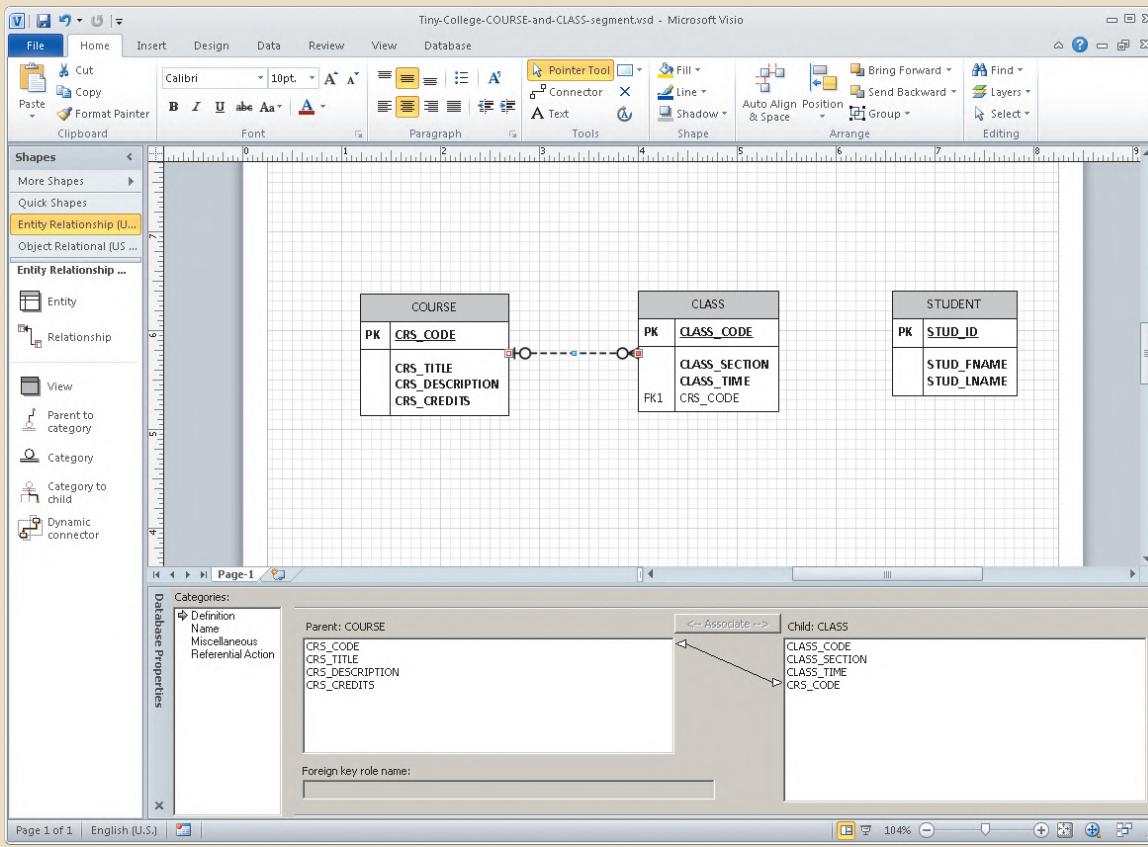
Attach the “1” side of the relationship line to the COURSE entity by dragging the “1” end of the relationship line to the COURSE entity, as shown in Figure A1.23. Note—and this is very important—the relationship is not attached until the COURSE table is outlined in red. (You may have to drag the relationship line’s end all the way to the inside of the table before the red outline shows up.) When you release the relationship line, its attachment is verified by the red square on the entity (table) perimeter.

FIGURE A1.23 ATTACH THE "1" SIDE OF THE RELATIONSHIP LINE



Using the same technique that was used to attach the “1” side of the relationship, drag the “M” side of the relationship line to the CLASS entity to produce Figure A1.24. (Make sure that you see the red square on the CLASS entity side of the relationship line when you are done.)

FIGURE A1.24 ATTACH THE "M" SIDE OF THE RELATIONSHIP LINE



As you examine Figure A1.24, note these features:

1. The two red rectangles at the margin of each table indicate that the relationship was successfully established and that it is still selected. (If the relationship line is no longer selected, the red squares disappear. *To reselect the relationship line, click it.*)
2. Visio Professional created the CRS_CODE foreign key in the CLASS table, labeling it **FK1** to indicate that it is the first FK created for this table. Note that CRS_CODE in the CLASS table is not in boldfaced type. This lack of boldface indicates that, at this point, you have not yet specified that an FK value is mandatory. (Of course, it should be because a CLASS cannot exist without a COURSE. You will edit this FK property later.)
3. The cardinality next to COURSE was automatically changed to indicate an optional (0,1) relationship between CLASS and COURSE. Because each class must be related to one course, a depiction of a (1,1) cardinality is appropriate. (A CLASS cannot exist without a COURSE.) Therefore, you'll have to edit this cardinality later.
4. The **Database Properties** window shows that the (default) **Definition** option is selected. (Look under the **Categories:** header.)
5. The relationship is reflected in the double-sided arrow linking the COURSE table's CRS_CODE and the CLASS table's CRS_CODE.

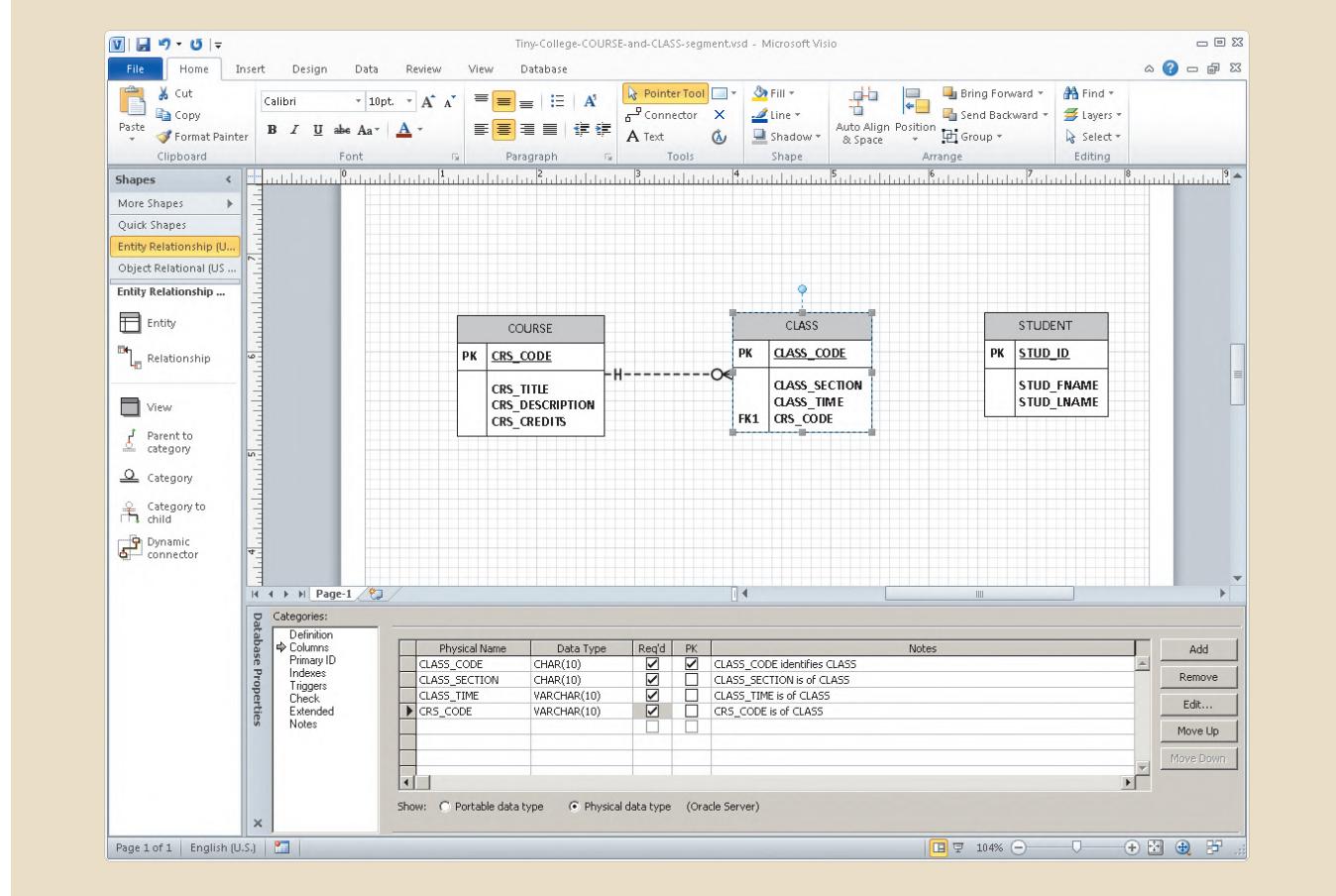
A1-5a Editing the Cardinalities

If you examine Figure A1.24, you'll notice that the CRS_CODE in the CLASS entity is not in boldfaced type. This lack of boldface indicates that the CRS_CODE in CLASS may be null, thus indicating incorrectly that COURSE is optional to CLASS. To change the (0,1) cardinality to a (1,1) cardinality:

1. Select the **CLASS** entity.
2. Check the CRS_CODE and note that its **Req'd** check box is *not* checked. (That means that a value entry is not required, thus allowing nulls—and making the relationship between CLASS and COURSE optional.)
3. Click the CLASS entity's CRS_CODE **Req'd** check box to place a check mark in it. (That means that a value entry will be required, thus making the relationship between CLASS and COURSE mandatory.)

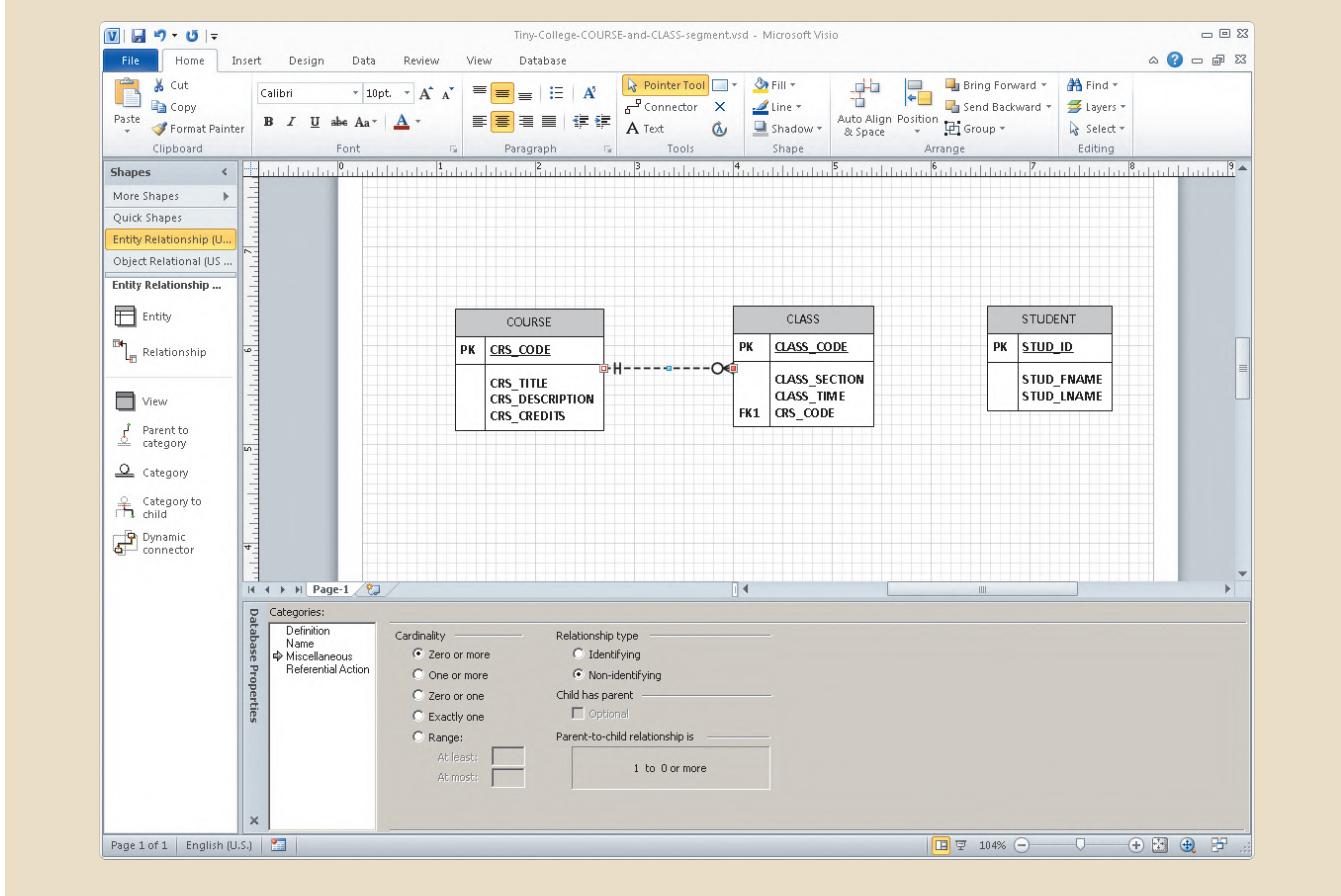
When you have completed those three steps, you will see the results in Figure A1.25. Note that the CRS_CODE in the CLASS entity is now in boldface to indicate the mandatory relationship between CLASS and COURSE. That mandatory relationship is reflected by the change in the (0,1) cardinality to a (1,1) cardinality on the COURSE entity.

FIGURE A1.25 FORCING A MANDATORY ENTRY FOR A FOREIGN KEY VALUE



You can edit the “M” side of the 1:M relationship by selecting the relationship line and the **Miscellaneous** option in the **Categories:** list. Then select the **Zero or more** cardinality (if it is not already selected). Figure A1.26 shows the screen after the selections have been properly made.

FIGURE A1.26 SELECTING THE CARDINALITY FOR THE "MANY" SIDE OF A RELATIONSHIP



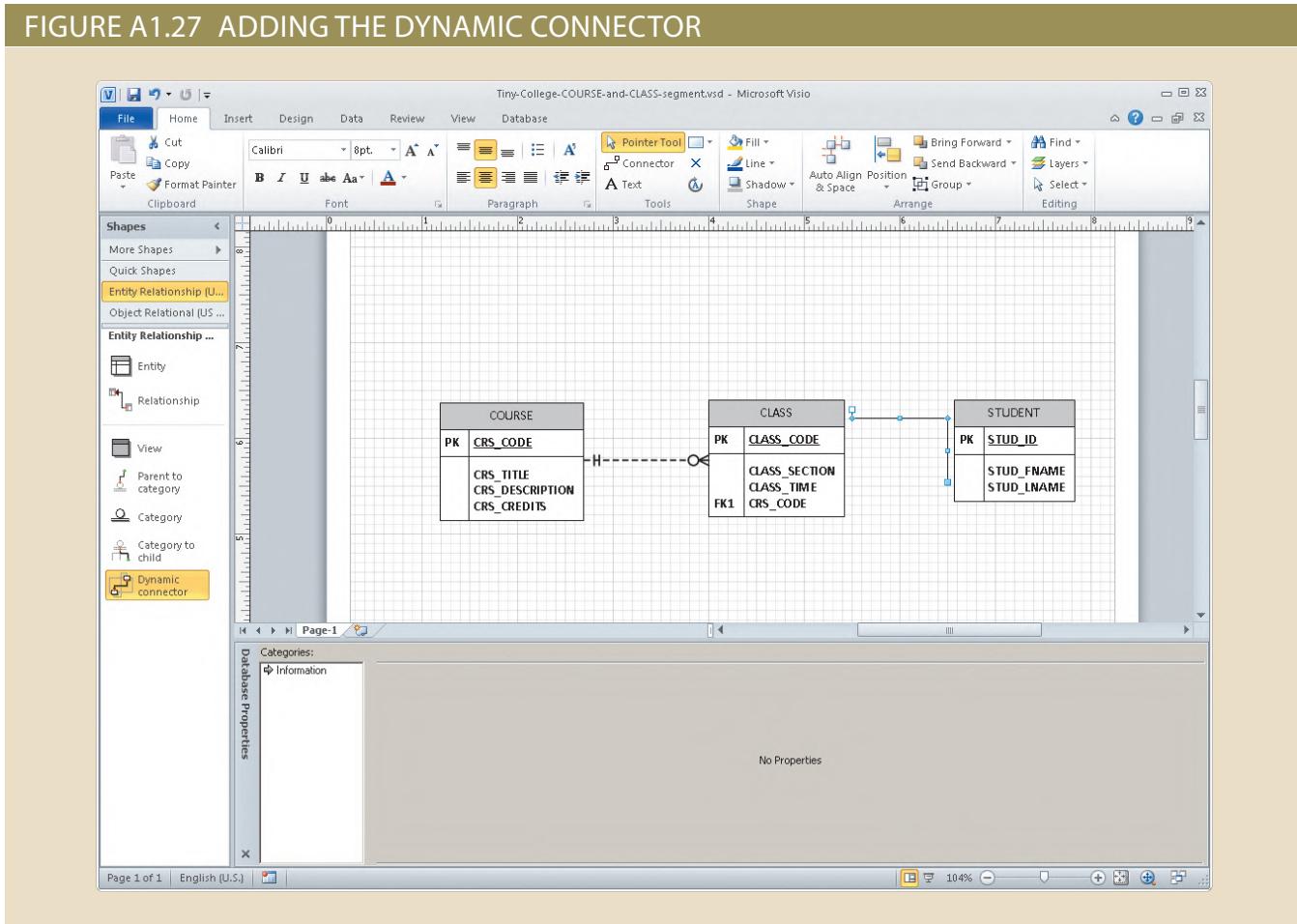
A1-5b Creating a “many-to-many” Relationship

As discussed in Chapter 4, Entity Relationship Modeling, “many-to-many” relationships are appropriate in conceptual data models. However, Visio focuses on the creation of logical models for relational databases, hence the name “Database Model Diagram” instead of “Data Model Diagram” for the drawing template. Recall that M:M relationships cannot be directly implemented in a relational database without the use of an associative or composite entity, and part of the process of converting a conceptual model to a logical model is the decomposition of M:M relationships into 1:M relationships. The result is that as a logical modeling tool, Visio does not expect you to be drawing M:M relationships. While it is possible to create these relationships in Visio, it is slightly different from other relationships.

To create a “many-to-many” relationship between the entities, click the **Dynamic connector** object, drag it to the grid, and drop it between the **CLASS** and **STUDENT** entities as shown in Figure A1.27. A dynamic connector is not the same as a relationship object in Visio. It does not automatically place foreign keys, Visio does not associate cardinalities with it, and it cannot embed primary key/foreign key relationships. A dynamic connector is simply a line that we can manually manipulate. Note that there are

no symbols at the ends of the relationship line to indicate cardinality. Remember that the relationship to be established between **CLASS** and **STUDENT** reflects the business rule “Many STUDENTs may enroll in many CLASSES.” Therefore, we need to change the representation of the relationship to represent the “many-to-many” side of it:

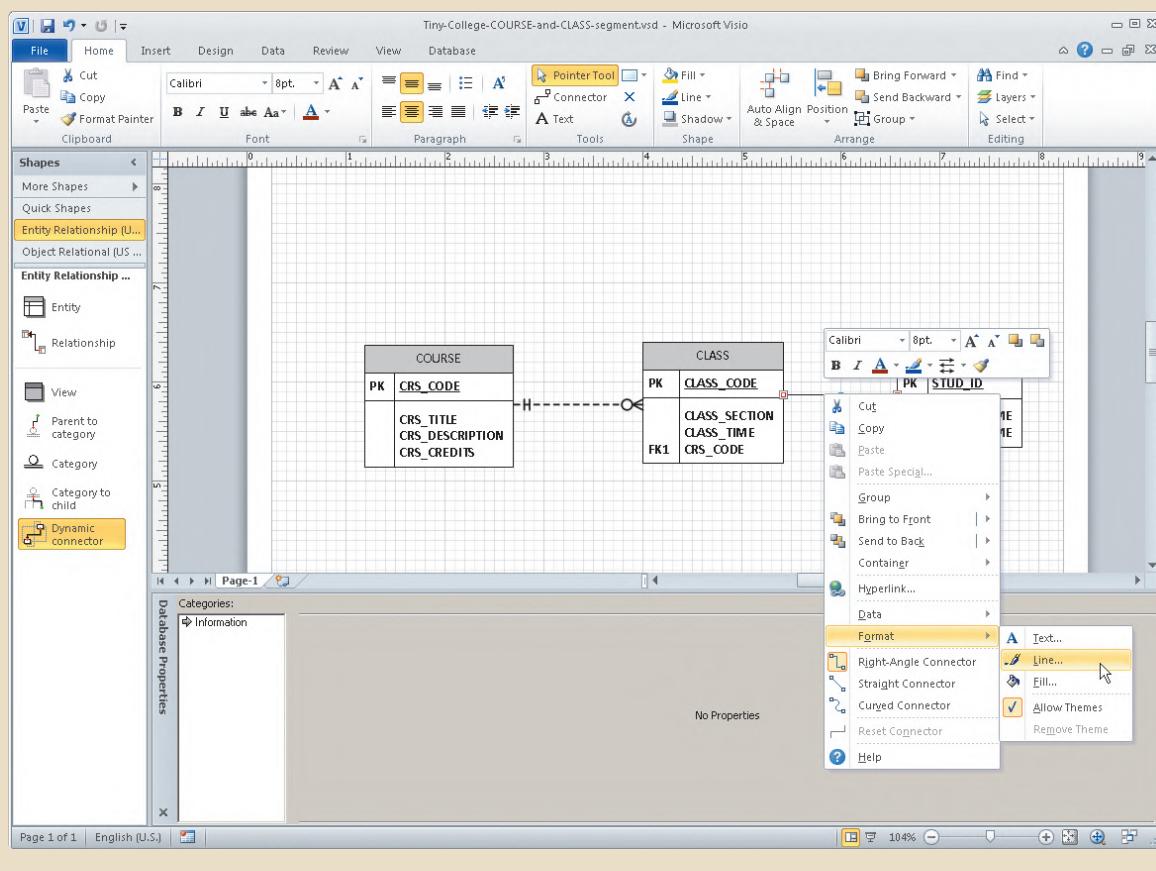
FIGURE A1.27 ADDING THE DYNAMIC CONNECTOR



Attach one side of the dynamic connector line to the **CLASS** entity and the other side of the line to the **STUDENT** entity. Right-click the line and select **Format → Line** as shown in Figure A1.28.

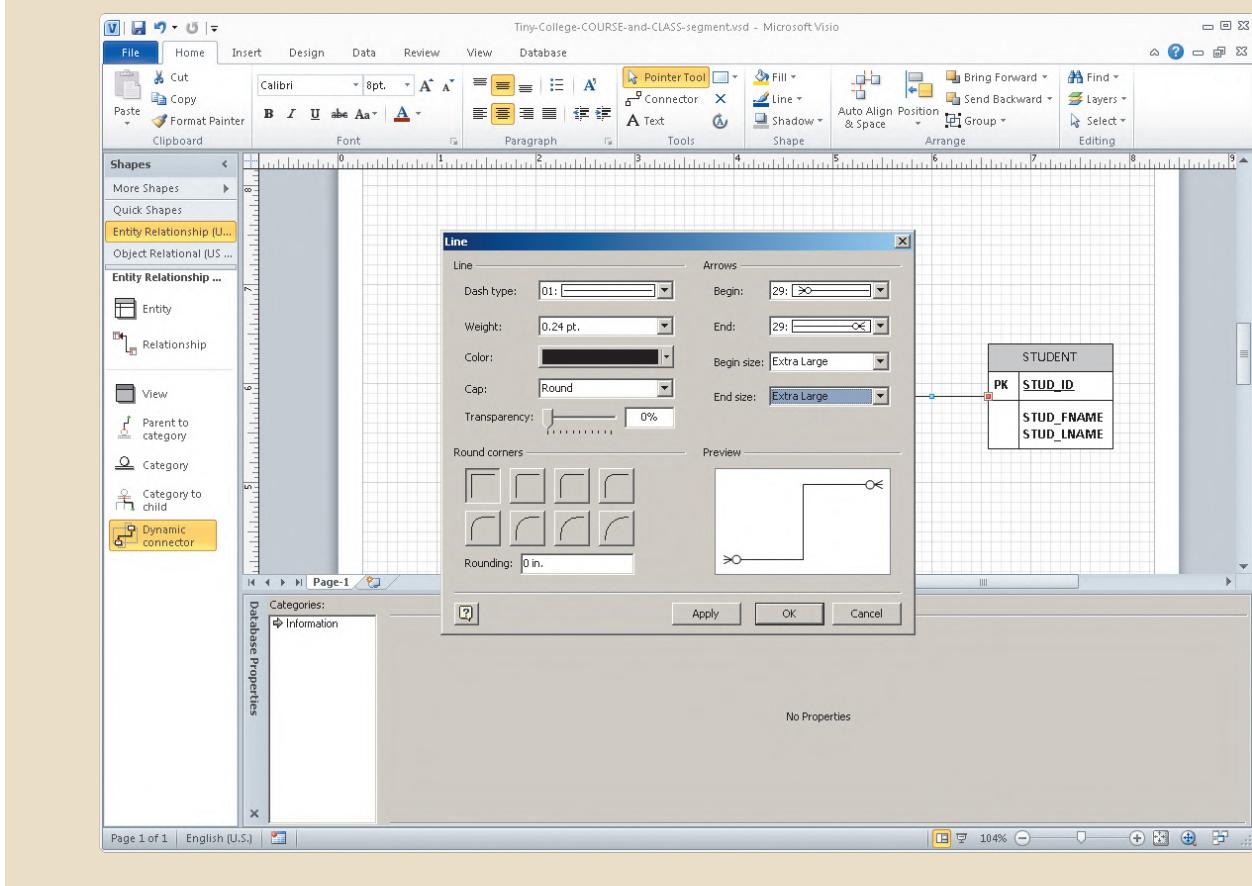
A1-28 Appendix A1

FIGURE A1.28 FORMAT THE "MANY-TO-MANY" RELATIONSHIP



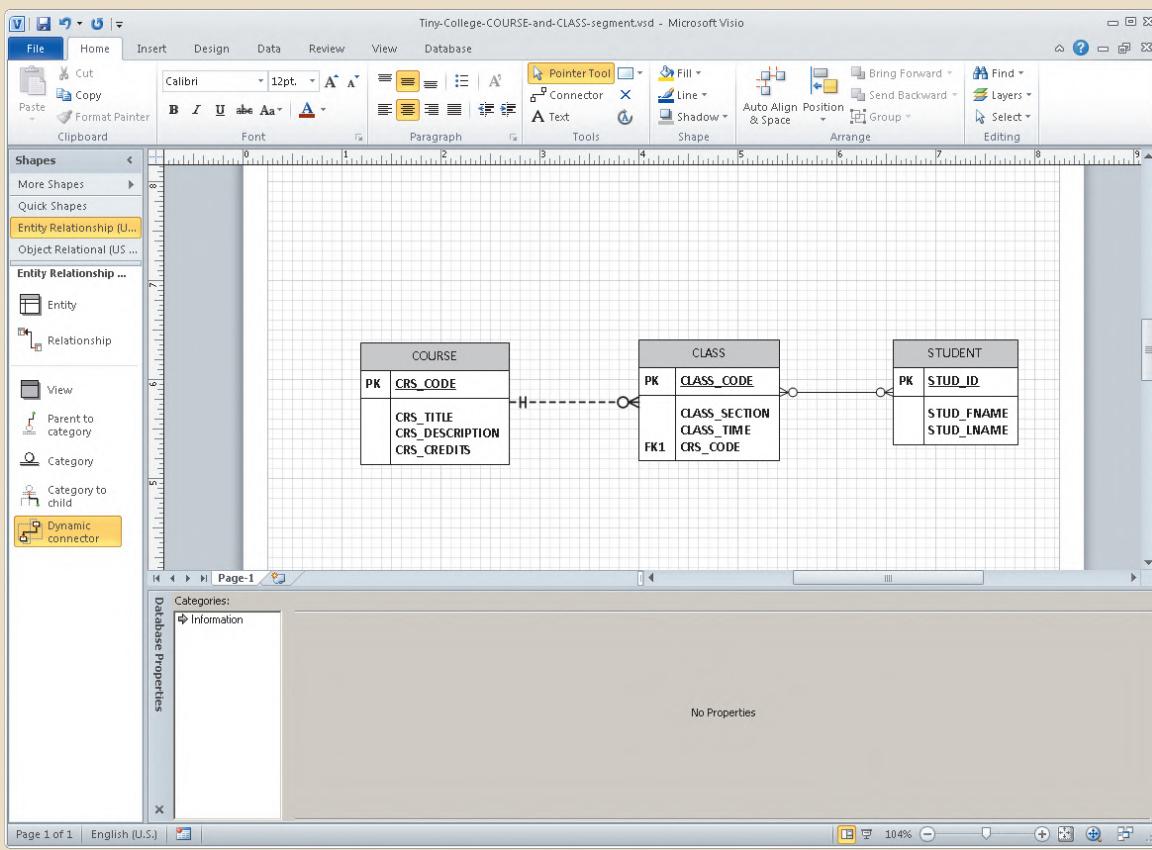
In the Line format window, we can manually select the appropriate Crow's foot notation symbols to represent the connectivity and participation that reflect our business rules. Select 29: (0, M) as the **Begin** and **End** symbols for the relationship line. Also, select to make the begin size and end size be **Extra Large**, as shown in Figure A1.29. Click **OK** to accept your selections.

FIGURE A1.29 SELECTING THE SYMBOLS OF A "MANY-TO-MANY" RELATIONSHIP



At this point, the diagram should accurately reflect the cardinalities of the relationships specified by the business rules. Figure A1.30 shows the many-to-many relationship between **CLASS** and **STUDENT** after the selections have been properly made. You will notice that the relationship line for the M:M relationship created using a dynamic connector is smaller than the 1:M relationship created using a relationship object. We will look at how to change the presentation of the relationship line, whether created by a relationship object or a dynamic connector, later in this tutorial.

FIGURE A1.30 "MANY-TO-MANY" RELATIONSHIP



A1-5c Selecting the Relationship Type

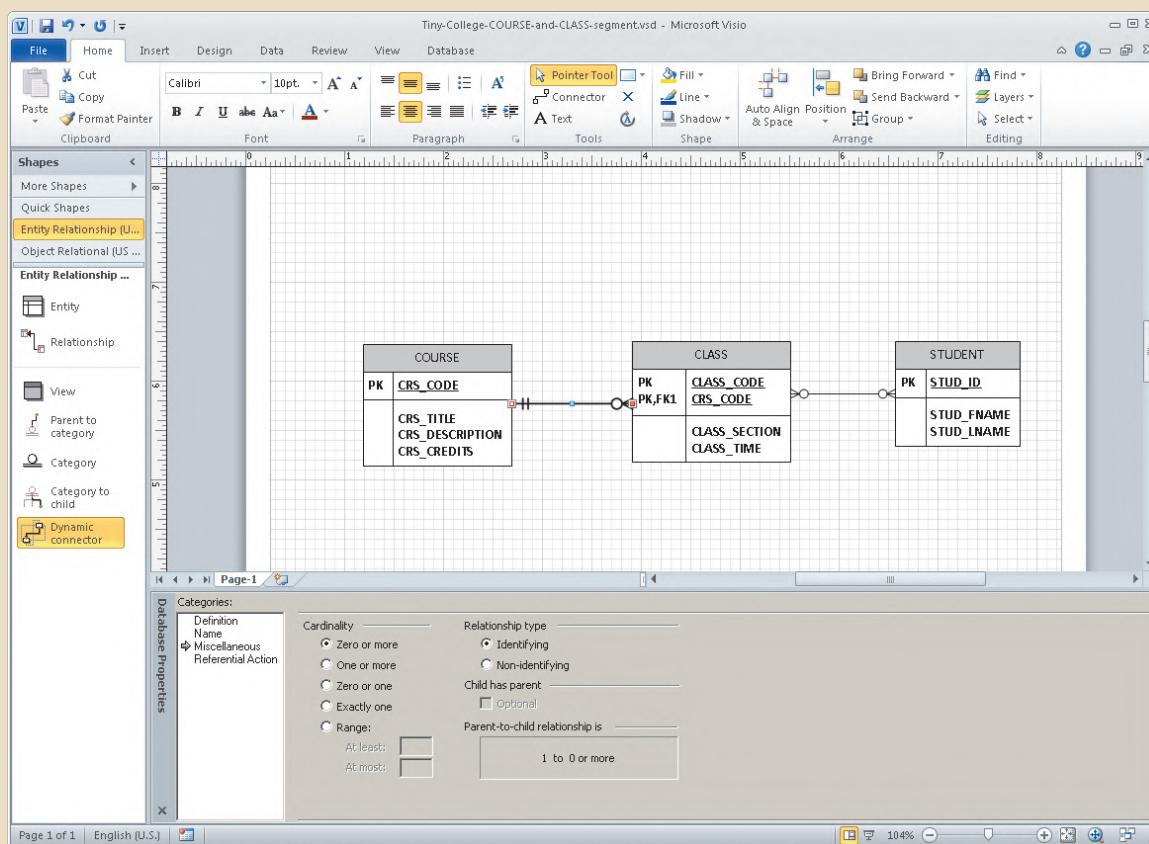
As you examine Figure A1.26, note the many options you have available. In this case, the relationship type is properly defined to be **Non-identifying** because the dependent CLASS entity did not inherit its PK from the parent COURSE entity. (When you created the CLASS entity, you defined its PK to be CLASS_CODE, which is not found in the COURSE entity. In other words, the ERD in Figure A1.26 indicates that the CLASS entity is *not* a weak entity. A weak entity always has a strong relationship—that is, an identifying relationship—with its parent entity.)

The nature of the relationships between entities, the effect of optional and mandatory participation, and the existence of weak entities all have critical effects on the database design. If necessary, review Chapter 4 to review the nature and implementation of relationships.

Figure A1.26 shows the relationship between COURSE and CLASS as a dashed line. In Visio, a dashed relationship line between two entities always indicates a non-identifying (weak) relationship. You should recall that a weak (non-identifying) relationship always indicates the existence of a strong dependent entity. Conversely, a strong (identifying) relationship always indicates the existence of a weak dependent entity.

If you select an *identifying* relationship between COURSE and CLASS, Visio will automatically rewrite the PK of the CLASS entity for you and the relationship line will be solid. Figure A1.31 shows the effect of the relationship revision. After you have examined the effect of the identifying relationship selection, reset the relationship type to the one shown in Figure A1.26. (If you want to preserve the identifying relationship version of the ERD, save it with a different name, such as **Tiny-College-COURSE-and-CLASS-segment-Identifying-Relationship**.)

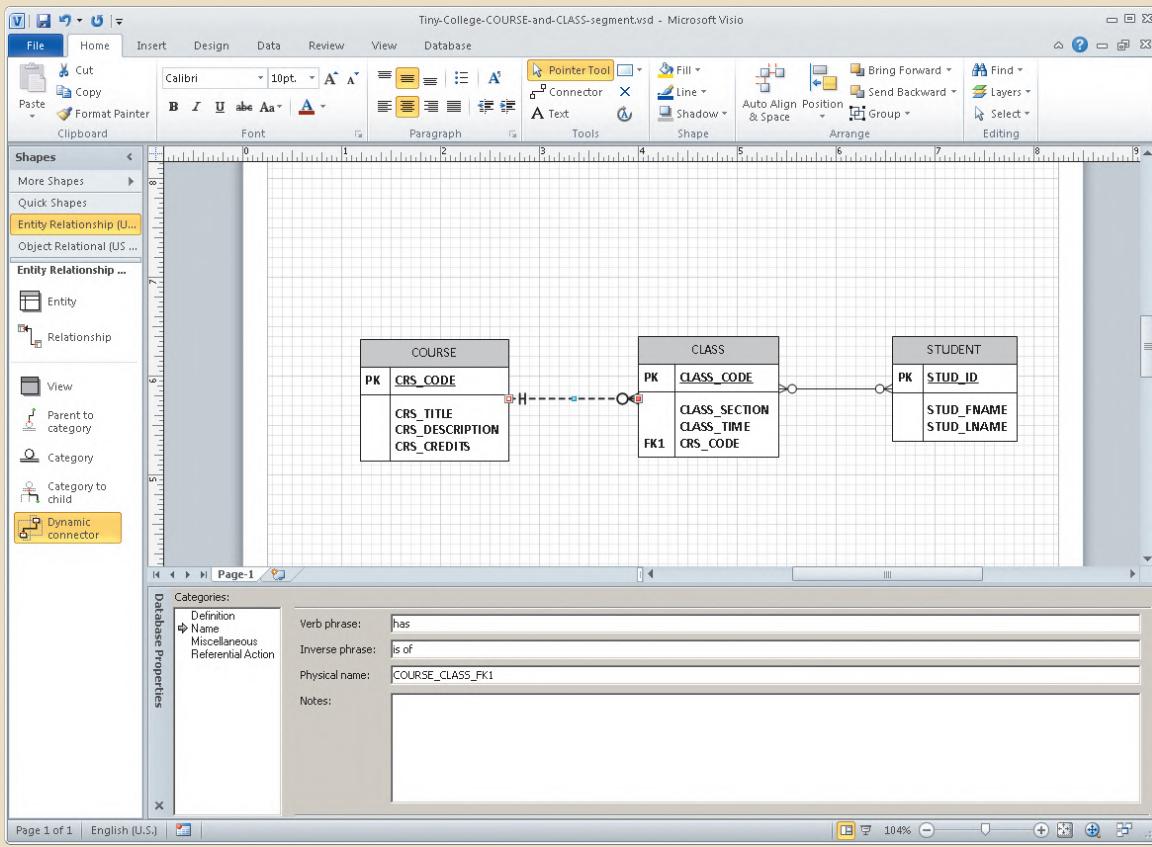
FIGURE A1.31 AN ILLUSTRATION OF AN IDENTIFYING (STRONG) RELATIONSHIP



A1-5d Naming the Relationships

Make sure that the relationship line is still selected. Then click the **Name** option in the **Database Properties** window at the bottom of the screen to produce the results displayed in Figure A1.32. (Note that the original ERD has been used to show the preferred non-identifying relationship between COURSE and CLASS.)

FIGURE A1.32 THE DEFAULT RELATIONSHIP NAME



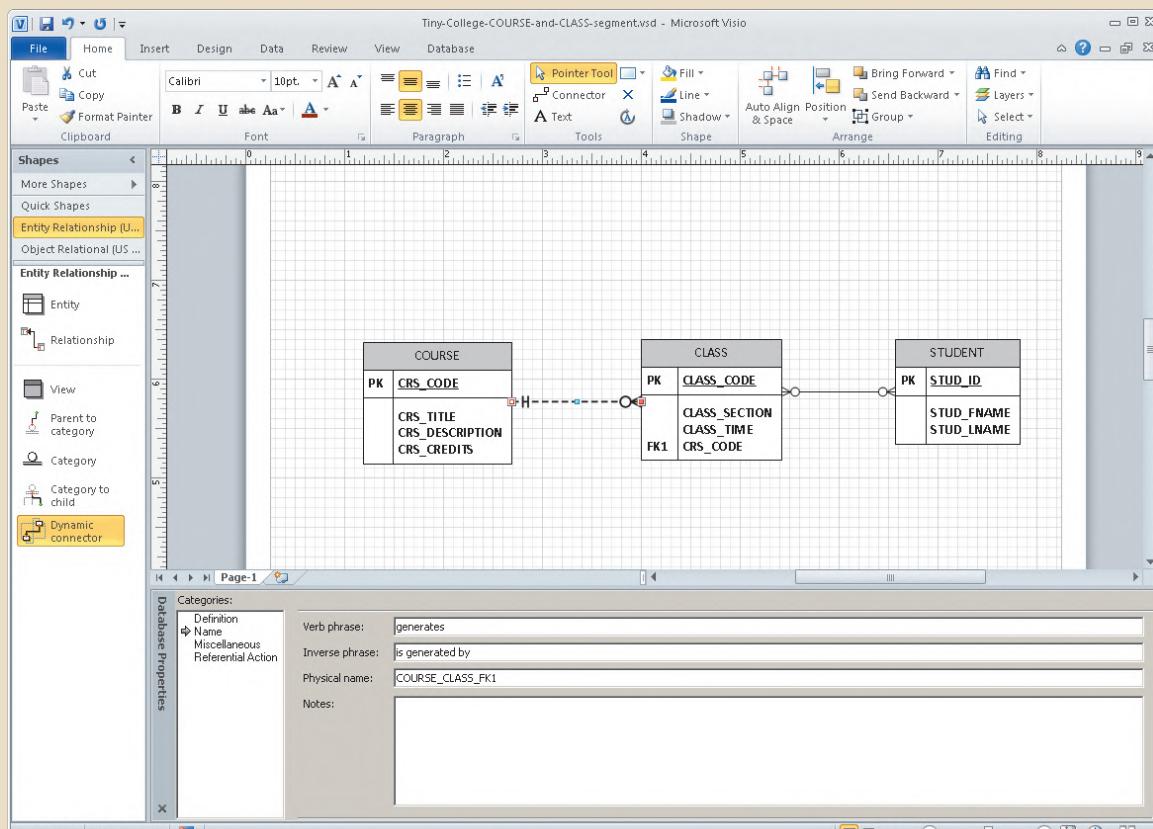
As you examine Figure A1.32, note that the default **Verb phrase** selection is **has** and that the default **Inverse phrase** selection is **is of**. It's useful to remember that:

1. All relationships are defined in both directions — from the “1” side to the “M” side, and from the “M” side to the “1” side.
2. Active verbs are typically used to label relationships from the “1” to the “M” side. Passive verbs are used to label relationships from the “M” to the “1” side.
3. Relationship names are written in lowercase.

Using the **Name** selection in Figure A1.32, type the **Verb phrase** and **Inverse phrase** entries “generates” and “is generated by” as shown in Figure A1.33. Note that active and passive verbs have been selected to describe the relationship between COURSE and CLASS in both directions, respectively:

1. COURSE **generates** CLASS.
2. CLASS **is generated by** COURSE.

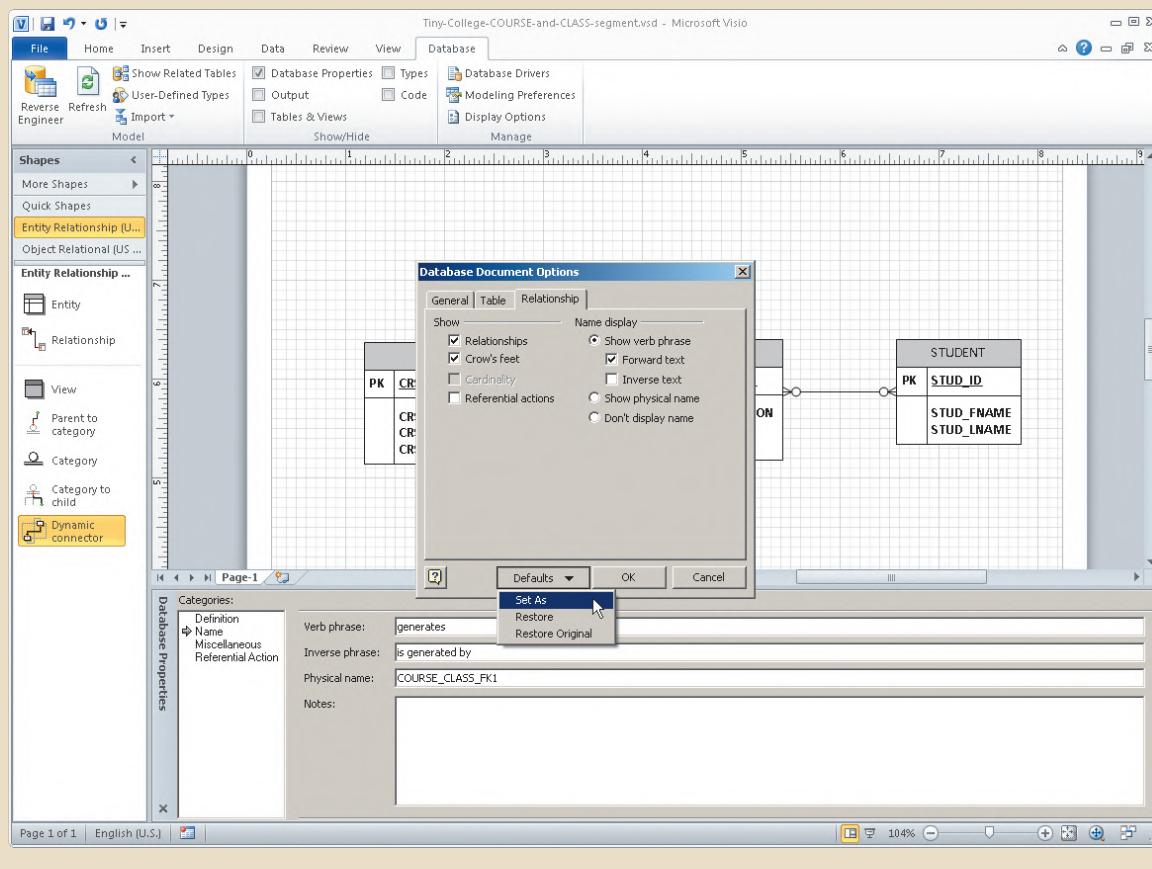
FIGURE A1.33 THE NAMED RELATIONSHIP



A1-5e Showing the Relationship Names

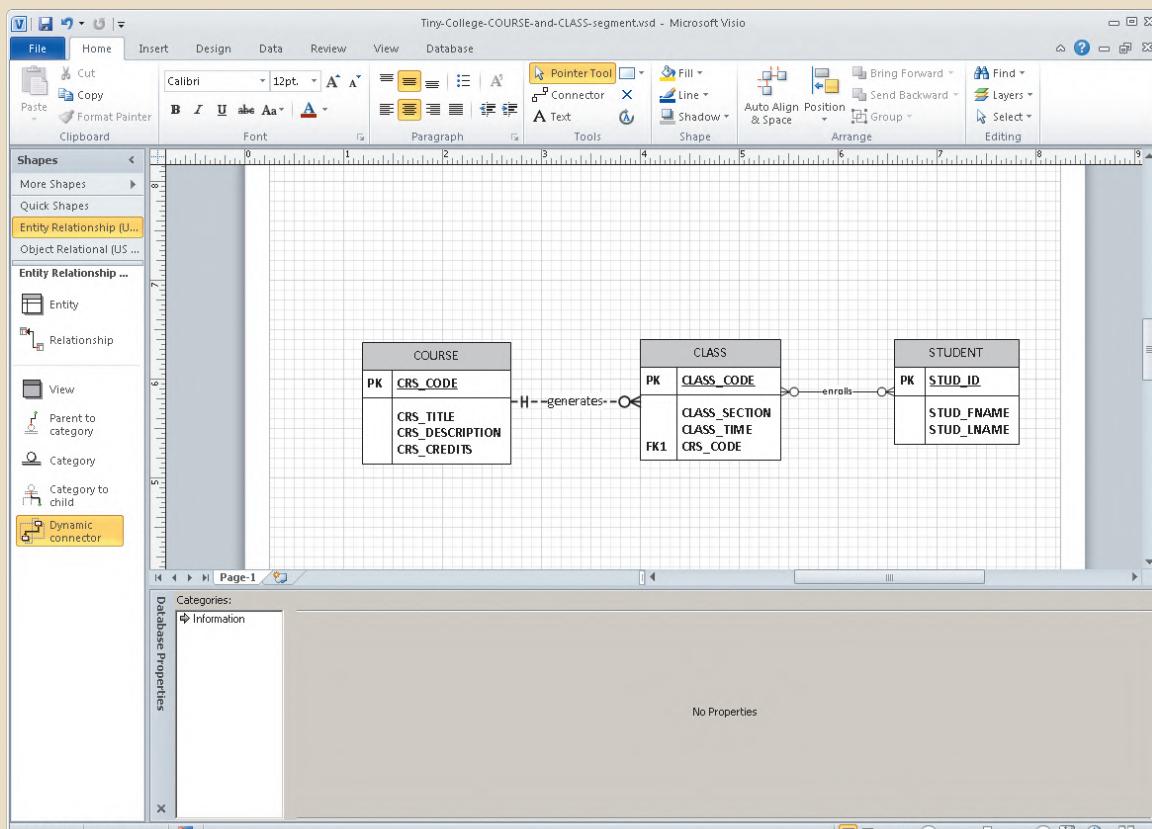
As you examine Figure A1.33, you may note that the relationship names are not shown. If you *do* want those relationship names shown, click the **Database** ribbon shown at the top of the screen and then select **Display Options** as you first saw in Figure A1.5.

Next, select the database document options **Relationship** tab (see Figure A1.34), select the **Show verb phrase** option, and then select the **Forward text** option and unselect the **Inverse text** option. (If you select *both* the **Forward text** and the **Inverse text** options, Visio writes the two relationship names on the same line and separates them with a slash. That option takes more space, so you may have to move the tables farther apart to make the relationship names readable.) Finally, select the **Defaults** button and choose **Set As** as shown in Figure A1.34, and then click the **OK** button to set the selection as the default.

A1-34 Appendix A1**FIGURE A1.34 SET THE RELATIONSHIP NAMES AS THE DEFAULT**

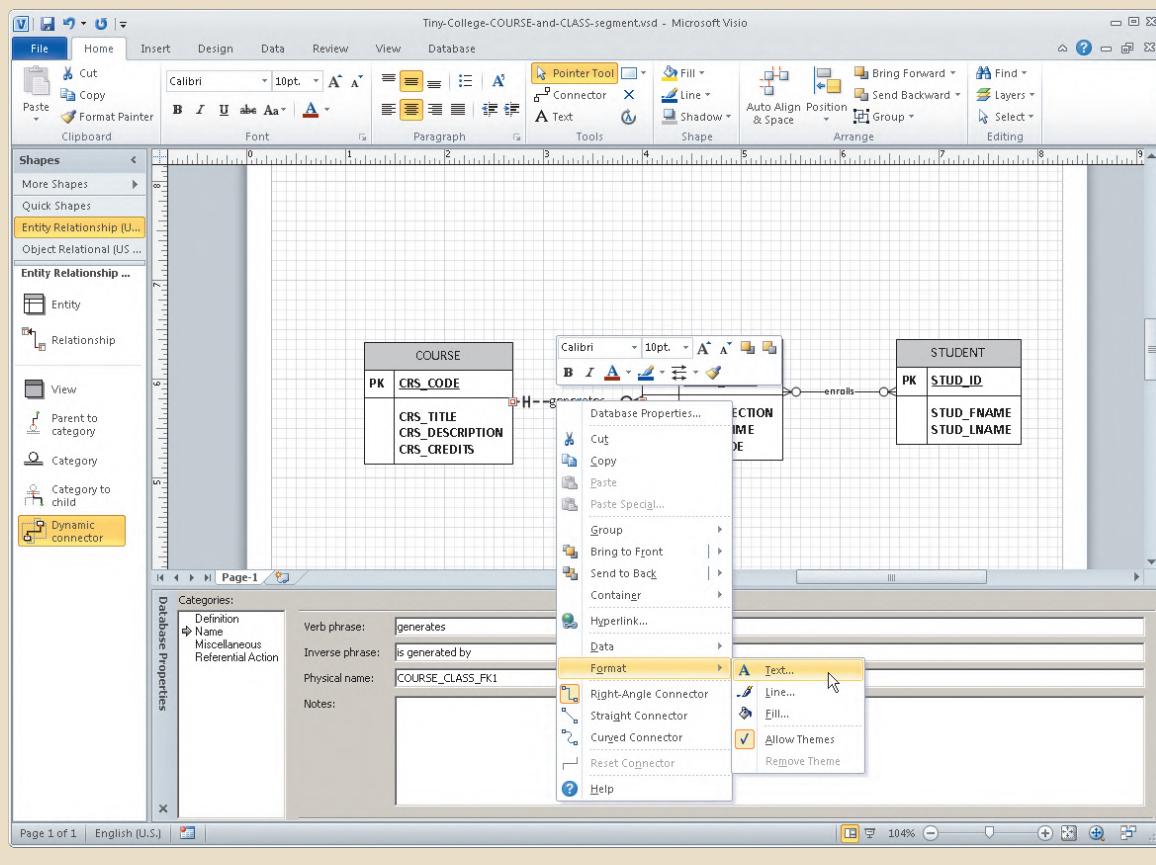
To add a name to the M:M relationship that was created using a dynamic connector, click to select the connector and simply begin typing. Remember, dynamic connectors only provide a visual representation — Visio is not actually tracking or maintaining a relationship through the dynamic connector. Figure A1.35 shows the ERD with the relationship names visible.

FIGURE A1.35 SHOWING THE RELATIONSHIP NAMES



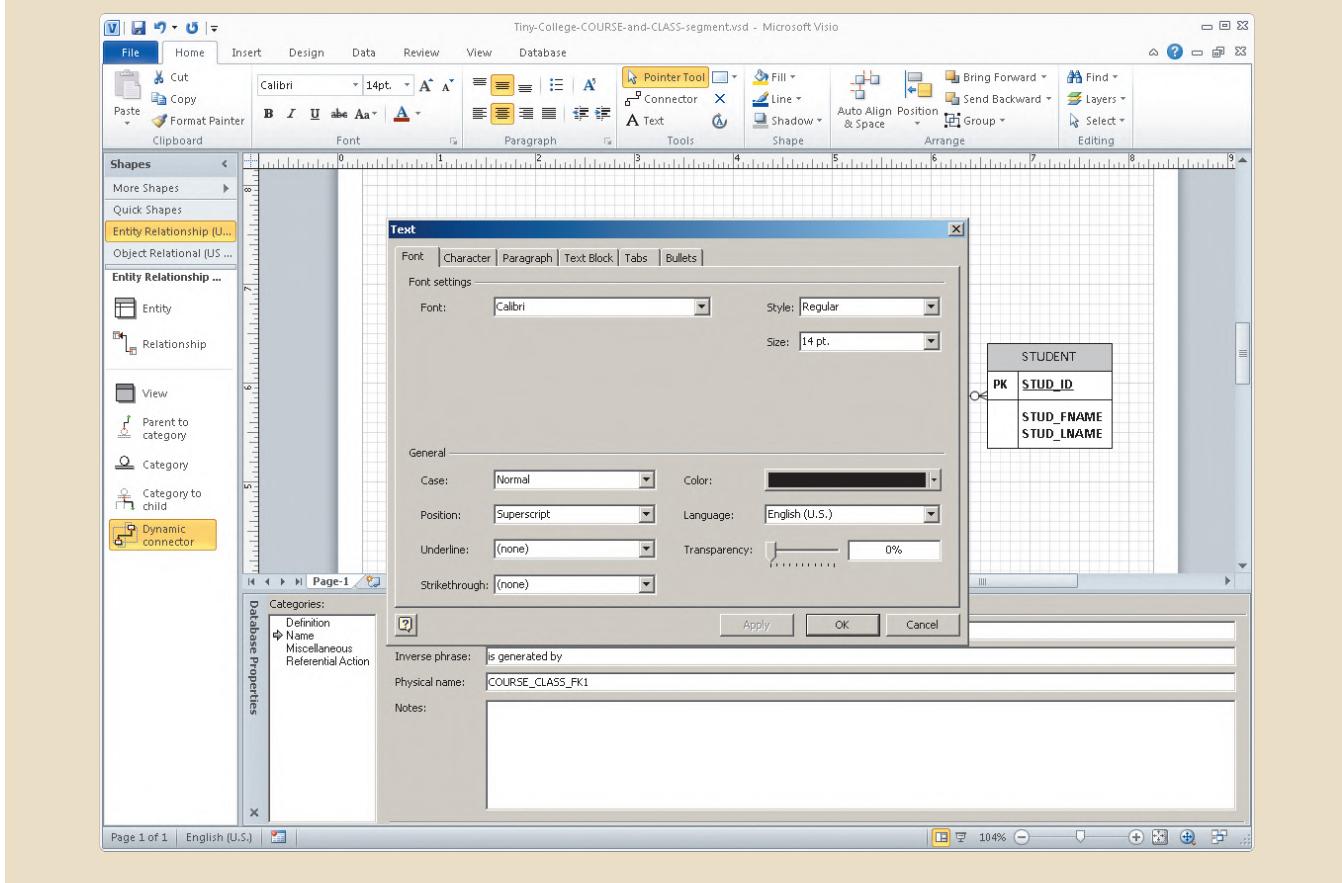
As you can tell by looking at the relationship name in Figure A1.35, it is written through the relationship line, thus making it difficult to read. You can change the placement of the relationship name through font control. For example, if you want to place the relationship name above the relationship line, right-click the relationship and choose the **Format → Text...** selection shown in Figure A1.36.

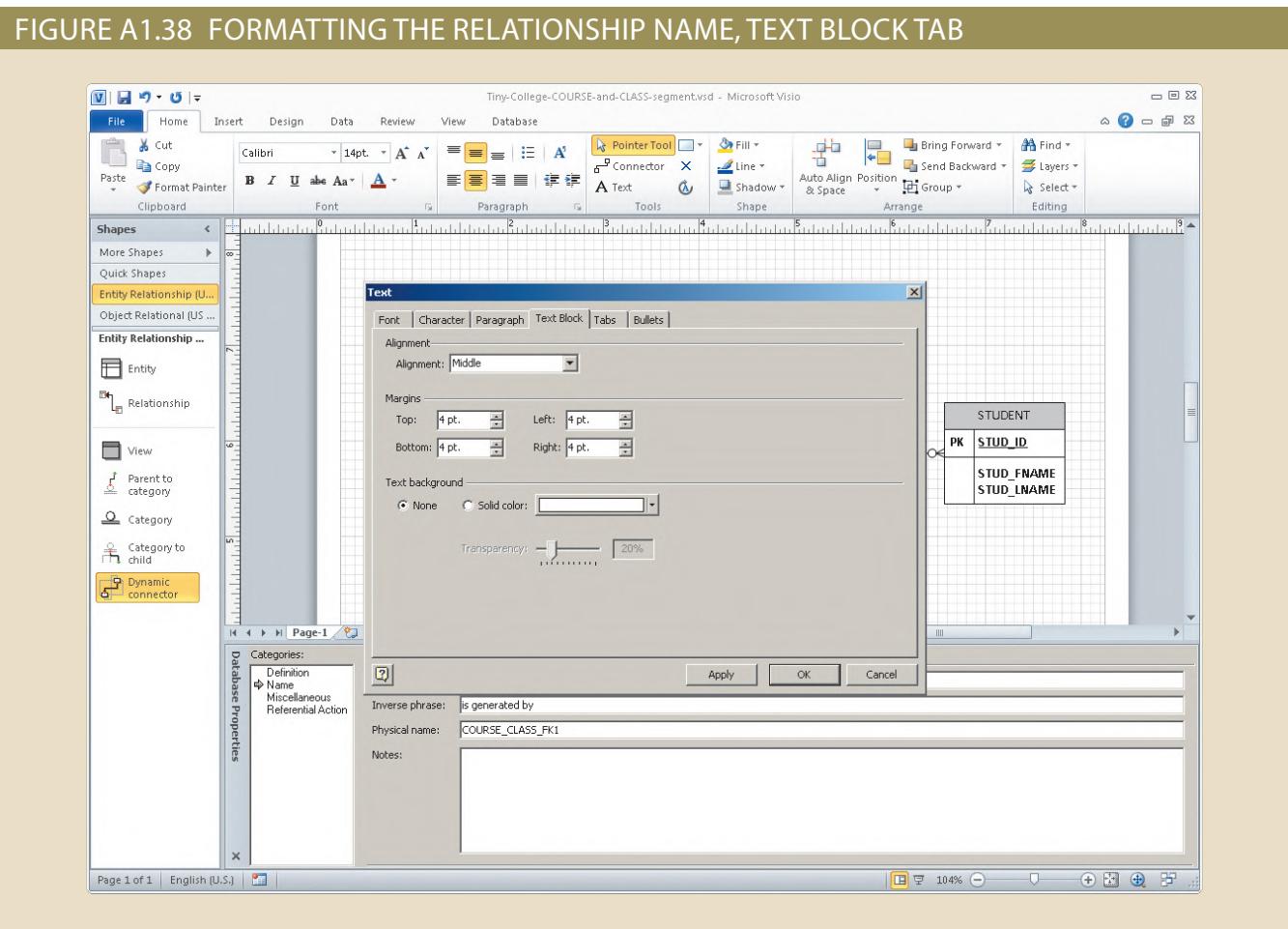
FIGURE A1.36 SELECTING THE RELATIONSHIP NAME TEXT FORMAT



When you click the **Text...** selection shown in Figure A1.36, you will see the window in Figure A1.37. On the **Font** tab, change the **Position** to be **Superscript** and the font **Size** to be **14 pt**. On the **Text Block** tab, change the **Text background** option to **None** as shown in Figure A1.38. Click **OK** to apply these changes. Then make the same changes to the text for the M:M relationship.

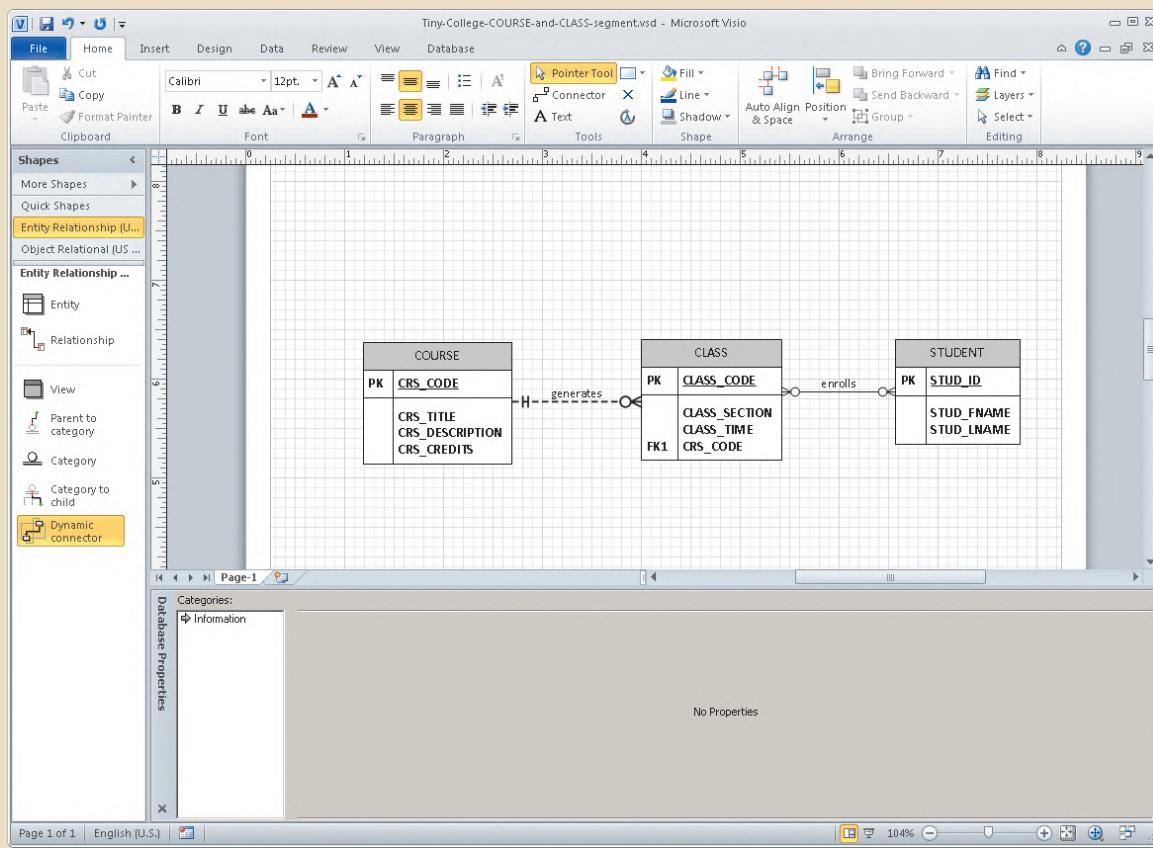
FIGURE A1.37 FORMATTING THE RELATIONSHIP NAME, FONT TAB



A1-38 Appendix A1

After you have made these changes to both relationships, the relationship names will appear above the relationship lines as shown in Figure A1.39. (The relationship line has been deselected by clicking an empty portion of the grid to make it easier to read the repositioned relationship name.)

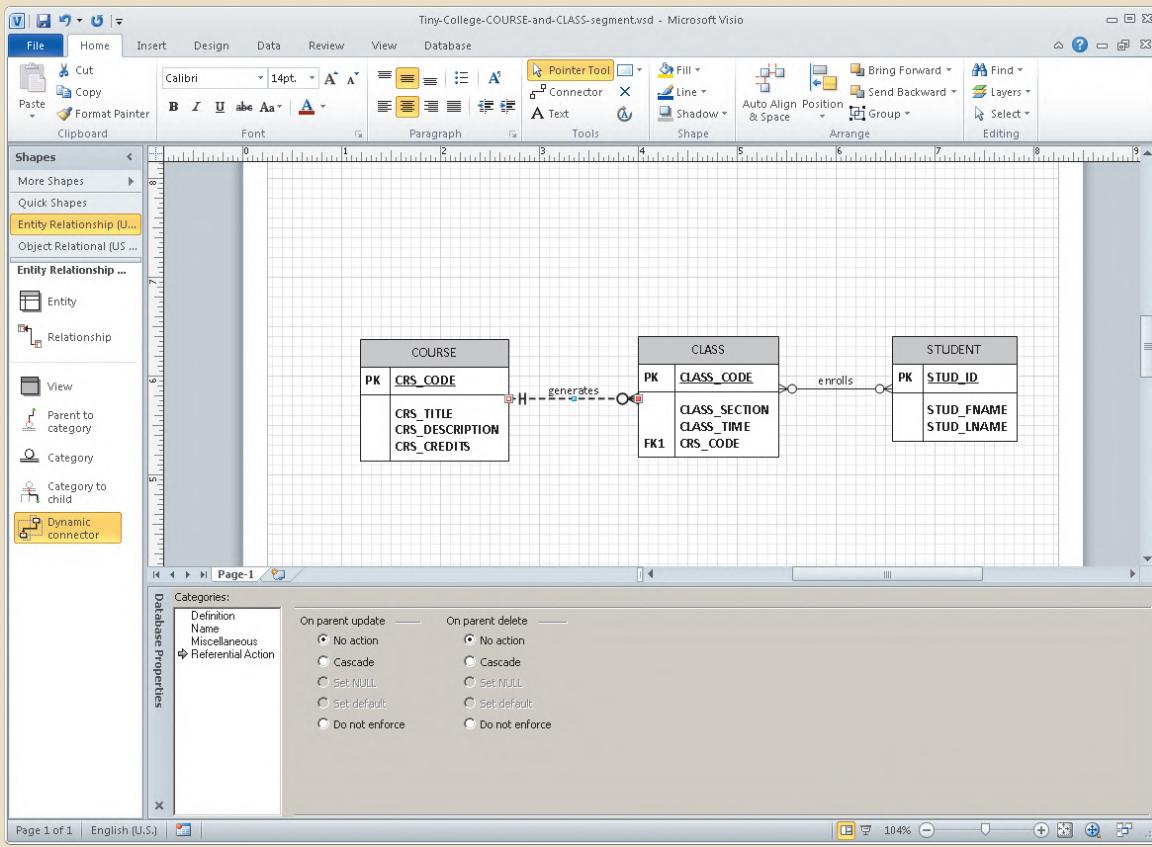
FIGURE A1.39 THE REPOSITIONED RELATIONSHIP NAMES



A1-6 Referential Action

Select the “generates” relationship line, and then click the **Referential Action** option in the **Database Properties** window at the bottom of the screen to produce the results displayed in Figure A1.40.

FIGURE A1.40 THE DEFAULT REFERENTIAL ACTION



As you examine Figure A1.40, think of the consequences of a deletion in the parent (COURSE) table. For example, if a COURSE is deleted, do you want to delete all of the classes that are associated with that course? That is, do you want to **Cascade** the deletion? Similarly, if the CRS_CODE for a row in the COURSE table changes, should that change be reflected in the CRS_CODE of the related rows in the CLASS table? It is important to remember that referential integrity action describes what should be done in the child table in response to changes in the parent table. It does not impact what can be done in the parent table, but how that action is reflected in the related child rows.

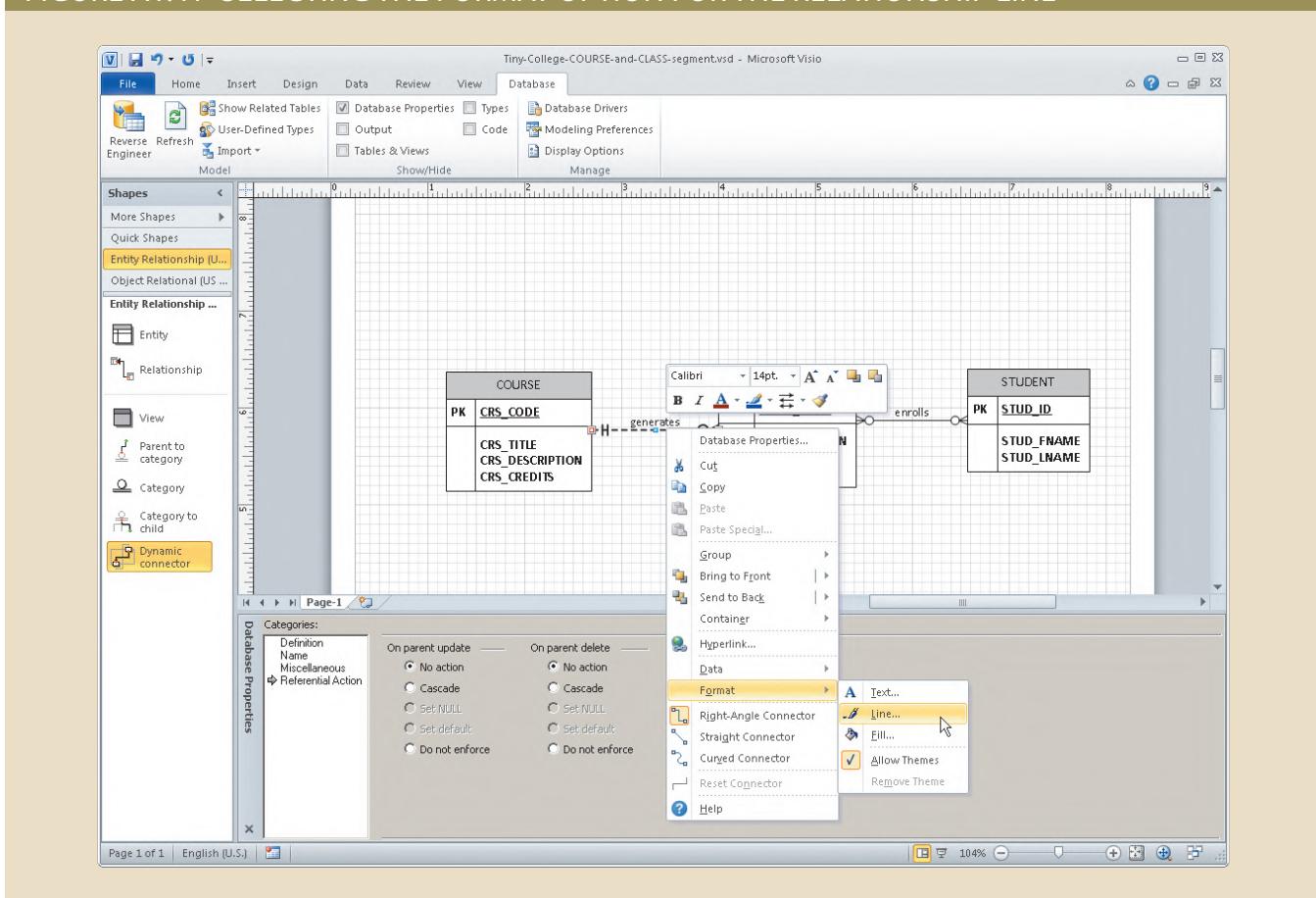
Since our relationship has mandatory participation from CLASS to COURSE (every CLASS must be associated with a COURSE), Visio indicates that we have limited options to maintain referential integrity when rows are deleted or primary keys updated in the parent table. We can not define a referential integrity action (which is the default selection in Visio), we can cascade the change from the COURSE table into the CLASS table, or we can choose not to enforce referential integrity for that action. If the participation from CLASS to COURSE had been optional — that is, if a CLASS could exist without being associated with a COURSE — then Visio would have given the additional options of setting the value of the related FK in the CLASS table to either NULL or some default value when the value of the PK changes in the parent table or when rows are deleted.

The **Referential action** selection forces you to make sure that the database design is appropriate to the data environment and that you really do understand the ramifications of any database action. Given the many action options shown in Figure A1.40, you may want to create a small database and try each action to see its effect.

A1-7 Controlling the ERD's Presentation Format

If you want to modify the ERD presentation format, Visio Professional provides many options. For example, if you want to color the relationship lines dark red, right-click the relationship line, and then select the **Format → Line...** option shown in Figure A1.41.

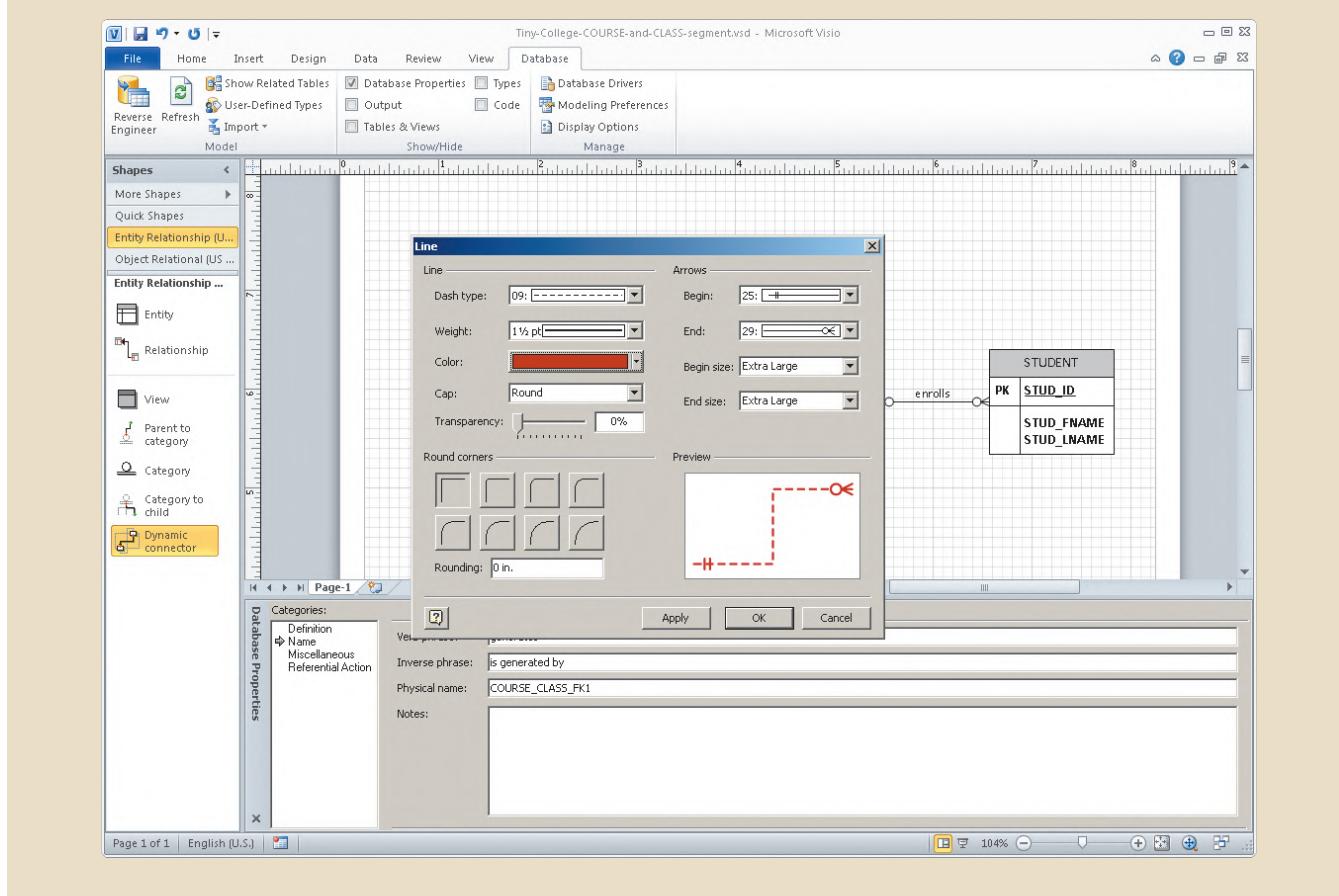
FIGURE A1.41 SELECTING THE FORMAT OPTION FOR THE RELATIONSHIP LINE



When you select the **Line...** option shown in Figure A1.41, you will see the options shown in Figure A1.42. Each selection option has its own drop-down list from which to make a selection. Note that the color **Dark Red** and the line weight **1½ pt.** have been selected. We have left the remaining options in their default settings. Click the **OK** button to accept the format changes shown in Figure A1.42. Make the same changes to the “enrolls” relationship line as well; however, also change the **Dash type** option to **09** so that the relationship line will be a dashed line indicating a non-identifying (weak) relationship.

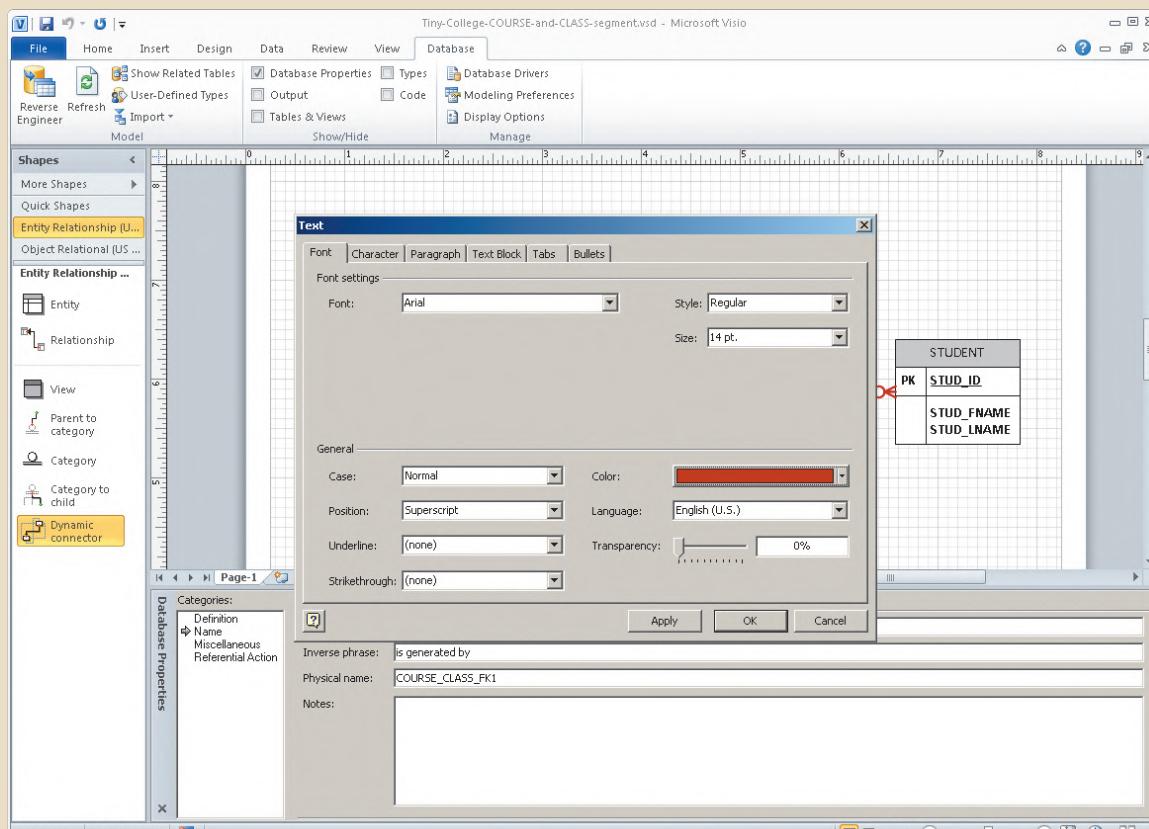
A1-42 Appendix A1

FIGURE A1.42 FORMATTING THE RELATIONSHIP LINE



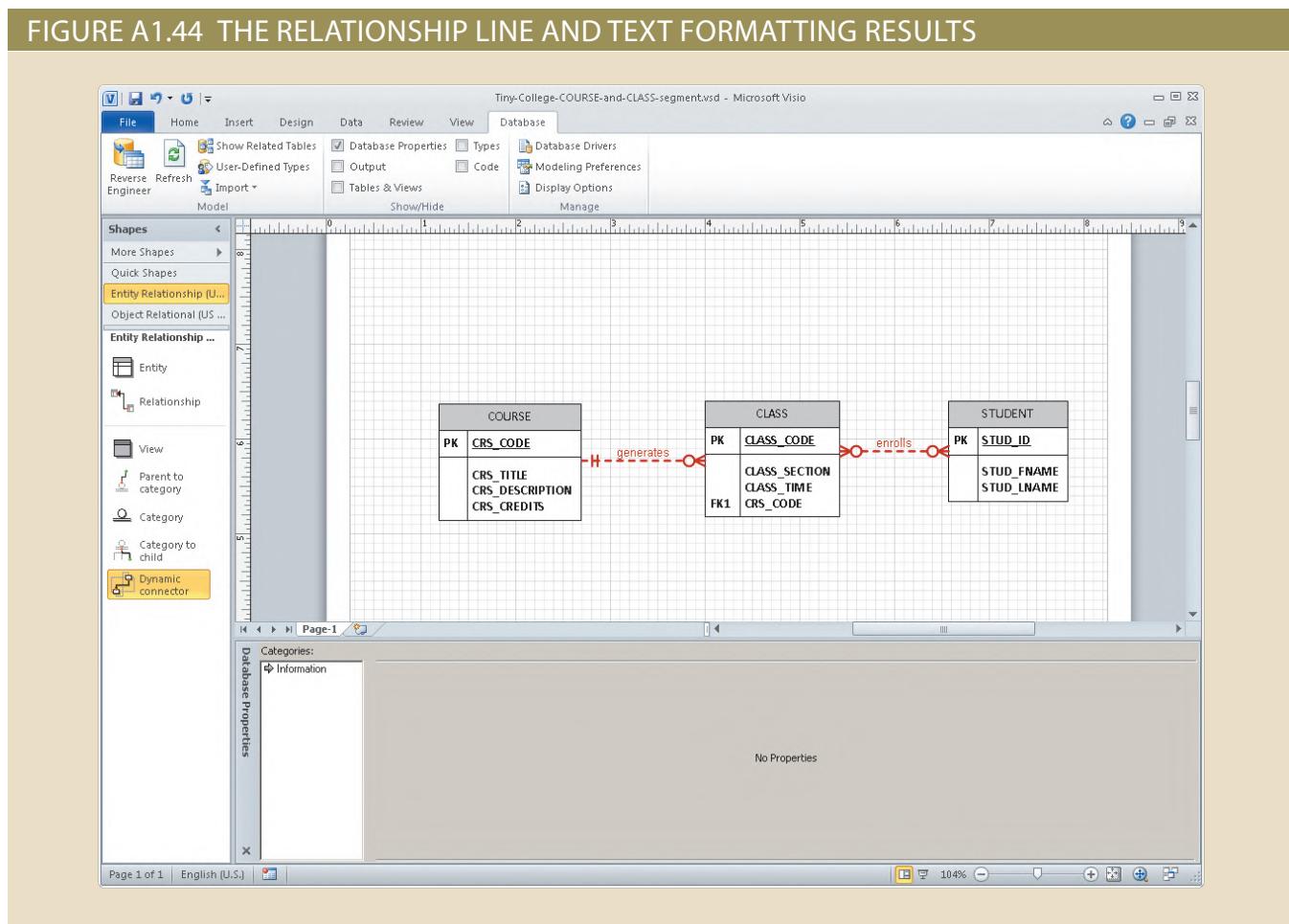
Using the same text formatting window used to reposition the relationship name, we can format the relationship name's text, as shown in Figure A1.43. To do that, select the **Format → Text...** option shown in Figure A1.36 to generate the window displayed in Figure A1.37. Be certain that the **Font** tab is selected. Select **Arial** as the font, and the **Dark Red** text color to match the color of the relationship line. The font **Size: (14 pt.)** and **Position: (Superscript)** reflect the choices made earlier. Make the same changes to the “enrolls” relationship line.

FIGURE A1.43 FORMATTING THE RELATIONSHIP TEXT



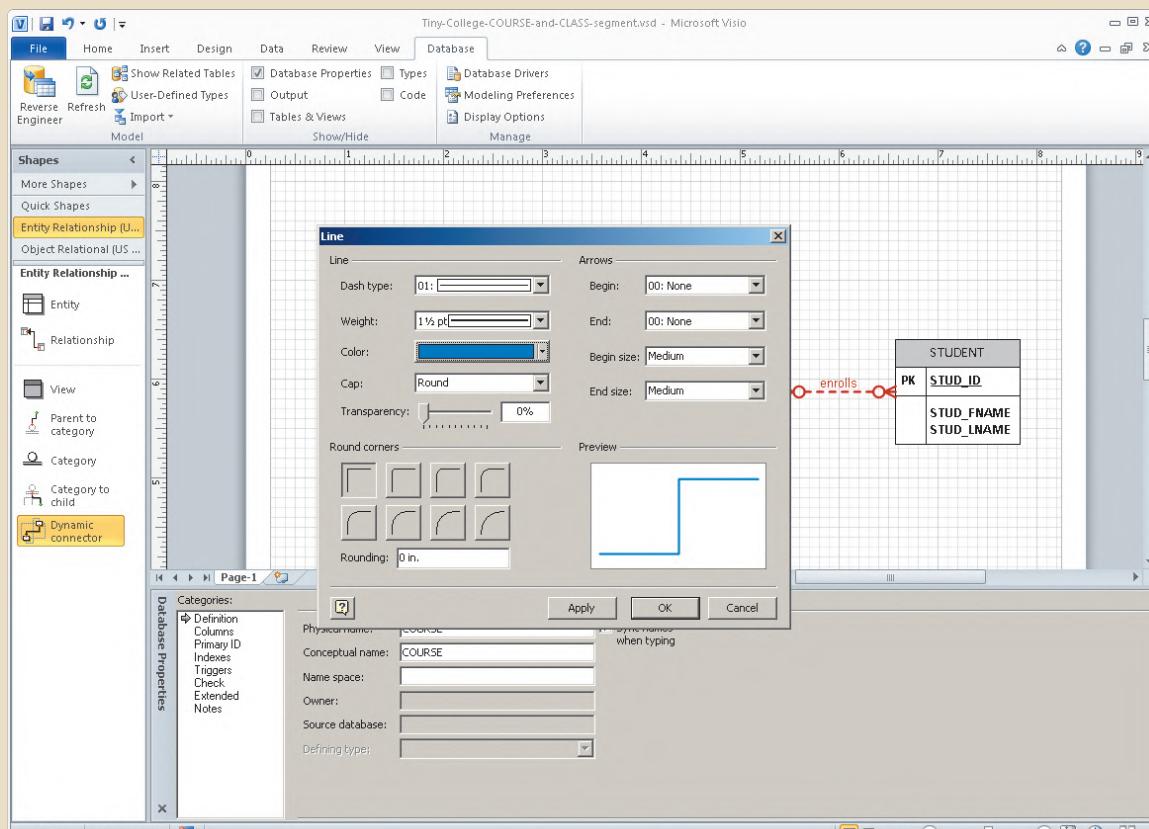
A1-44 Appendix A1

The results of the relationship line and text formatting are shown in Figure A1.44.

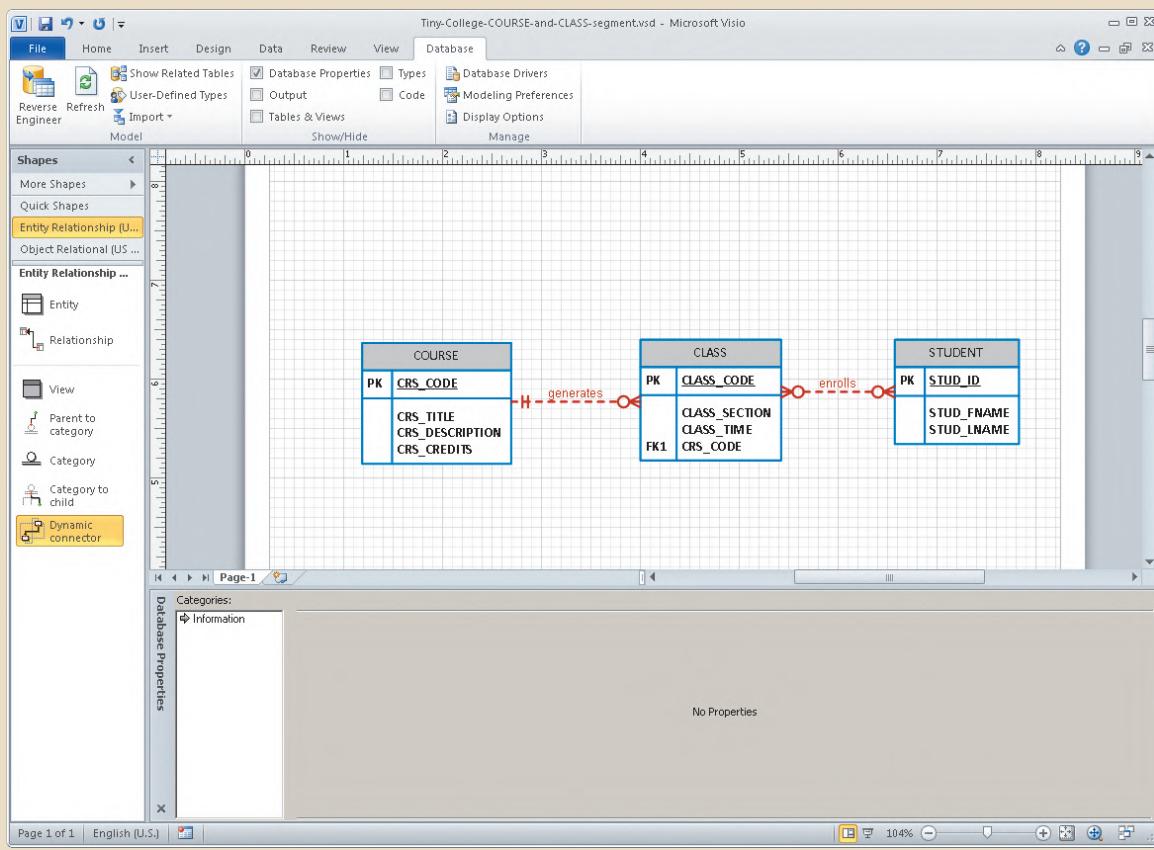


Naturally, you can also control the table's presentation format. To illustrate that process, let's make the table borders blue. To do that, right-click the table you want to format. Then select the format option (**Format → Line...**) to generate the line options shown in Figure A1.45.

FIGURE A1.45 FORMATTING THE TABLE LINE



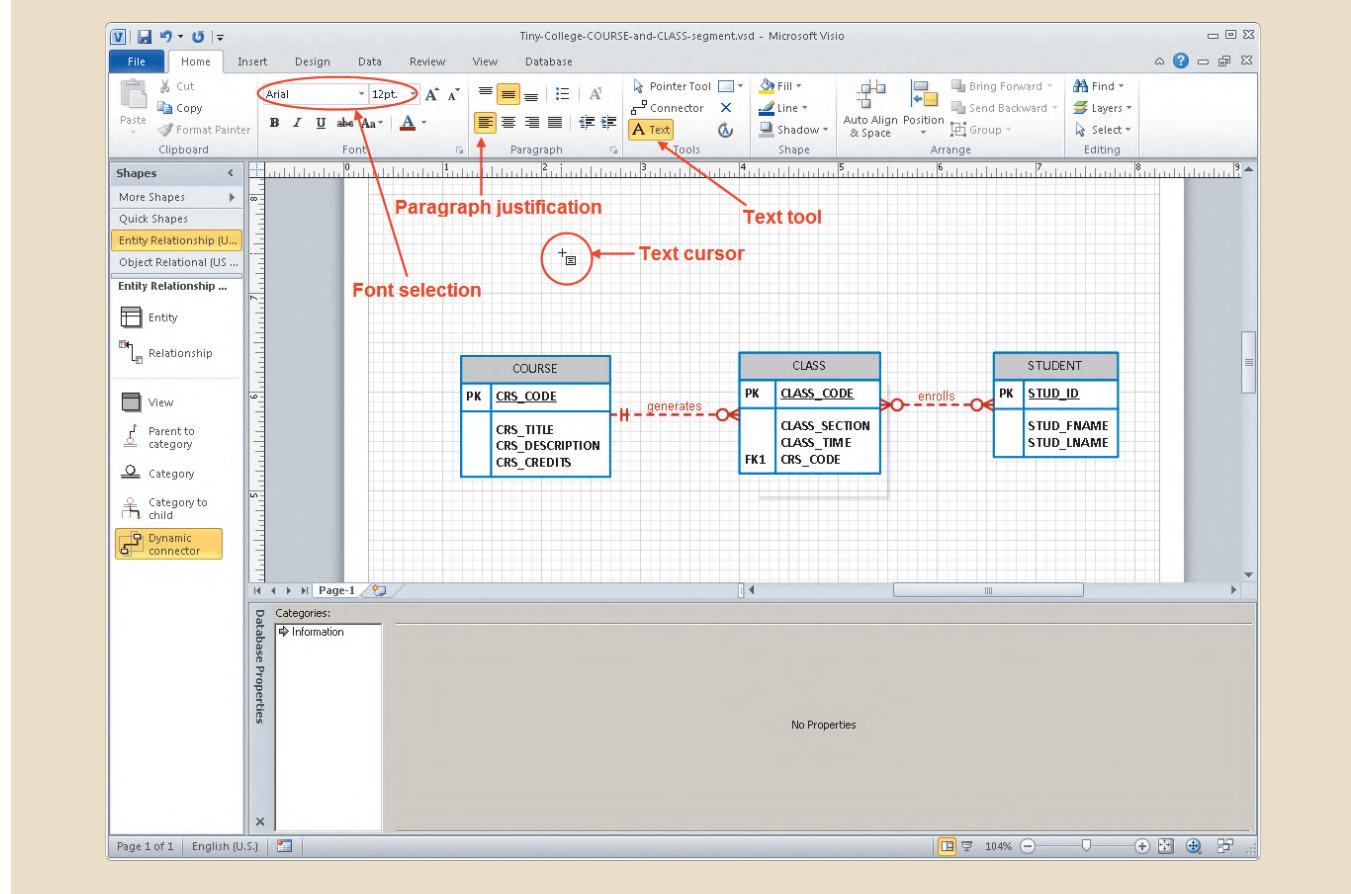
As you examine Figure A1.45, note that the line **Color:** was selected to be **Blue**. The selected line **Weight:** is **$1\frac{1}{2}$ pt**. Remember to click the **OK** button to save the changes. Now repeat the process for the CLASS and STUDENT tables to produce the results shown in Figure A1.46.

A1-46 Appendix A1**FIGURE A1.46 THE REFORMATTED TABLE LINES**

A1-8 Placing Text on the Grid

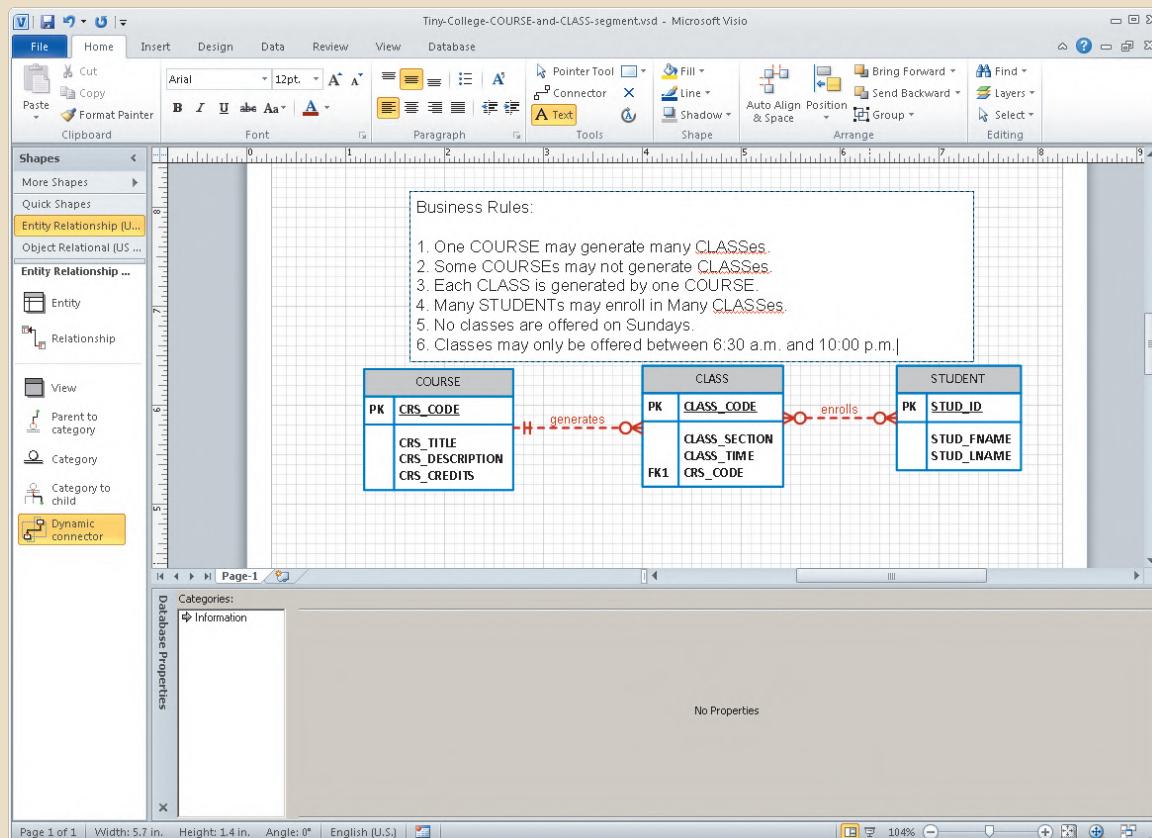
To help document the ERD, it may be helpful to place explanatory notes on the grid. Make sure that you have not selected any object by clicking a blank area of the screen. Select the **Text tool** (marked A) shown at the top of the screen on the **Home** ribbon. You will see the effect of your selection when you note the cursor's new look. Select the text format to suit your needs—left justification and a font face and size of Arial 12pt have been selected in Figure A1.47.

FIGURE A1.47 SELECTING THE TEXT TOOL



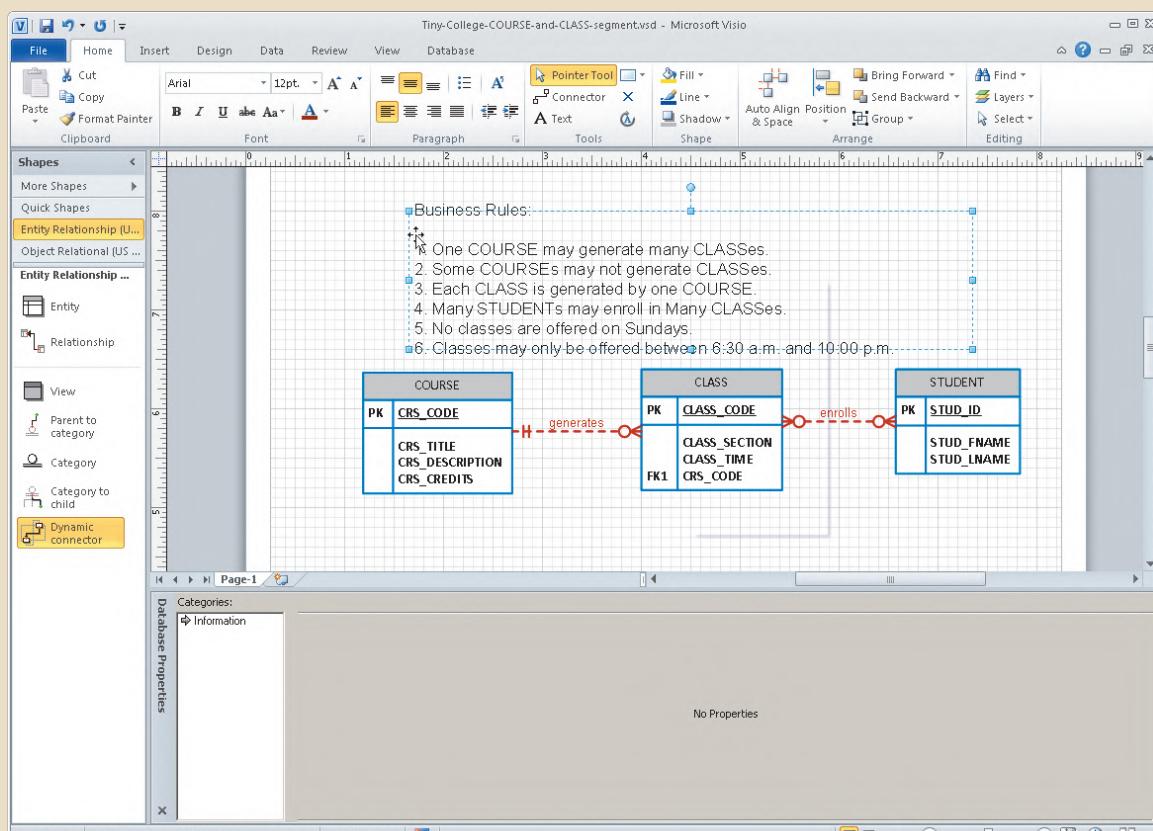
After making the selections shown, enter the text as shown in Figure A1.48. (You can modify any text format such as the font, size, color, and justification later.)

FIGURE A1.48 THE INITIAL TEXT



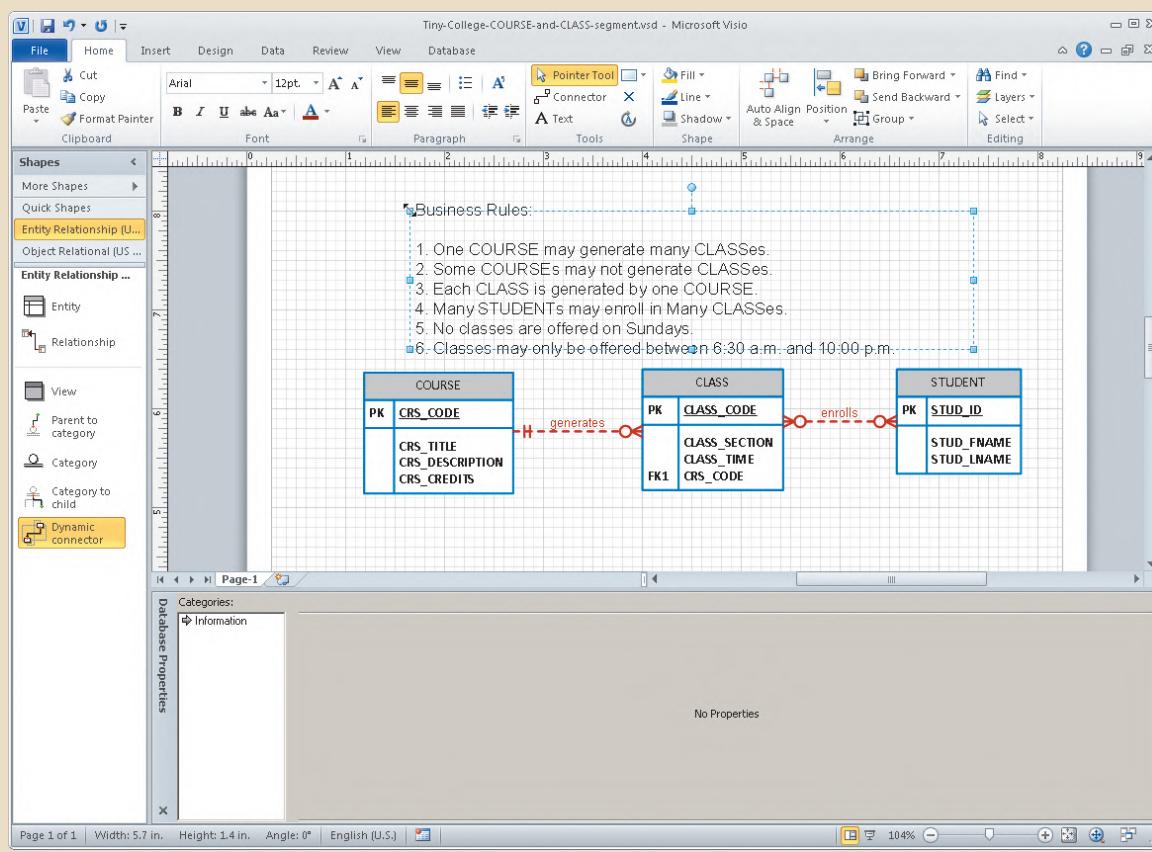
To move the text box, you must first make sure that the text tool has been deselected. If the text tool is still active, click the pointer tool. (You will know that the text tool is active when the cursor looks like the one shown in Figure A1.47.) If the text tool is not active, clicking the text box produces a set of small squares (handles) shown on the text box perimeter. You can see the handles around the text box in Figure A1.49. Also note that the four-sided arrow by the cursor indicates that the text box may be moved by dragging and dropping. (If you don't see the four-sided arrow on your screen, move the cursor until the four-sided arrow appears.)

FIGURE A1.49 SELECTING THE TEXT BOX TO MOVE IT



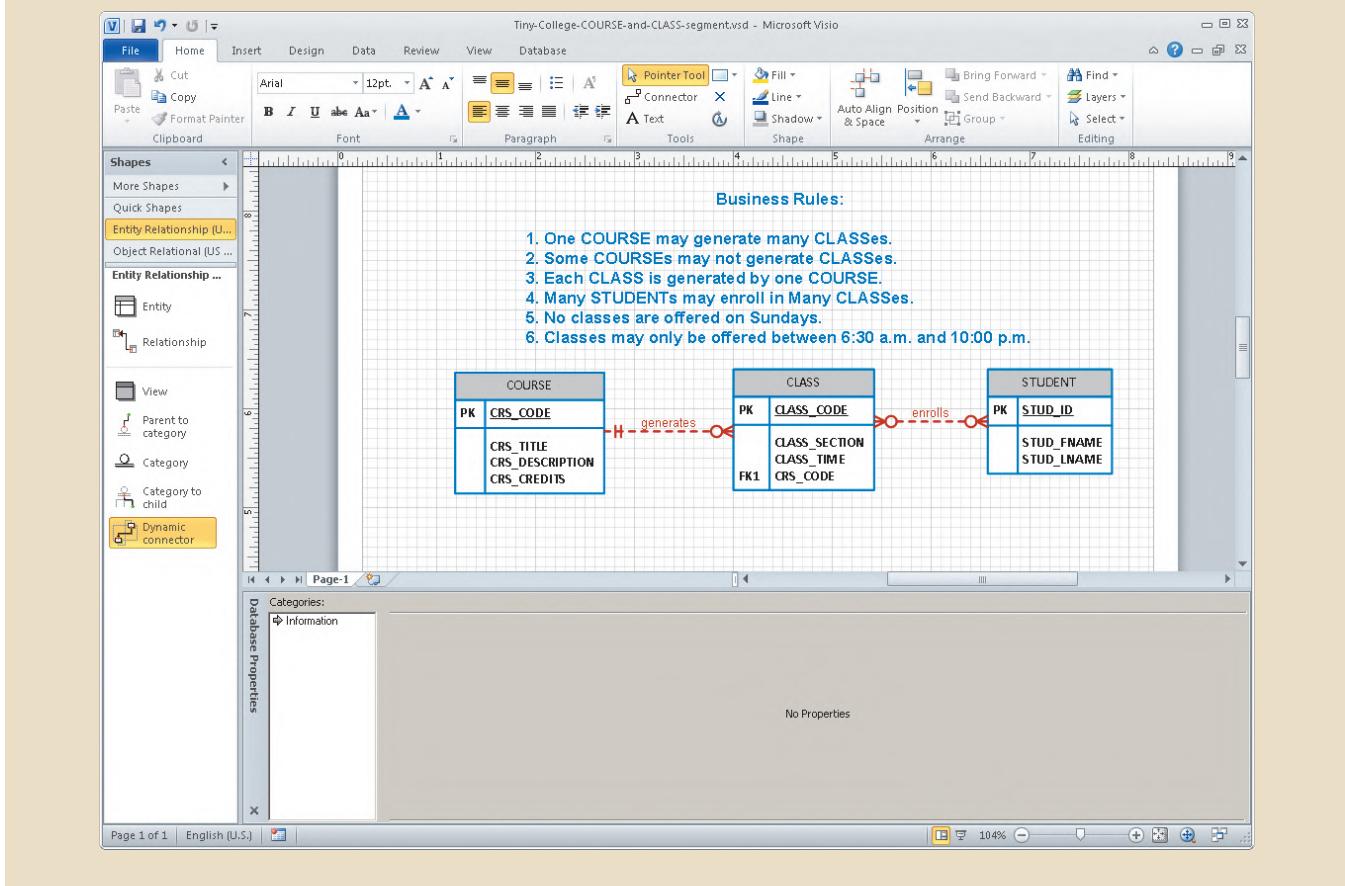
A1-50 Appendix A1

FIGURE A1.50 SELECTING THE TEXT BOX TO SIZE IT



You can now finish typing the text, formatting it to suit your needs. The final text box is shown in Figure A1.51. Note that a blue text color and boldface has been selected.

FIGURE A1.51 THE COMPLETED TEXT BOX



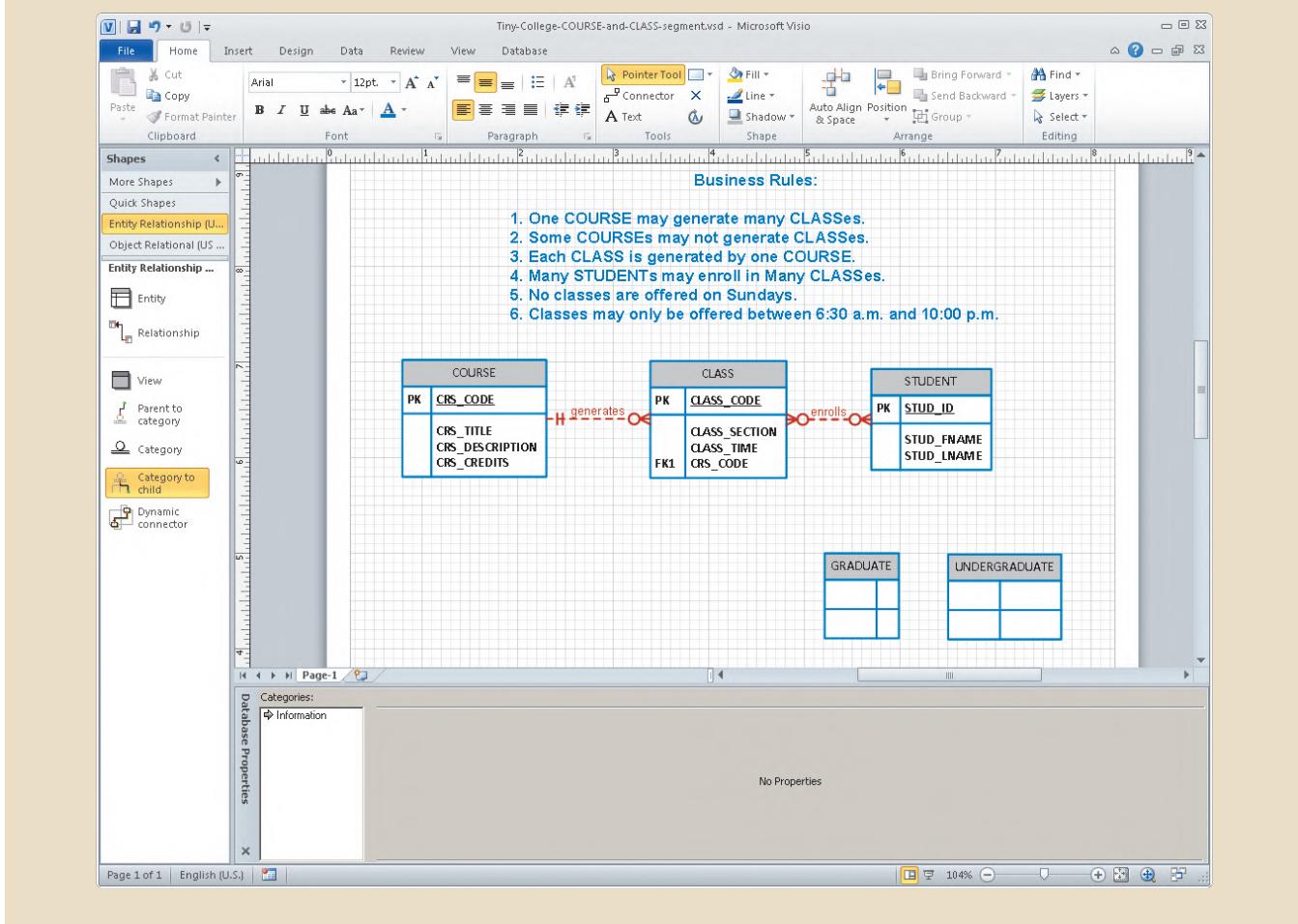
Don't forget to save your Visio file before you exit. As with all Windows applications, you will be reminded to save the file if you try to close it without first saving it.

A1-9 Using MS Visio 2010 to Create a Specialization Hierarchy

Specialization hierarchies are used to depict a special type of relationship as discussed in Chapter 5, Advanced Data Modeling. These relationships rely on supertype and subtype entities. In Visio, these entities are created and manipulated in exactly the same way as the entities created previously in this tutorial. The only thing different about the specialization hierarchy is the modeling of the relationship.

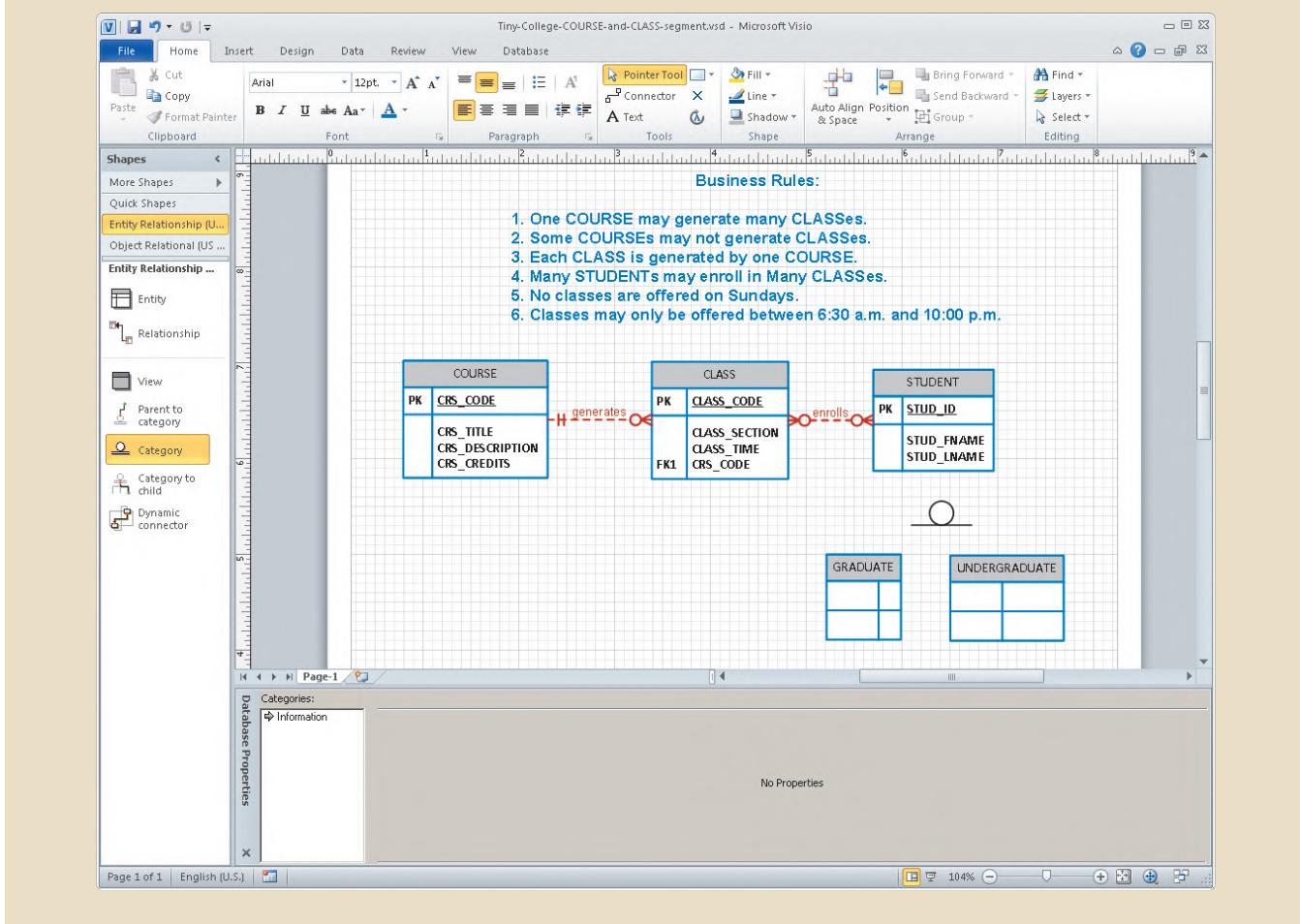
To begin, add two new entities named GRADUATE and UNDERGRADUATE to the existing data model as shown in Figure A1.52. These entities will be subtypes of STUDENT.

FIGURE A1.52 ADDING GRADUATE AND UNDERGRADUATE SUBTYPES



Visio uses Category notation to indicate a specialization hierarchy. The Category symbol is composed of a circle on top of one or two lines and is added to the diagram by dragging the **Category** object from the left side of the screen and dropping it on the drawing grid. Place a Category symbol between the STUDENT supertype entity and the subtype entities as shown in Figure A1.53.

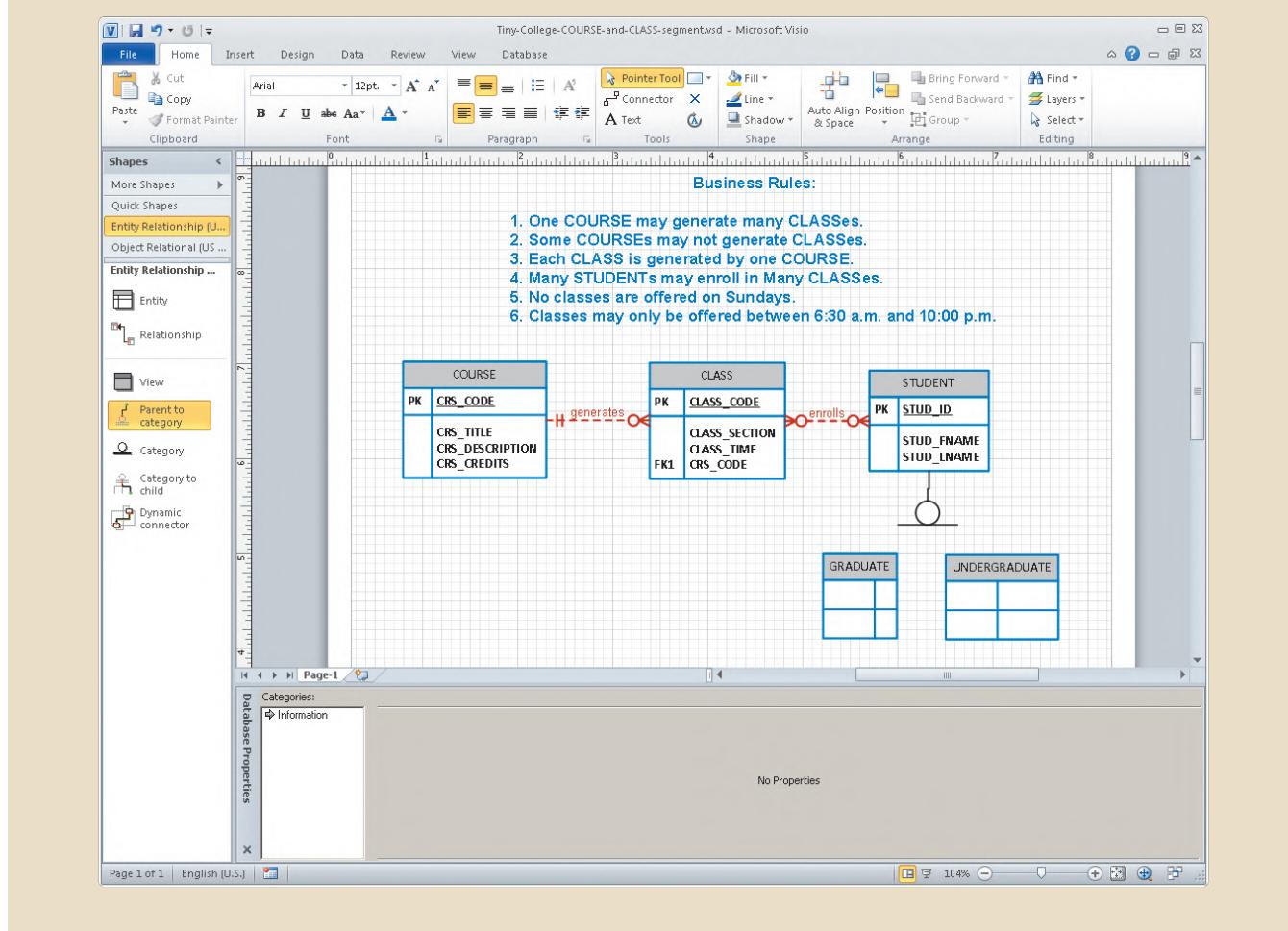
FIGURE A1.53 PLACING THE CATEGORY SYMBOL



The specialization hierarchy is modeled by relating the supertype entity, also called the parent entity, and the subtype entities, also called the child entities, to the Category object. Depicting these relationships on the diagram is the same as depicting other relationships except that instead of using the **Relationship** object or the **Dynamic connector** object, the **Parent to category** object is used to relate the supertype to the category and the **Category to child** object is used to relate the subtypes to the category. Use the **Parent to category** object to relate STUDENT to the category symbol as shown in Figure A1.54.

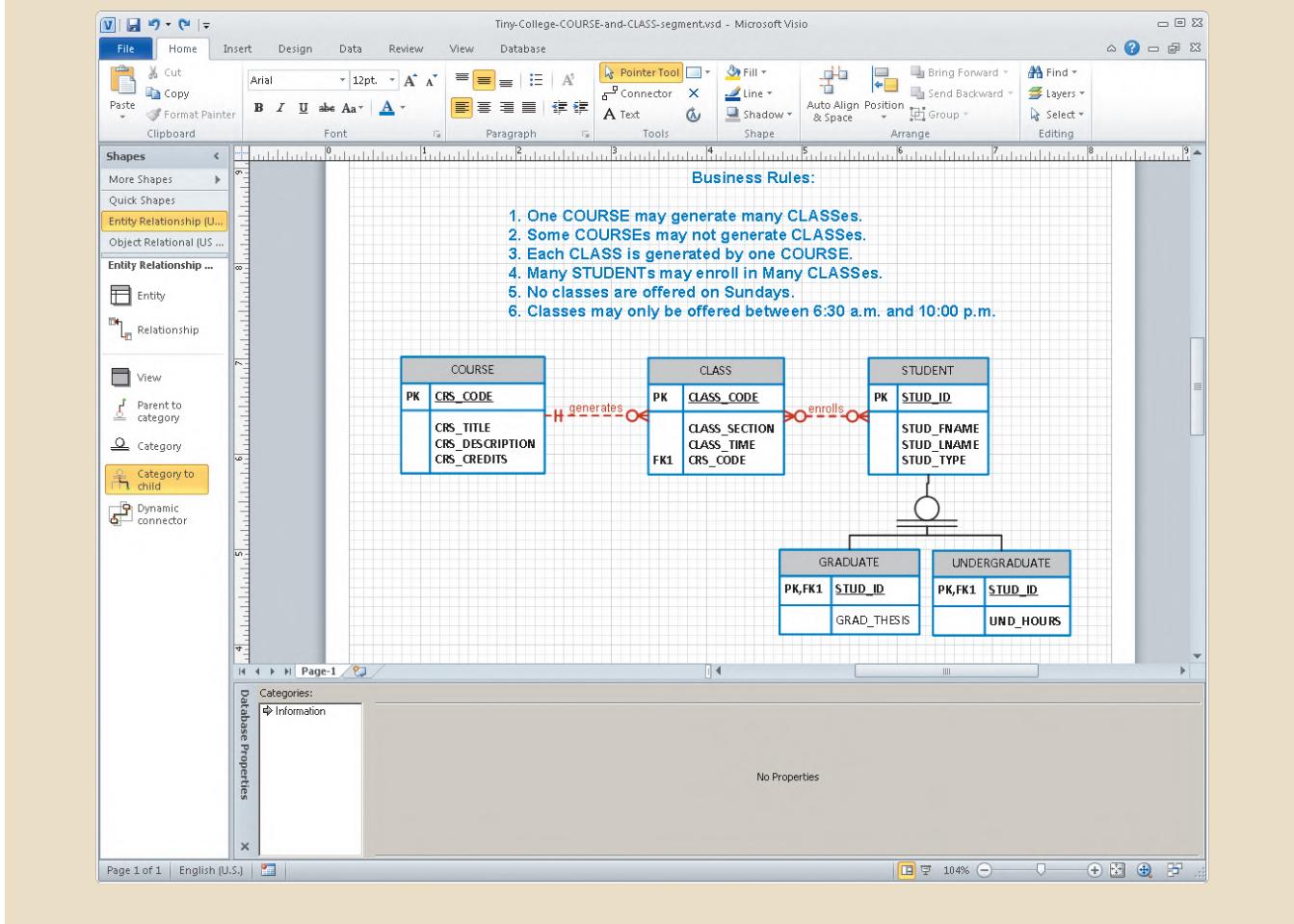
A1-54 Appendix A1

FIGURE A1.54 RELATING THE SUPERTYPE



Use the **Category to child** object to relate the category to GRADUATE. Then, use **Category to child** again to relate the category to UNDERGRADUATE as shown in Figure A1.55. Note that when you relate the subtypes to the category, Visio automatically (and correctly) places the PK of the supertype as both the primary key and a foreign key in the subtypes.

FIGURE A1.55 RELATING THE SUBTYPES

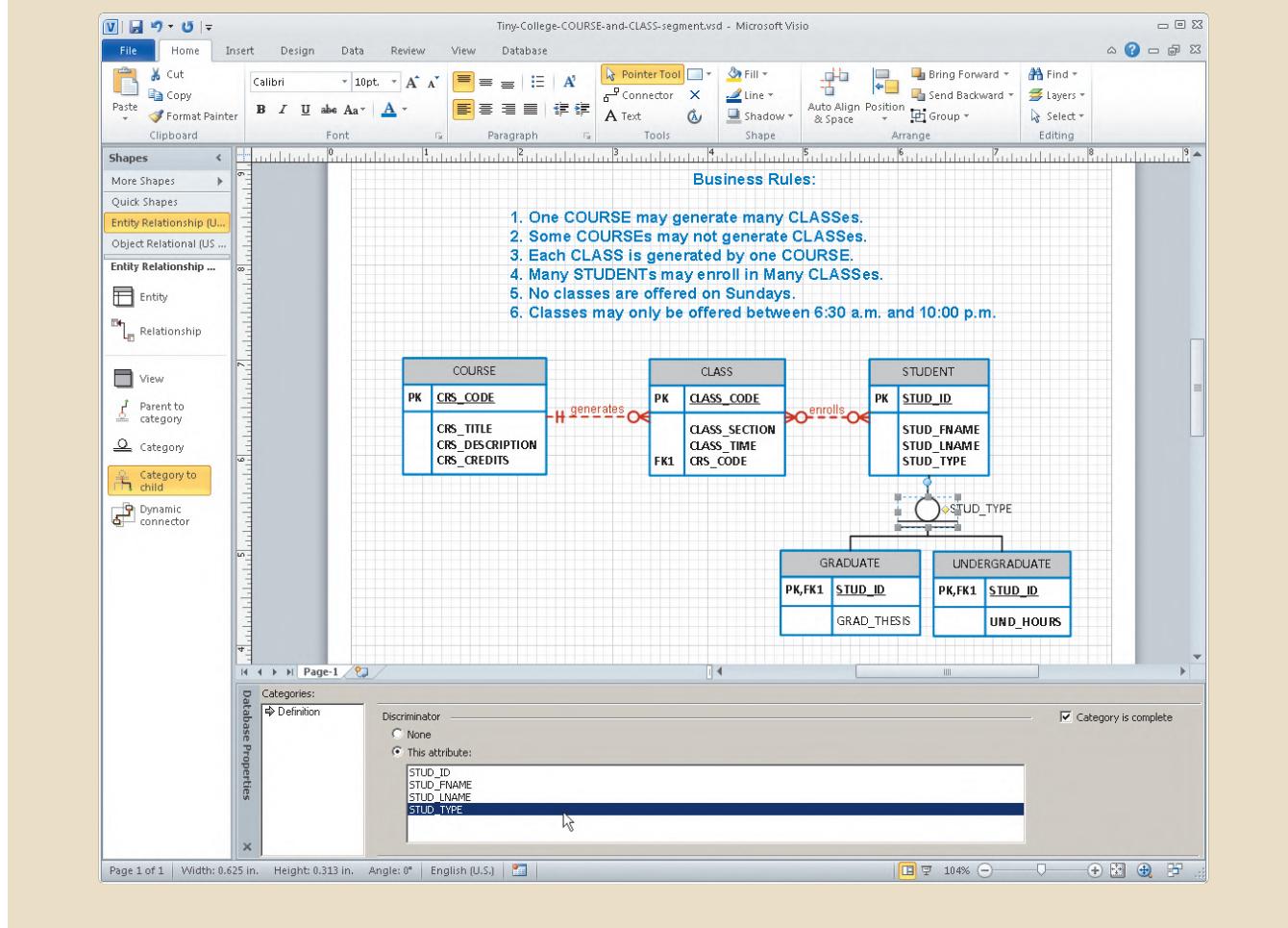


Both the supertype and subtypes can be edited to add additional attributes as necessary. This is done using exactly the same technique as manipulating the attributes of any entity. Add a required STUD_TYPE attribute to the STUDENT entity to serve as a subtype discriminatory. Also, add GRAD_THESIS to the GRADUATE entity, and required UND_HOURS attribute to the UNDERGRADUATE entity.

The specialization hierarchy can be modeled as requiring total completeness or partial completeness. Recall from Chapter 5 that a value of “total” for the completeness constraint means that every entity instance in the supertype must be a member of at least one subtype, while a value of “partial” for the complete constraint means that it is optional for an instance of the supertype to be a member of any subtype. By default, the category symbol indicates a partial completeness constraint — it has a single line at the bottom of the symbol. Click the category symbol to select it, then in the **Database Properties** window at the bottom of the screen, click to mark that the **Category is complete**. Also in the **Database Properties** window, note that you can choose the subtype discriminator for the specialization hierarchy. Select **This attribute:** and choose STUD_TYPE for the list of supertype entity attributes as shown in Figure A1.56.

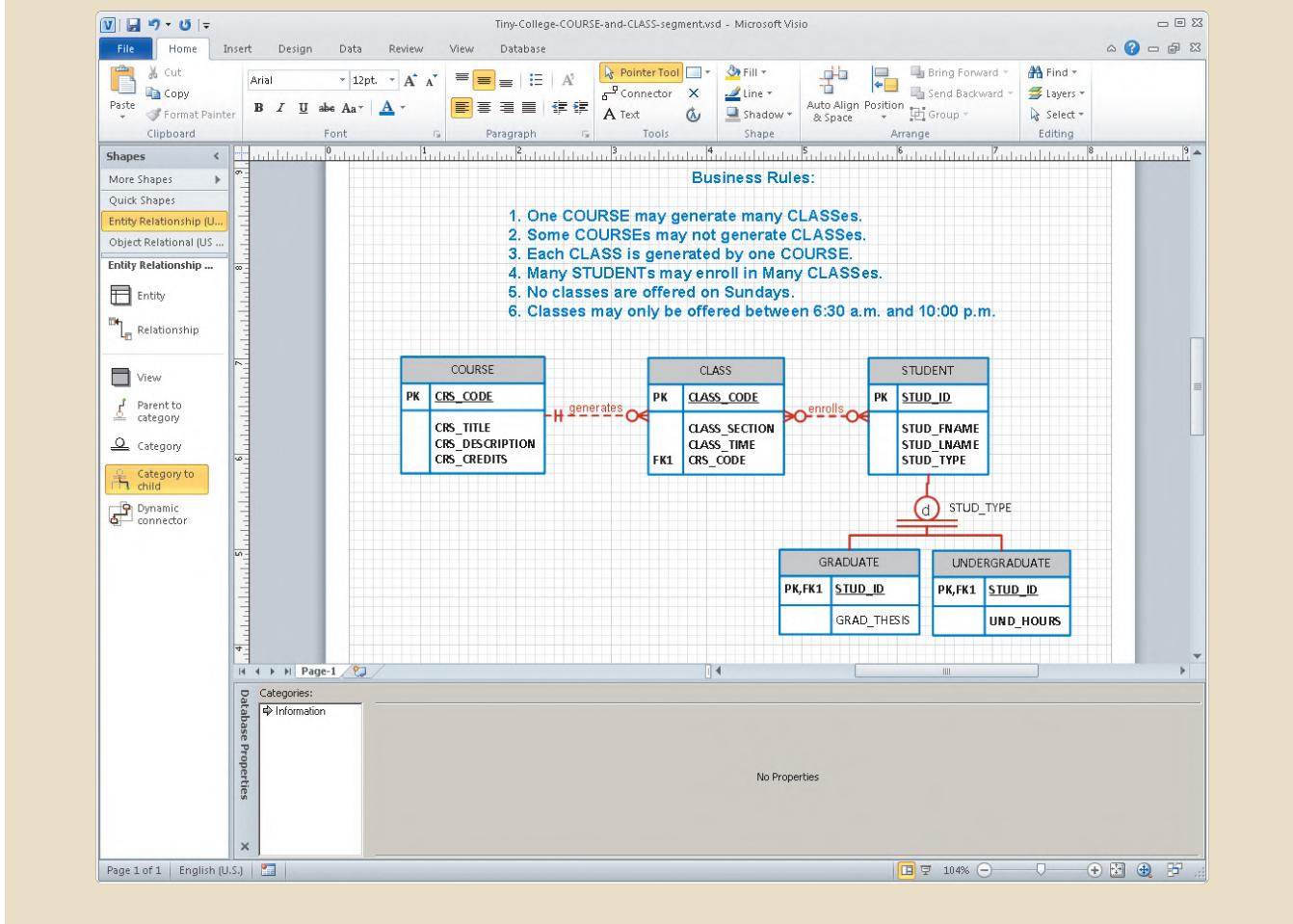
A1-56 Appendix A1

FIGURE A1.56 CATEGORY OPTIONS



Visio does not support adding text to category symbols. Therefore, to place the indicator for whether the subtypes are disjoint or overlapping, use the Text tool to place a “d” or “o” on the category symbol. The category symbol and the relationship lines that relate the supertype and subtypes to it can all be formatted using the same techniques previously described to format the other entities and relationships. The completed diagram is shown in Figure A1.57.

FIGURE A1.57 THE COMPLETED ERD



Appendix A2

Designing Databases with Visio 2016: A Tutorial

Microsoft Visio 2016 includes database design and modeling tools among many other diagramming options. The Visio software has so many features that it is impossible to demonstrate all of them in this short tutorial. However, you will learn how to:

- Select the Crow's Foot entity relationship diagram (ERD) option.
- Create the entities and define their components.
- Create the relationships between the entities and define the nature of those relationships.
- Edit the Crow's Foot ERDs.
- Insert text into the design grid and format the text.

Preview

Once you have learned how to create a Visio Crow's Foot ERD, you will be sufficiently familiar with the basic Visio software features to experiment on your own with other modeling and diagramming options. You will also learn how to insert text into the Visio diagram to document features you consider especially important or to simply provide an explanation of some segment of the ERD.

Data Files and Available Formats

[MS Access](#) [Oracle](#) [MS SQL](#) [My SQL](#)

[MS Access](#) [Oracle](#) [MS SQL](#) [My SQL](#)

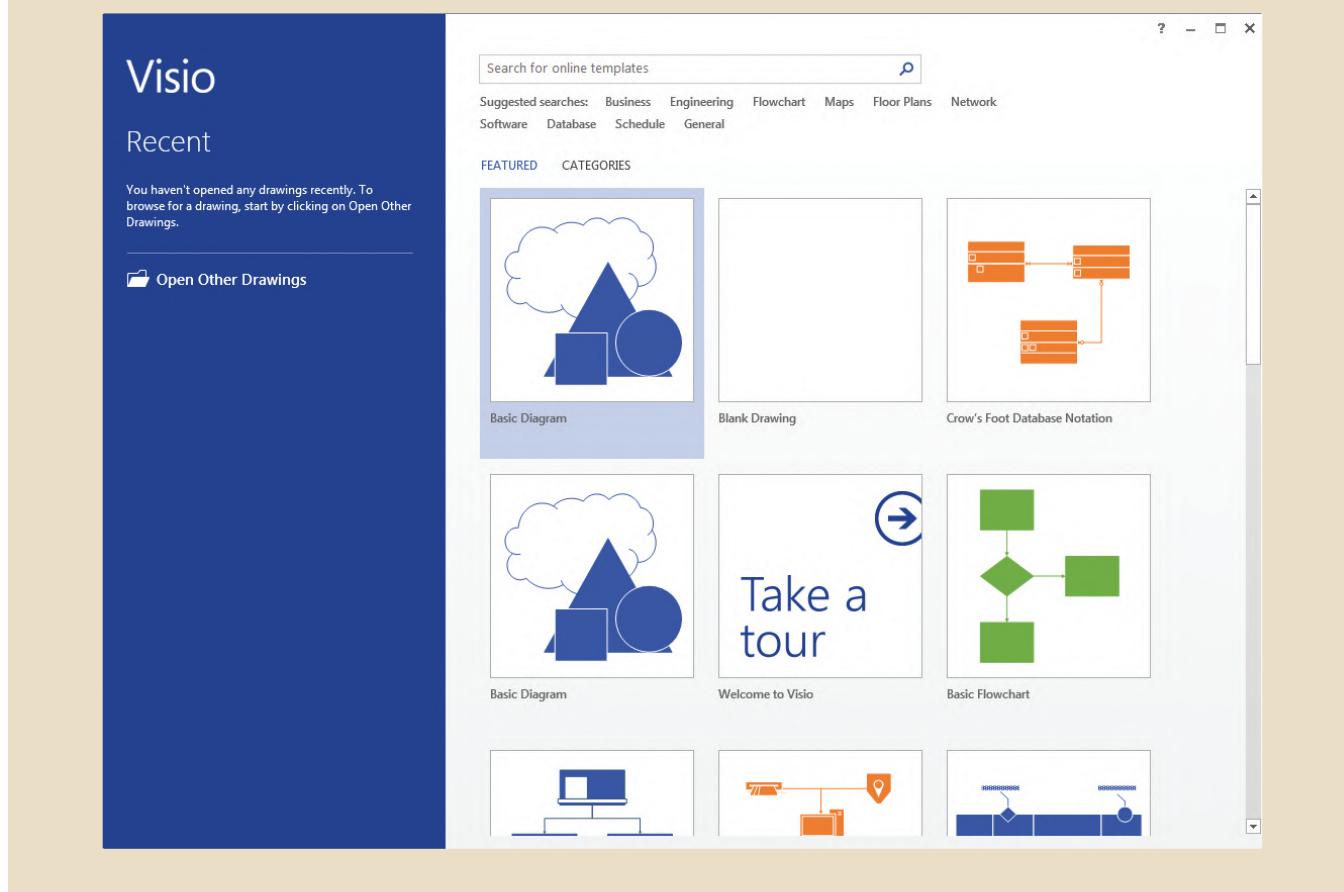
There are no data files for this appendix.

Data Files Available on cengagebrain.com

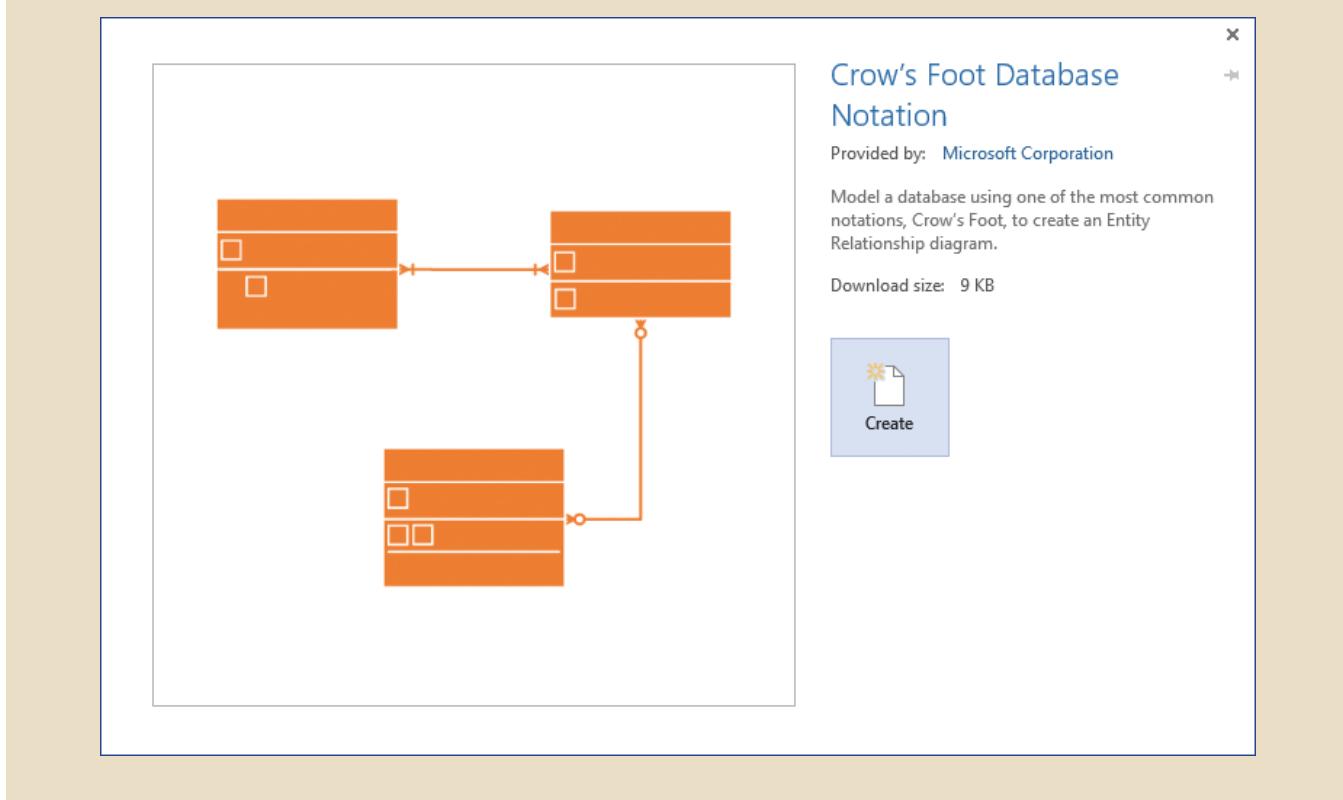
A2-1 Starting Visio

The Visio software on your computer is started in the same fashion as other programs. The exact procedure is dependent on the operating system being used. After the Visio software has been activated, the start screen shown in Figure A2.1 will display. Previously created Visio files show up under the **Recent** header on the left side of the screen.

FIGURE A2.1 THE VISIO PROFESSIONAL START SCREEN



The option to create a Crow's Foot Database Notation diagram should be presented to you. However, if you do not see this option, you can click the **Database** selection at the top of the screen (visible in Figure A2.1). This will display the set of database modeling options available, and you can select Crow's Foot Database Notation from this window. Once the window shown in Figure A2.2 appears, you can click the large **Create** button to begin a new Crow's Foot diagram.

FIGURE A2.2 CROW'S FOOT DATABASE NOTATION SELECTION

When you begin a new diagram, the screen shown in Figure A2.3 will appear. Because the preference here is for a larger grid than the one shown in Figure A2.3, start by selecting the **VIEW** tab on the Ribbon at the top of the screen. Click the **Zoom** option to generate the list of size options. Figure A2.4 shows that the **100%** option has been selected. When you choose the **100%** selection and click **OK**, the grid expands to fill the screen.

A2-4 Appendix A2

FIGURE A2.3 THE DRAWING BOARD

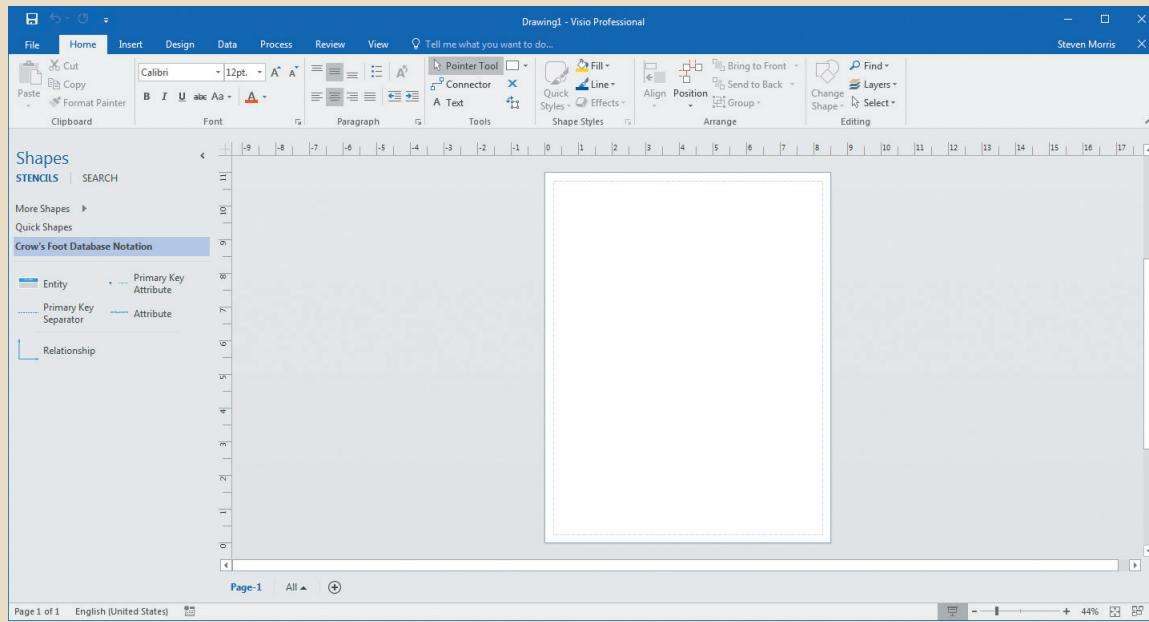
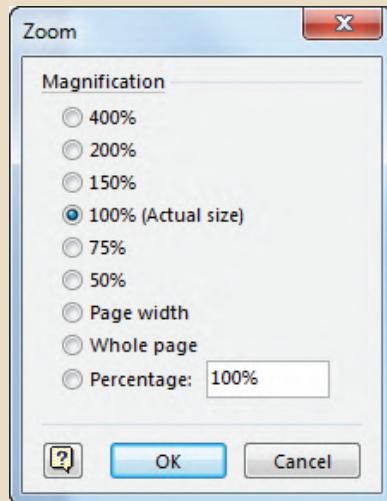


FIGURE A2.4 THE DRAWING BOARD SIZE OPTION



By selecting the Visio Professional database option and its drawing board, you have completed the preliminary work required to create ERDs. You are now ready to draw the ERDs on the drawing board. You will use the Crow's Foot option, the same one used to create all of the ERDs in this text.

A2-2 Creating a Crow's Foot ERD

To illustrate the development of the Visio Professional's Crow's Foot ERD, you will create a simple design based on the following business rules:

- A course can generate many classes.
- Each class is generated by a course.
- A course may or may not generate a class.
- A student can enroll in many classes.
- A student may or may not have enrolled in any class.
- A class can have many students in it.
- A class may or may not have had any students enroll in it.

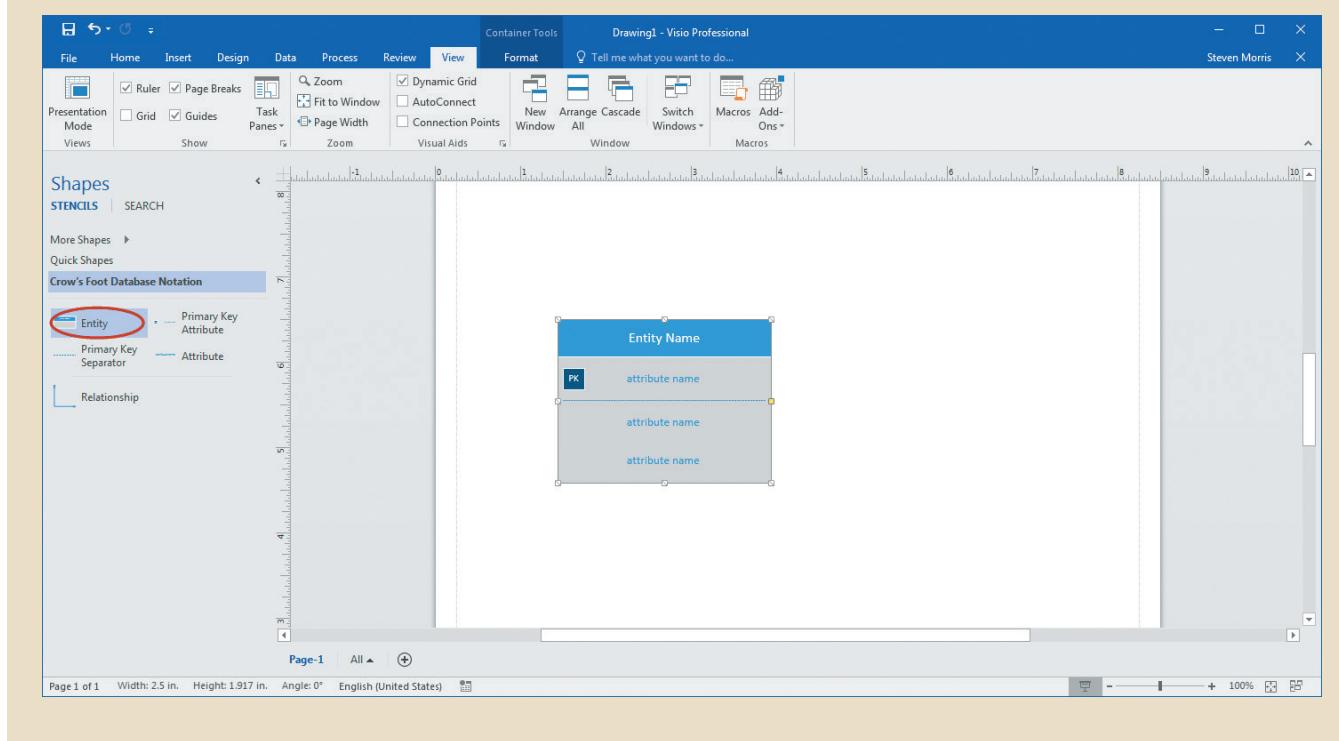
Note that a class has been defined as a section of a course. That definition reflects the real world's use of the labels *class* and *course*. Students have a *class* schedule rather than a section schedule. The catalog that lists all of the courses offered by a department is called a *course catalog*. Some courses are not taught each semester, so they may not generate a class during any given semester.

A2-2a Creating an Entity

Now that you have some idea of the proposed design components, let's create the first entity for the design. Click the **Entity** object shown in Figure A2.5. (It is circled in red in the figure.) Drag the **Entity** object to the grid and then drop it. That action will produce the **Entity Name** object shown in the grid in Figure A2.5.

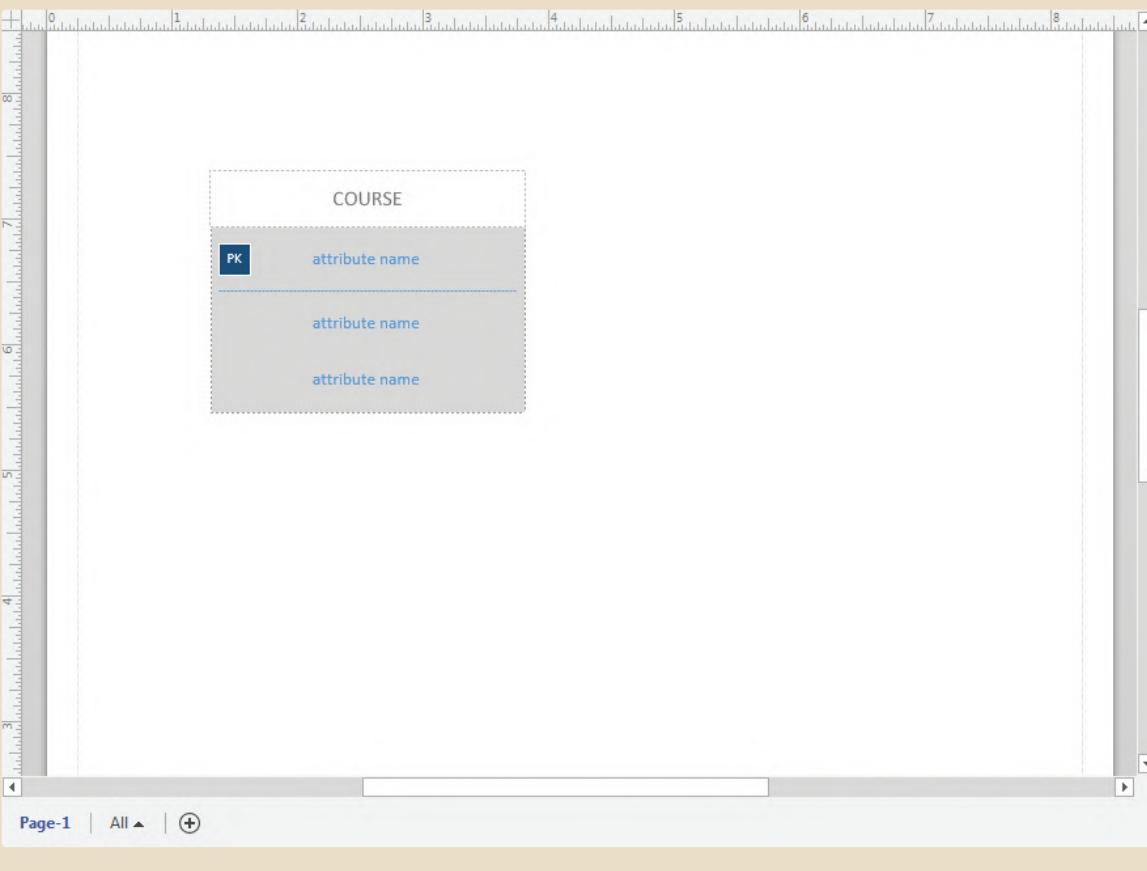
A2-6 Appendix A2

FIGURE A2.5 PLACING THE ENTITY OBJECT IN THE GRID



As you examine Figure A2.5, note that the small “squares” or handles around the entity object perimeter indicate that the object has been selected. You can deselect the object by clicking an empty portion of the grid. If the entity object has not been selected, click it to select it.

First, create a COURSE entity by double-clicking the text **Entity Name** to make the text editable and replacing it with COURSE, as shown in Figure A2.6.

FIGURE A2.6 TYPING THE COURSE ENTITY NAME

When you have finished typing the **COURSE** label as shown in Figure A2.6, you are ready to start defining the table columns. Visio places three undefined attributes in the entity by default to get you started.

A2-2b Defining the Entity Attributes (Columns)

Each table column represents one of the characteristics (attributes or fields) of the entity. For example, if the COURSE entity, represented by the COURSE table, is described by the course code, the course description, and the course credits, you can expect to define three columns in the COURSE table. Table A2.1 provides a preview of the expected COURSE table structure. (A few sample records are entered to give you an idea of the COURSE table contents.)

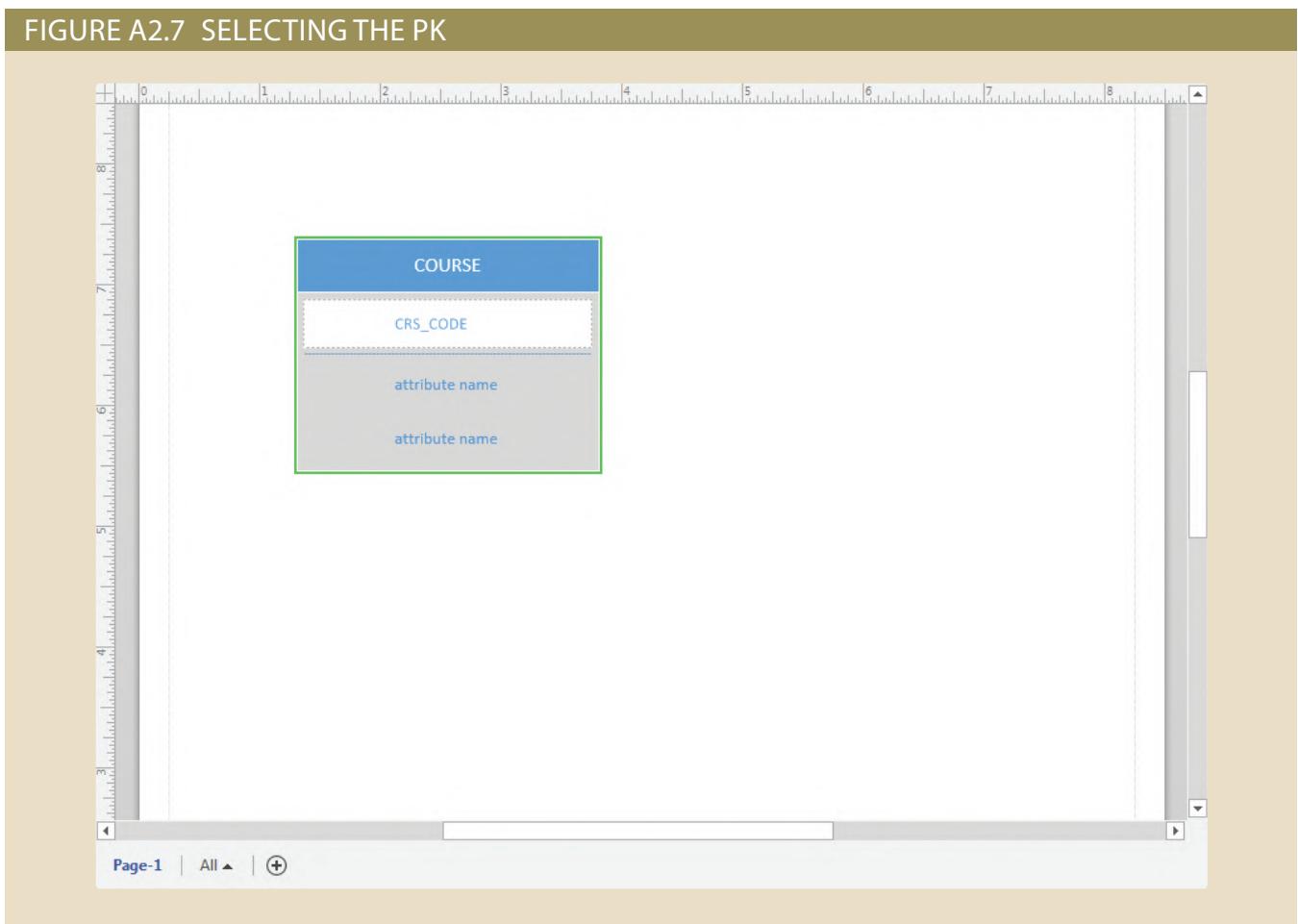
A2-8 Appendix A2

TABLE A2.1

SOME SAMPLE COURSE RECORDS

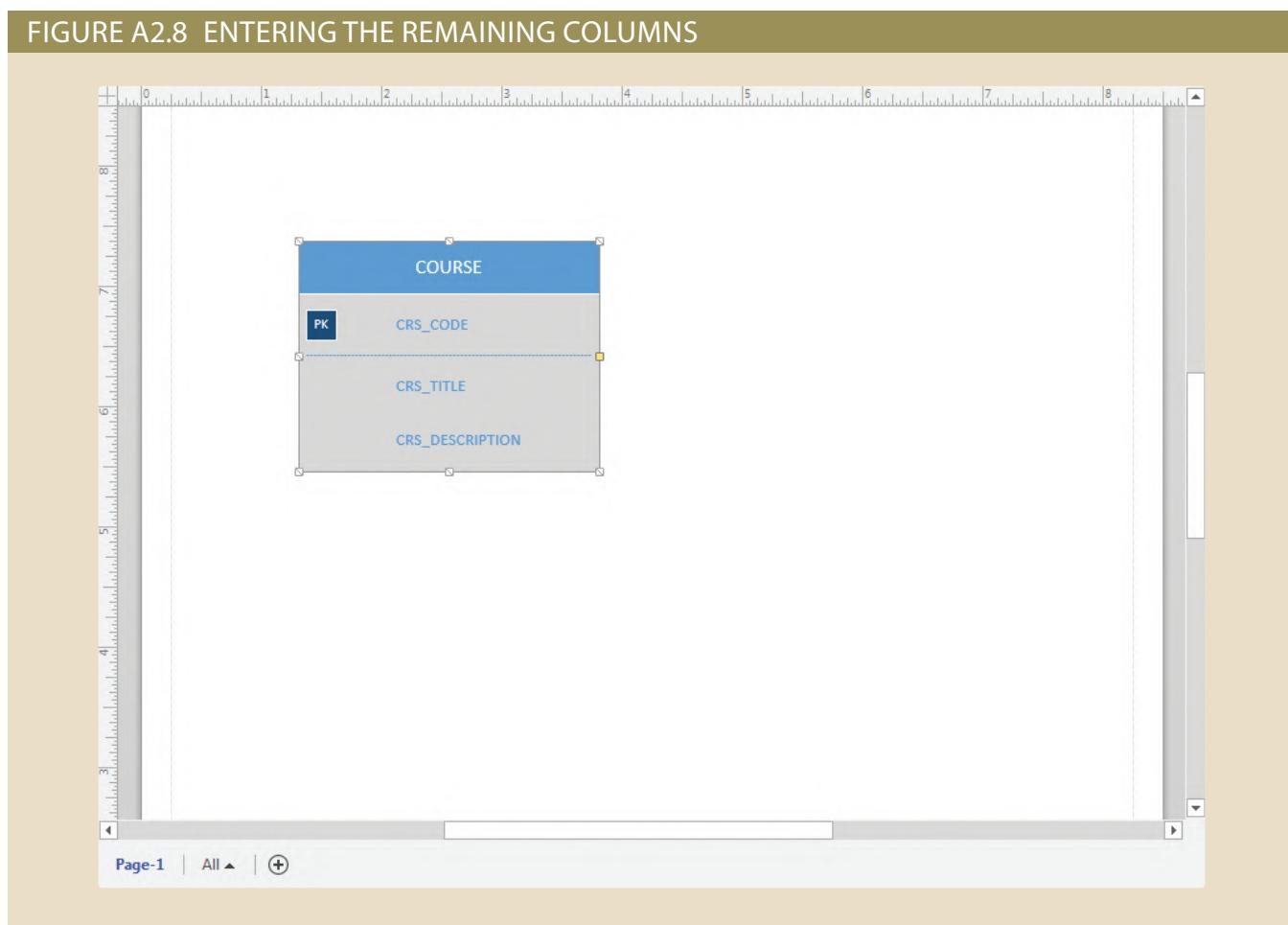
CRS_CODE	CRS_TITLE	CRS_DESCRIPTION	CRS_CREDITS
ACCT-345	Managerial Accounting	Accounting as a management tool. Prerequisites: Junior standing and ACCT-234 and 245.	3
CIS-456	Database Systems Design	Creation of conceptual models, logical models, and design implementation. Includes basic database applications development and the role of the database administrator. Prerequisites: Senior standing and at least 12 credit hours in computer information systems, including CIS-234 and CIS-345.	4
ECON-101	Introduction to Economics	An introduction to economic history and basic economic principles. Not available for credit to economics and finance majors.	3

To define the columns of the COURSE table, you must assign column names and characteristics. The first column in the COURSE table will be the CRS_CODE, which serves as the table's primary key (PK). Visio assumes that every entity will have a primary key, as it should, and begins the entity with the first attribute already labeled PK. If a composite primary key is needed, another primary key attribute can be added to the entity by dragging the **Primary Key Attribute** symbol from the left side of the screen and dropping it in the entity. In the current business rules, a single attribute primary key is appropriate. Double-click the first attribute in the entity to make it editable, and type **CRS_CODE** as the attribute name to match Figure A2.7.

FIGURE A2.7 SELECTING THE PK

You are now ready to make the entries for the second and third COURSE attributes. Name these attributes **CRS_TITLE** and **CRS_DESCRIPTION**, respectively (Figure A2.8).

FIGURE A2.8 ENTERING THE REMAINING COLUMNS



While Visio starts the entity with three attributes, more attributes can be added as needed. One way to add attributes to the entity is to drag the **Attribute** symbol from the left side of the screen and drop it on the entity. Attributes in the entity can be dragged and dropped within the entity to change their order so it is easy to adjust the order of the attributes. An attribute can also be added to an entity by hovering the mouse to the right of the entity border for about two seconds. At this time, a red line will appear in the entity indicating that an attribute can be inserted at that location in the entity, as shown in Figure A2.9. Left-click when the red line is positioned at the bottom of the attribute list in the COURSE entity to insert a new attribute symbol. Change the name of this attribute to **CRS_CREDITS** to match Figure A2.10

FIGURE A2.9 INSERTING A NEW ATTRIBUTE

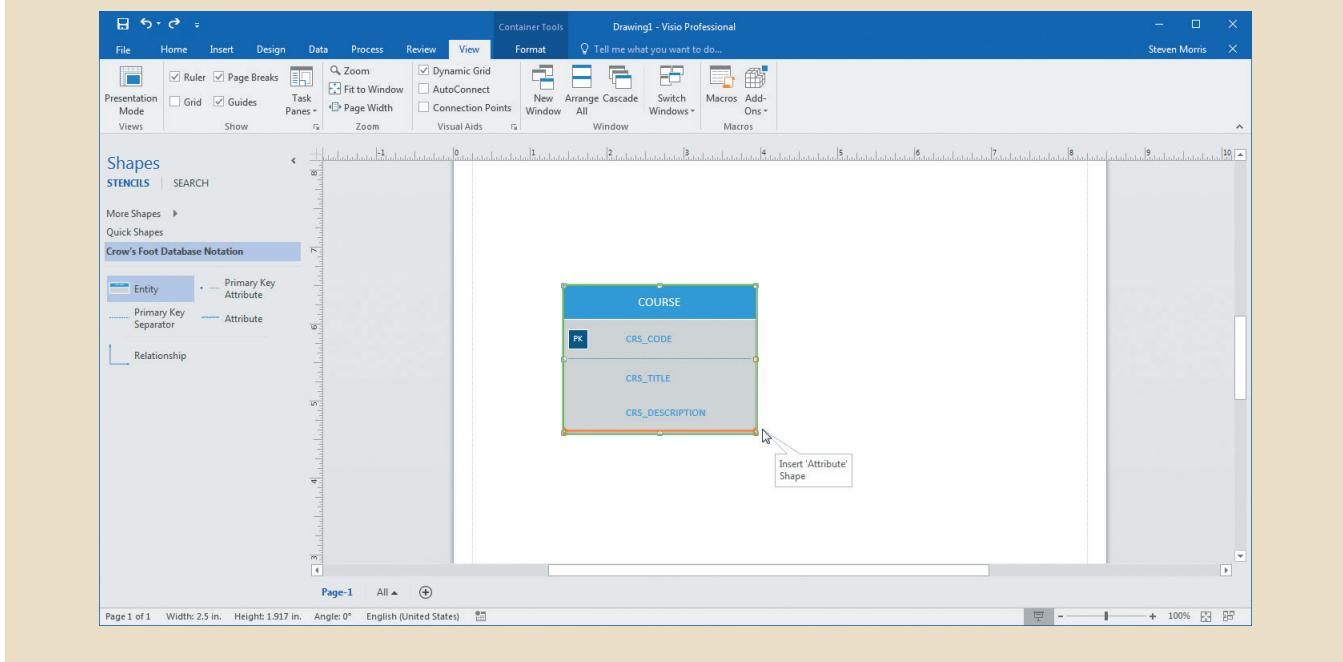
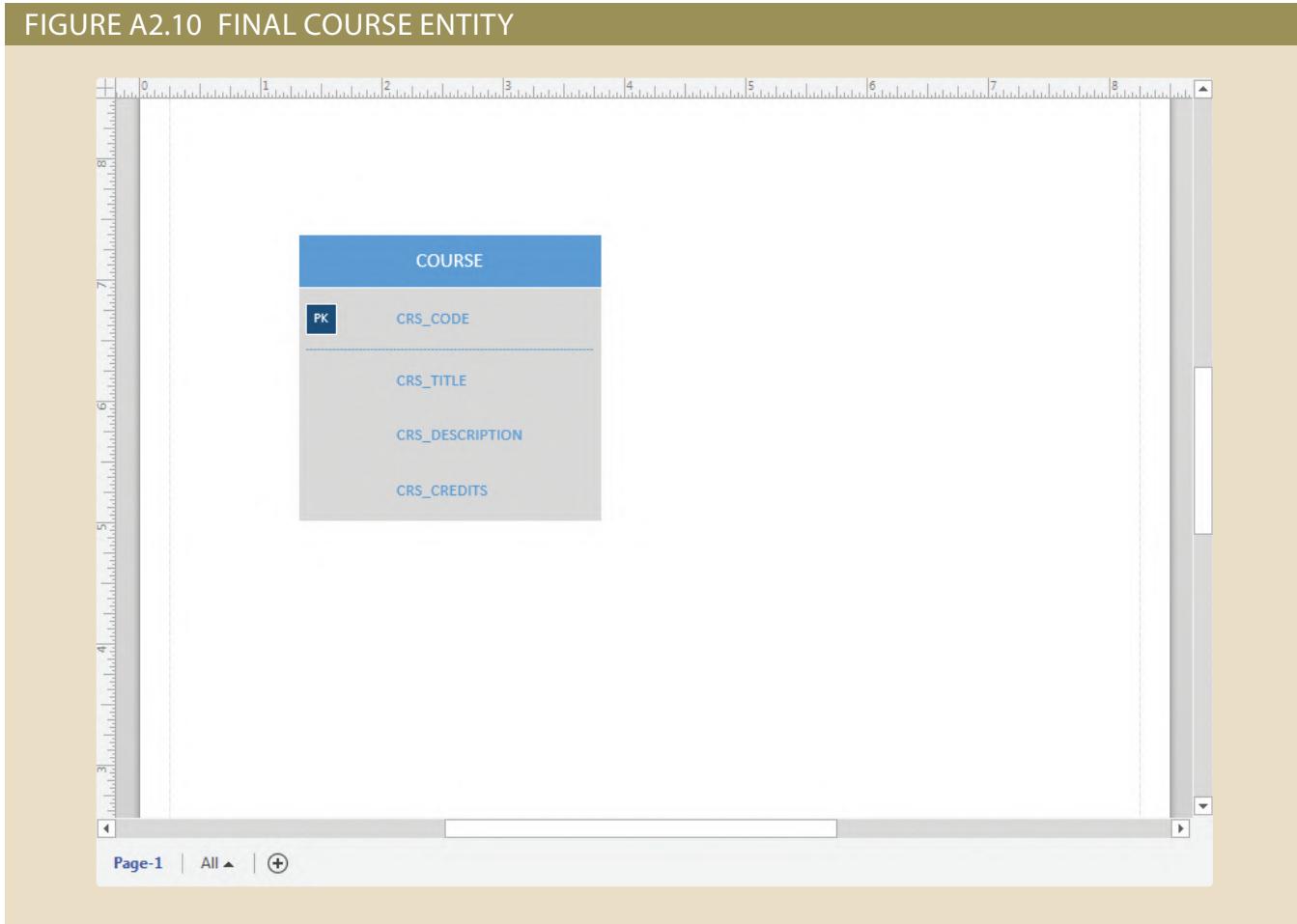


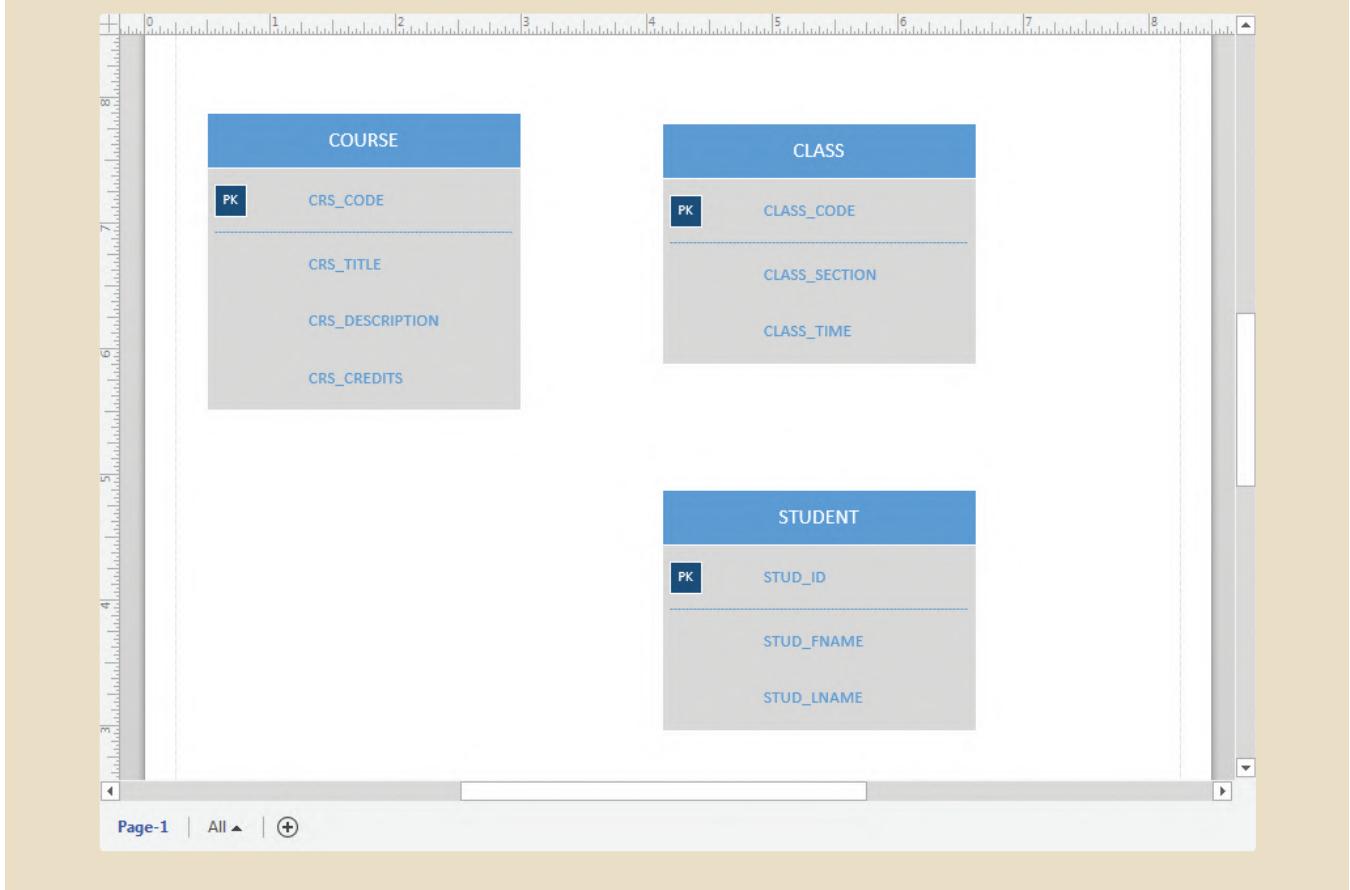
FIGURE A2.10 FINAL COURSE ENTITY



A2-2c Defining the CLASS and STUDENT Entities

You are now ready to define the CLASS and STUDENT entities, using the same techniques you used to create the COURSE entity. When you are done, the screen will look like Figure A2.11.

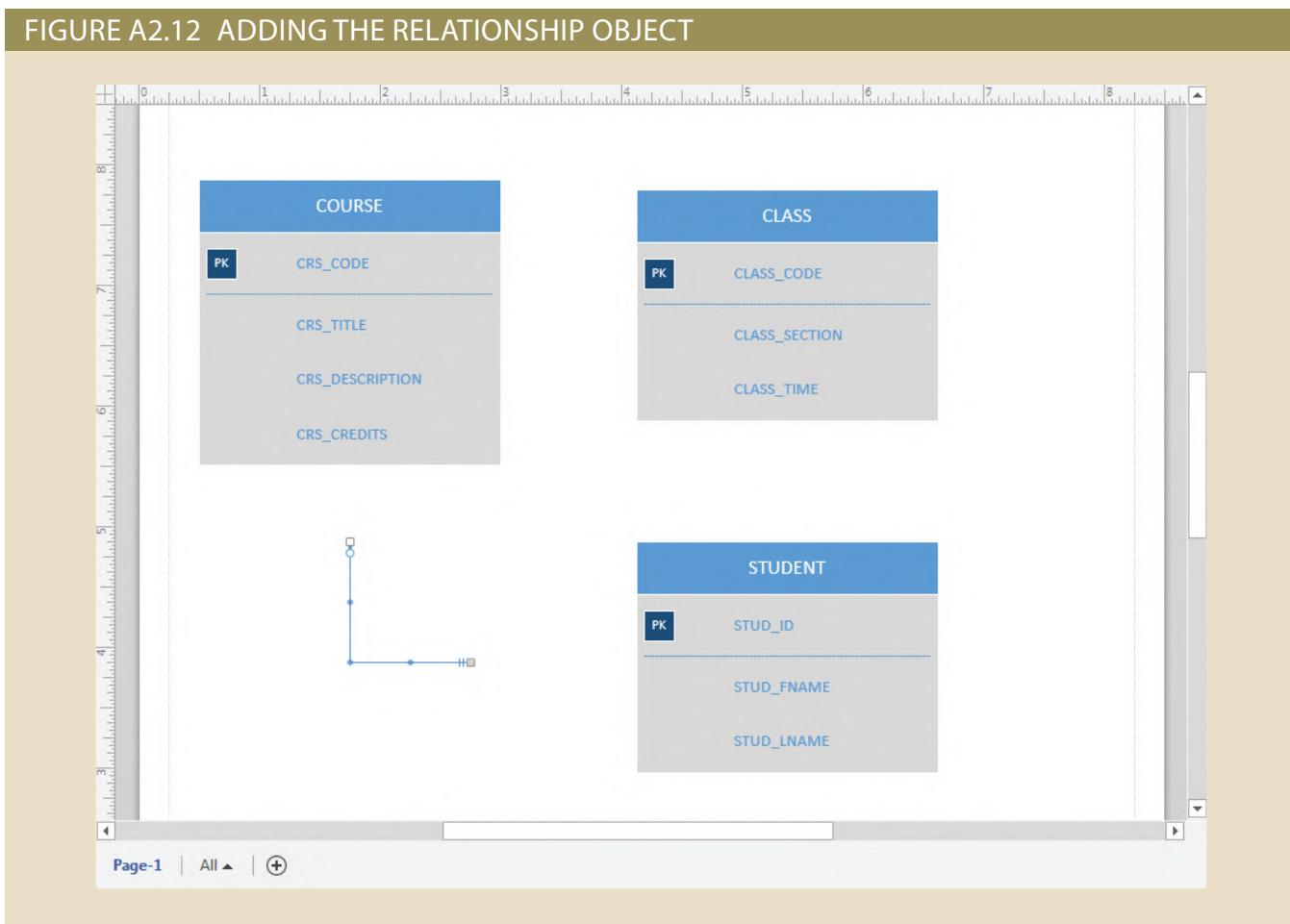
FIGURE A2.11 ADDING THE CLASS AND STUDENT ENTITIES



A2-3 Defining Relationships

To create a relationship between the entities, click the **Relationship** object, drag it to the grid, and drop it on the drawing board to produce the results shown in Figure A2.12.

FIGURE A2.12 ADDING THE RELATIONSHIP OBJECT



Dropping the **relationship** object on the grid produces the relationship line. Further note that the symbols at the two ends of the relationship line reflect default cardinalities of (1,1) and (0,N). By default, relationship lines and the Crow's Foot symbols are small and can be difficult to see. To enlarge the relationship line and the symbols, right-click the relationship object and select **Format Shape** as shown in Figure A2.13. The Format Shape window will open on the right side of the screen. From this window, the format of the relationship object can be adjusted. If the **Line** section is collapsed, click the pointer next to it to expand the line options. Set the **Width** to **1.5 pt**, and the **Begin Arrow size** and **End Arrow size** options to **Jumbo** to match Figure A2.14. When you have finished with the format options, close the **Format Shape** window by clicking the **Close** button at the top right corner of that window.

FIGURE A2.13 FORMATTING THE RELATIONSHIP LINE OBJECT

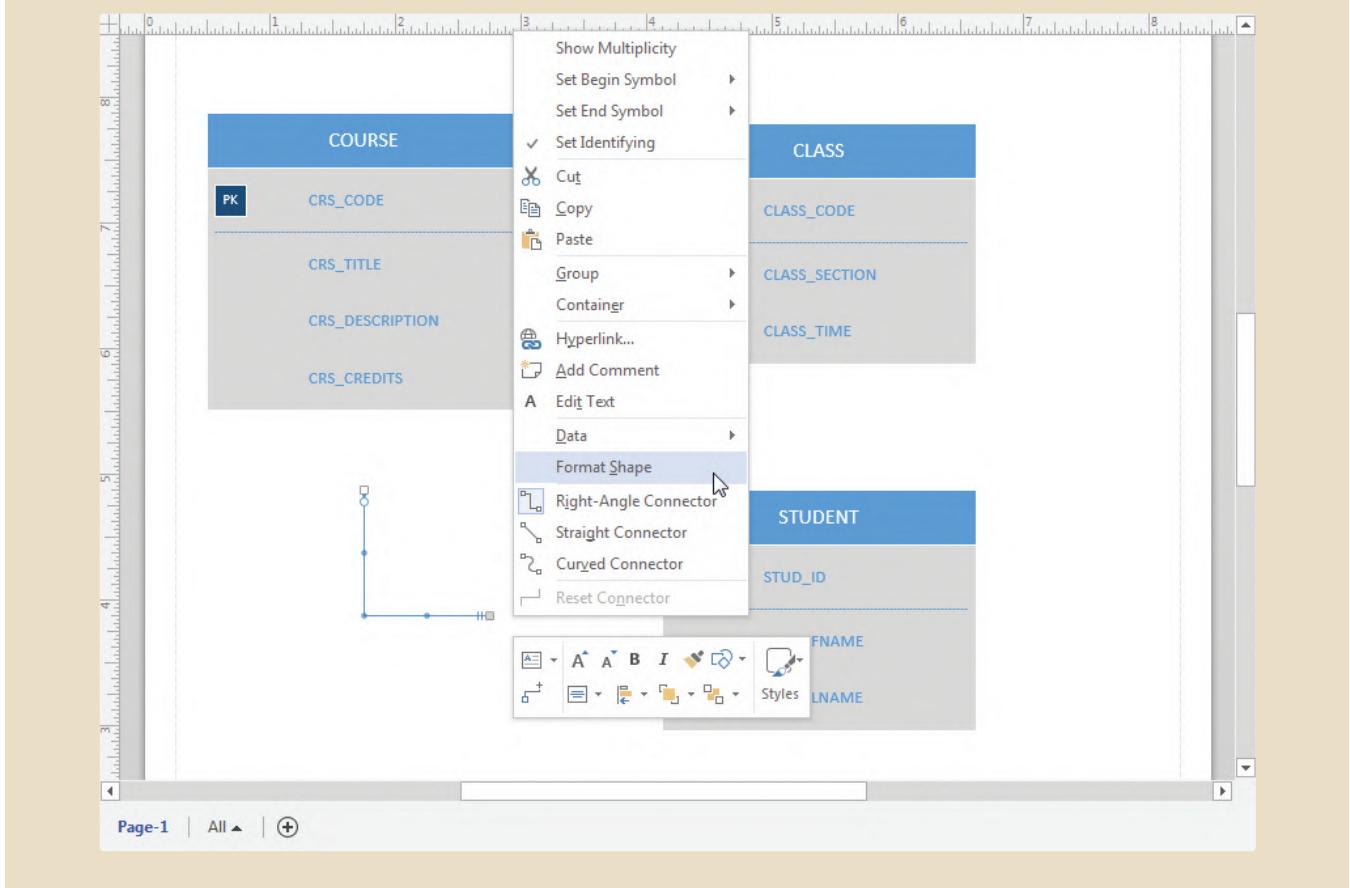
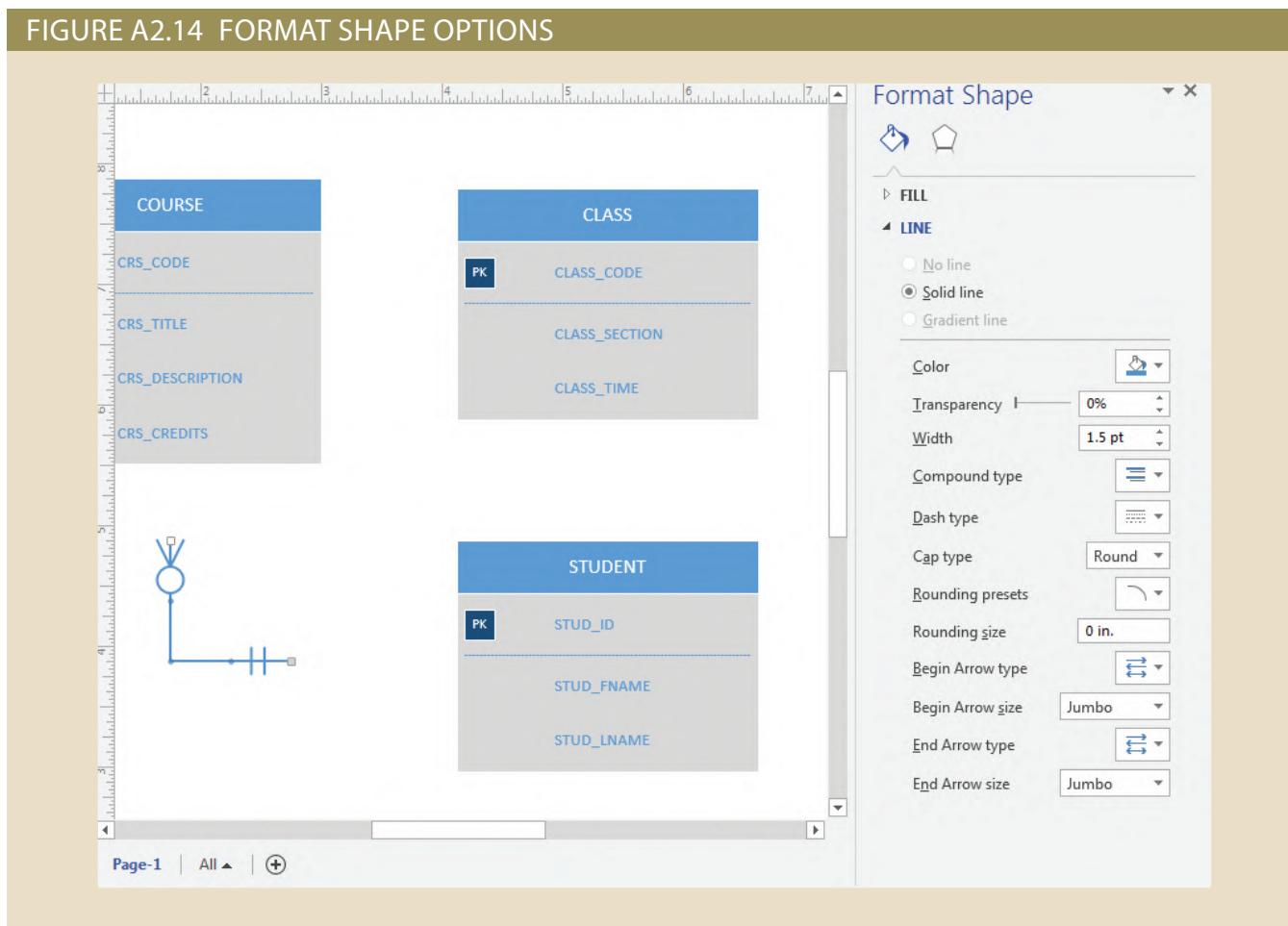
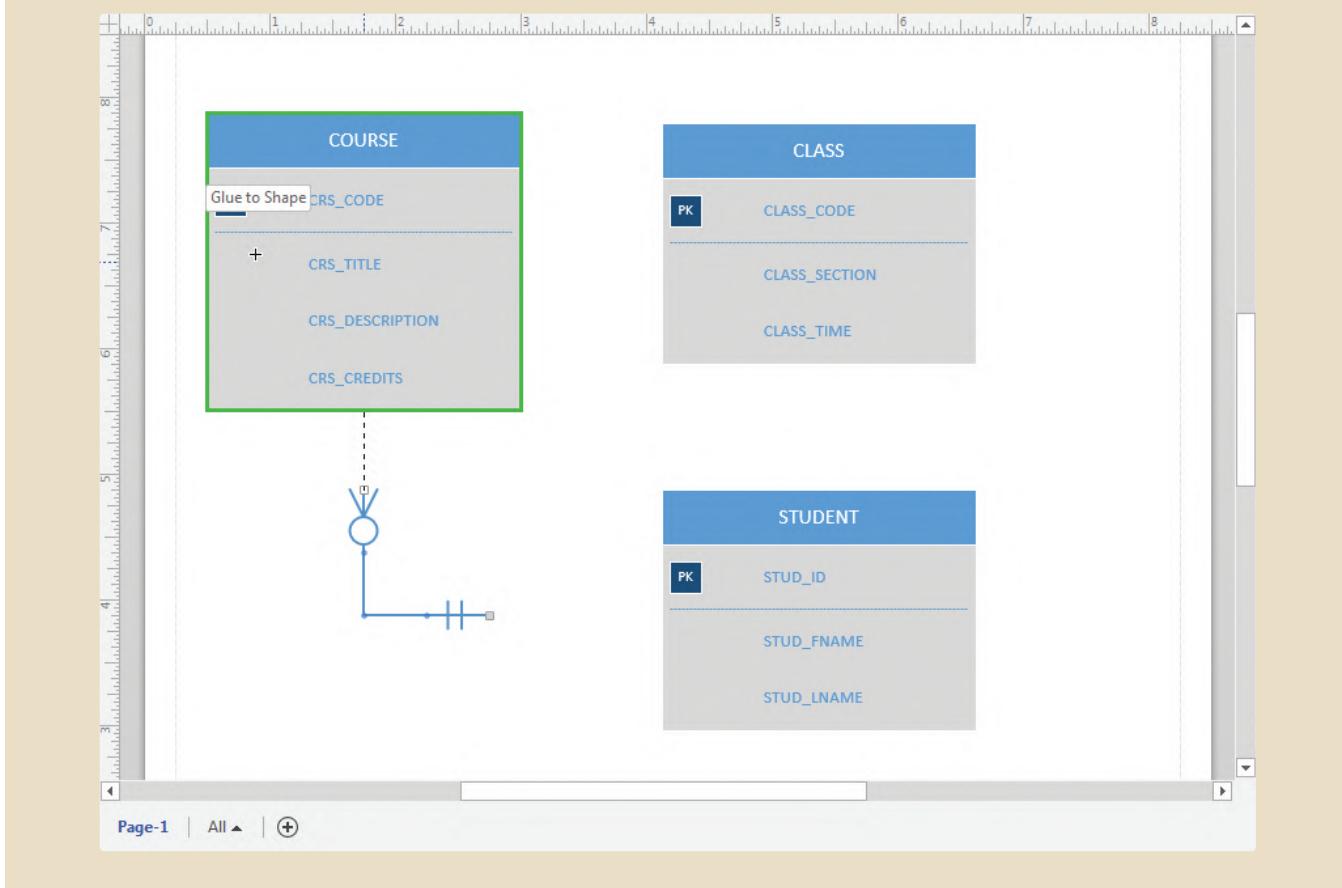


FIGURE A2.14 FORMAT SHAPE OPTIONS



Remember that the relationship to be established between COURSE and CLASS reflects the business rule “One COURSE may generate many CLASSES.” Therefore, the COURSE represents the “1” side of the relationship and the CLASS represents the “many” side of the relationship.

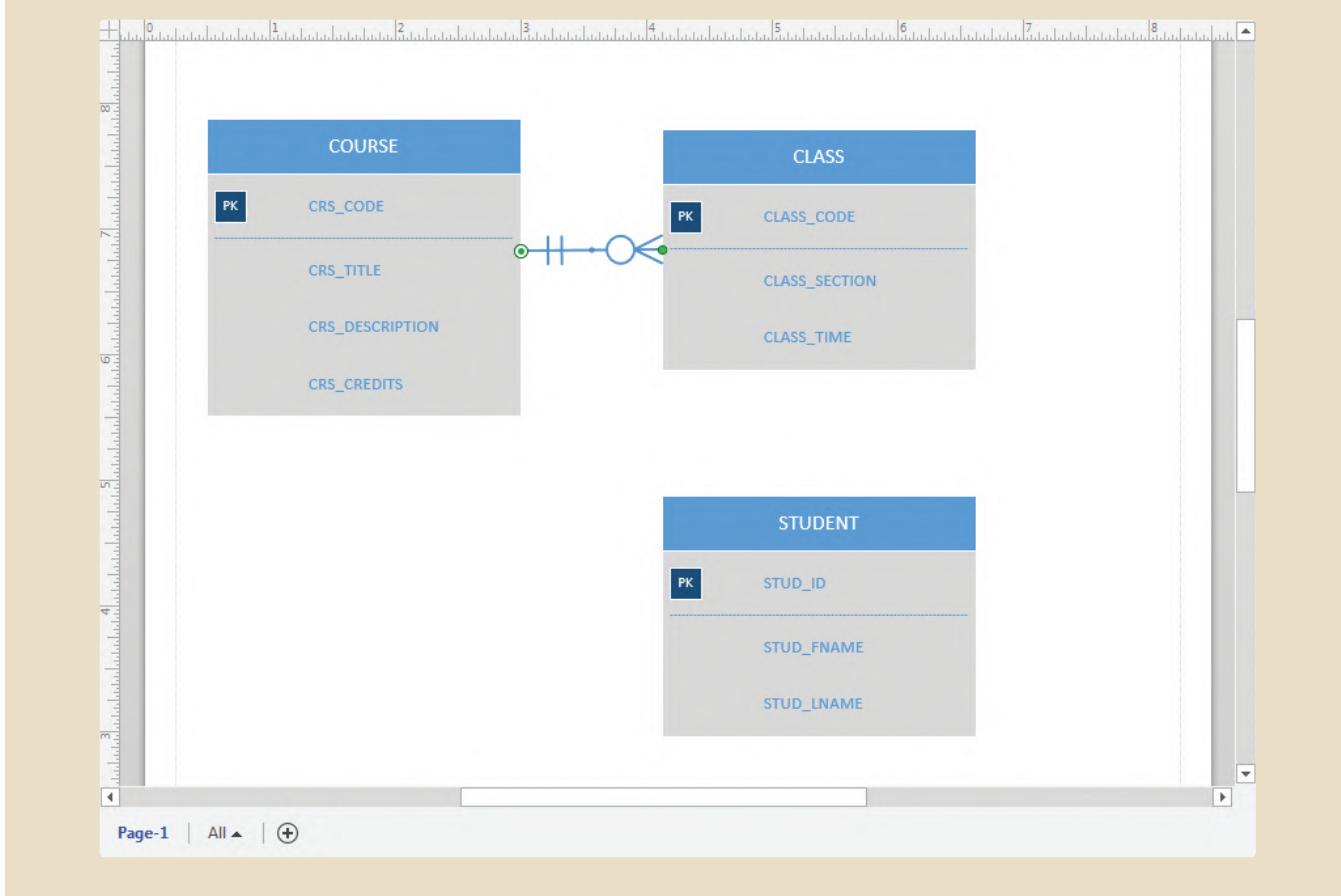
Attach the “1” side of the relationship line to the COURSE entity by dragging the “1” end of the relationship line to the COURSE entity, as shown in Figure A2.15. Visio will allow you to attach the relationship to several points in the entity. The relationship can be attached to a particular location on the border of the entity, a particular attribute box, the entity name box, or to the entity as a whole. Wherever you attach the relationship object, Visio will always anchor the relationship line to that location. We recommend attaching to the entity as a whole so that Visio can more effectively move the relationship object as needed to cleanly route relationships on the diagram. The relationship is attached to the entity as a whole when the entire entity perimeter is highlighted in green, as shown in Figure A2.15.

FIGURE A2.15 ATTACHING THE "I" SIDE OF THE RELATIONSHIP LINE

Using the same technique that was used to attach the “1” side of the relationship, drag the “M” side of the relationship line to the CLASS entity to produce Figure A2.16. You can use the handles in the relationship line to adjust the layout, if needed.

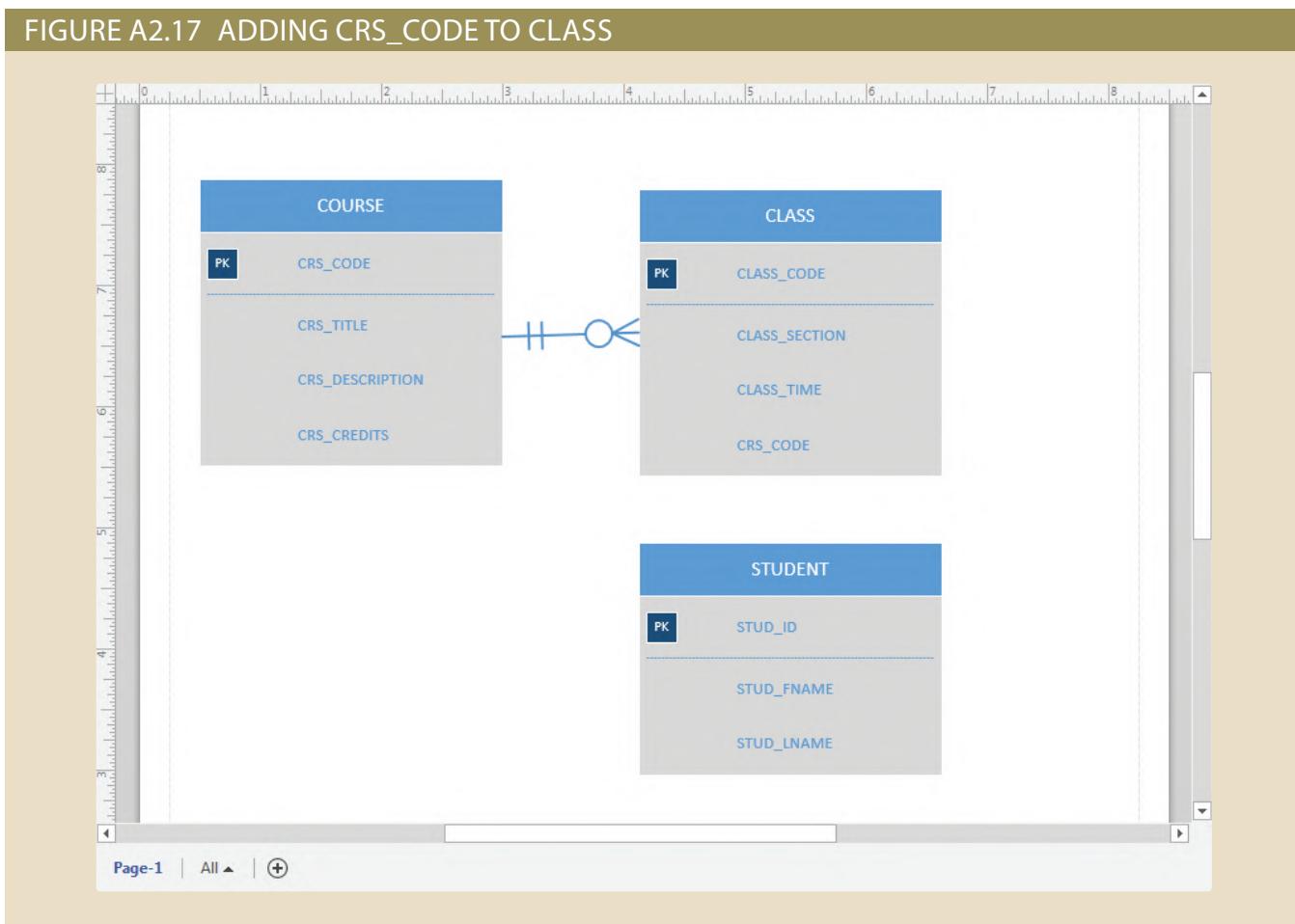
A2-18 Appendix A2

FIGURE A2.16 ATTACHING THE "M" SIDE OF THE RELATIONSHIP LINE



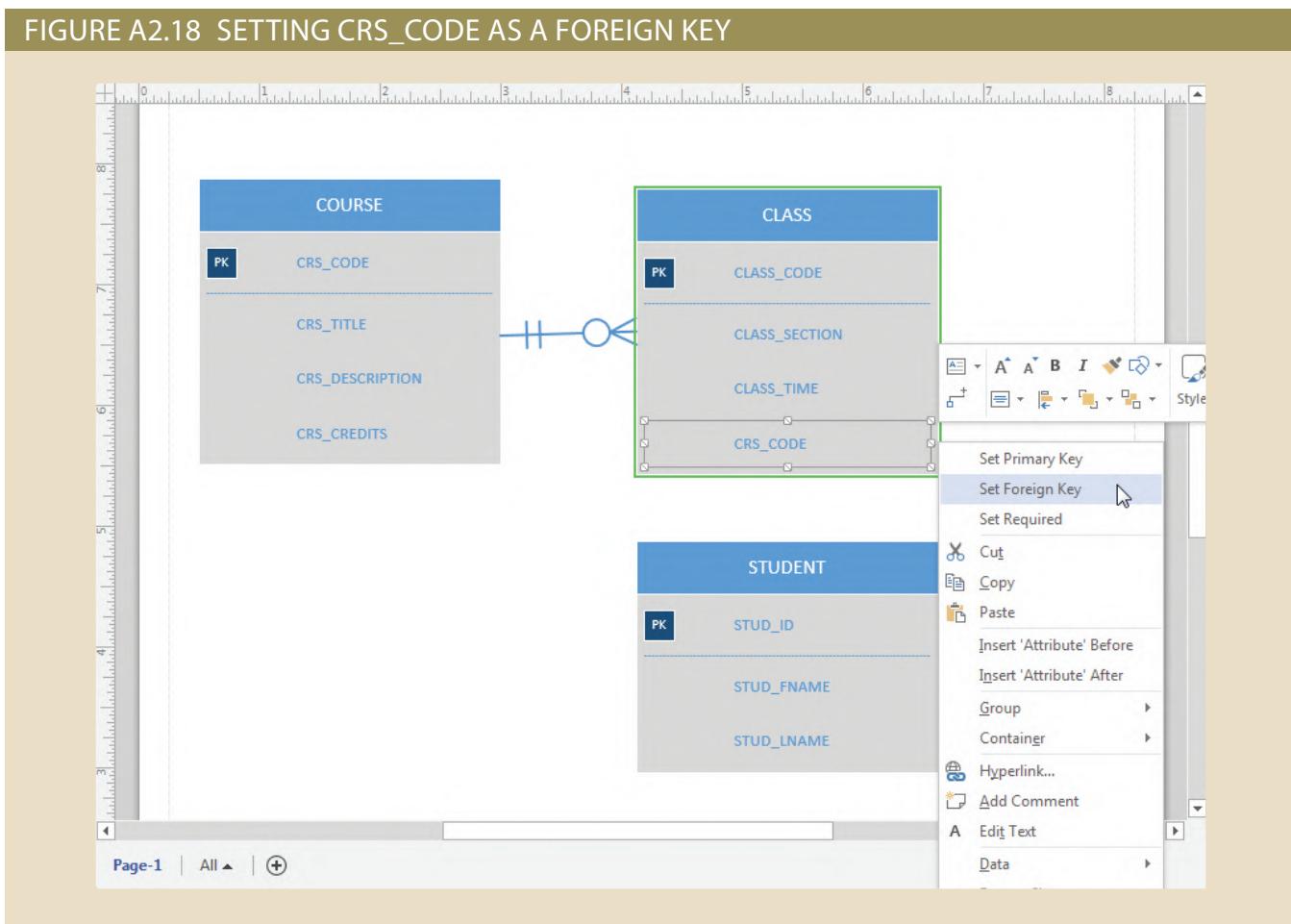
As you examine Figure A2.16, note that the model does not include a foreign key attribute in the CLASS table to implement the relationship that we just created. This foreign key attribute will have to be added manually and set as a foreign key. To do this, add an attribute to the CLASS entity using the same technique you used to add the CRS_CREDITS attribute to the COURSE entity earlier, as shown in Figure A2.17. Since the foreign key is always the primary key of the related table, the next step is to specify CRS_CODE as a foreign key in the CLASS table.

FIGURE A2.17 ADDING CRS_CODE TO CLASS



Visio allows database designers to indicate that an attribute is a foreign key by placing a small “FK” box next to the attribute, similar to the “PK” box used to identify primary keys. To mark CRS_CODE in the CLASS entity as a foreign key, click to select **CRS_CODE**, then right-click the CRS_CODE attribute and choose **Set Foreign Key** from the context menu, as shown in Figure A2.18.

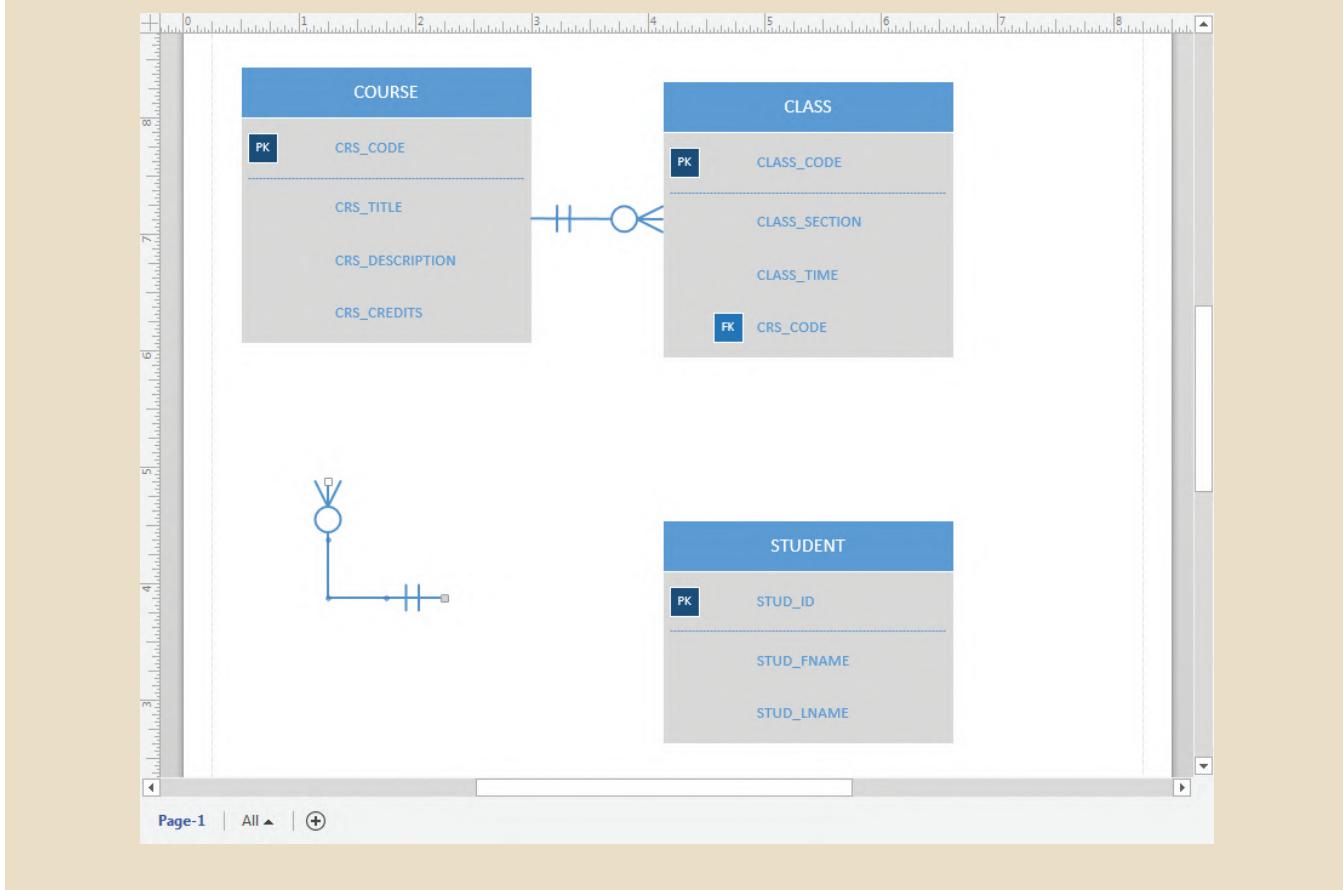
FIGURE A2.18 SETTING CRS_CODE AS A FOREIGN KEY



A2-3a Editing the Cardinalities

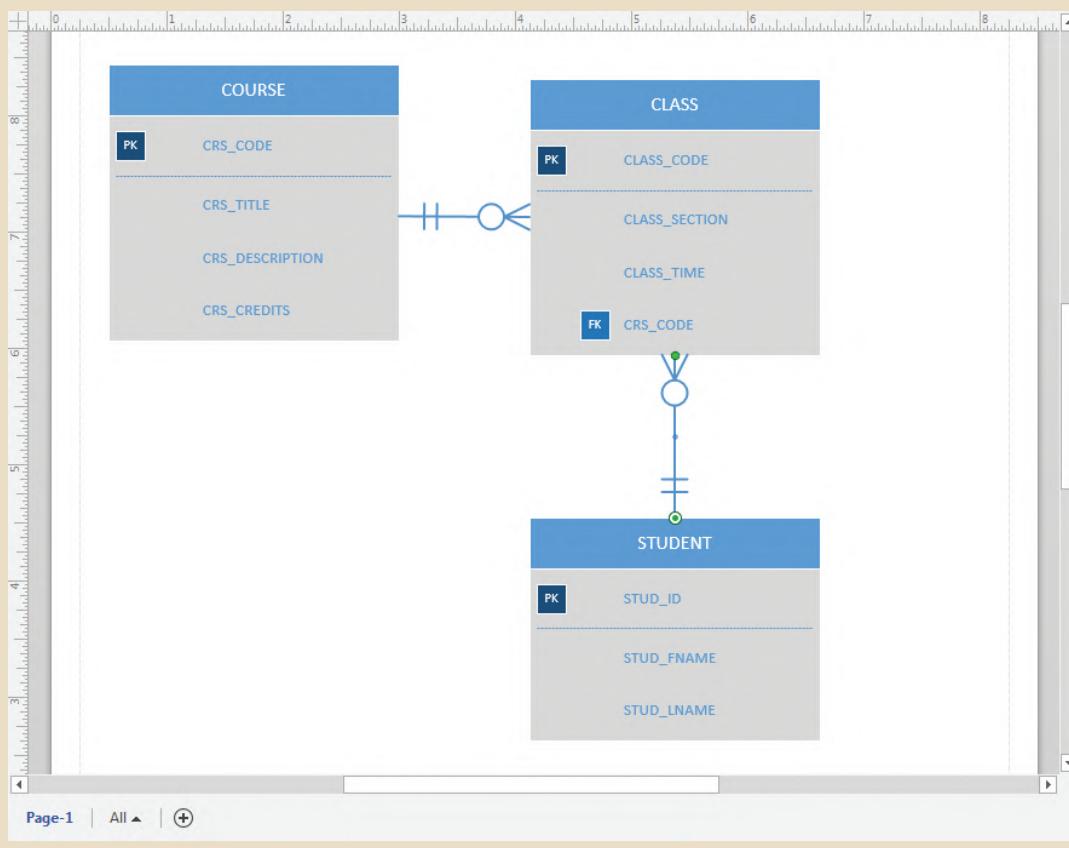
Using the same technique as above, add another relationship object to the drawing board, and set the format options the same as for the previous relationship. If you examine Figure A2.19, you'll notice that the default cardinalities of (1,1) and (0,N) are again presented. However, our business rules state that a student can enroll in many classes, and a class can have many students enrolled in it. Clearly, this will call for a M:N relationship, not a 1:M relationship; therefore, we will have to adjust the cardinalities of the relationship object.

FIGURE A2.19 ADDING A SECOND RELATIONSHIP OBJECT



Attach the relationship object to the CLASS and STUDENT entities, attaching the one-side of the relationship to STUDENT and attaching the many-side of the relationship to CLASS. Again, we prefer to attach the relationship object to the entity as a whole. The attached relationship object is shown in Figure A2.20.

FIGURE A2.20 ATTACHING THE SECOND RELATIONSHIP



To adjust the cardinalities, right-click the relationship object to open the context menu. From the context menu, the Crow's Foot symbols can be changed to match the cardinality of the business rules. Based on the business rules, we know that the relationship should be optional in both directions with a connectivity of "many" in both directions. The **Set Begin Symbol** option is already set to **Zero or more**, which corresponds to our desired cardinality. On the context menu, move the cursor to **Set End Symbol** and choose **Zero or more** to change the current (1,1) cardinality to (0,N) to match our business rules as shown in Figure A2.21. Your result should match Figure A2.22.

FIGURE A2.21 SETTING THE CARDINALITIES

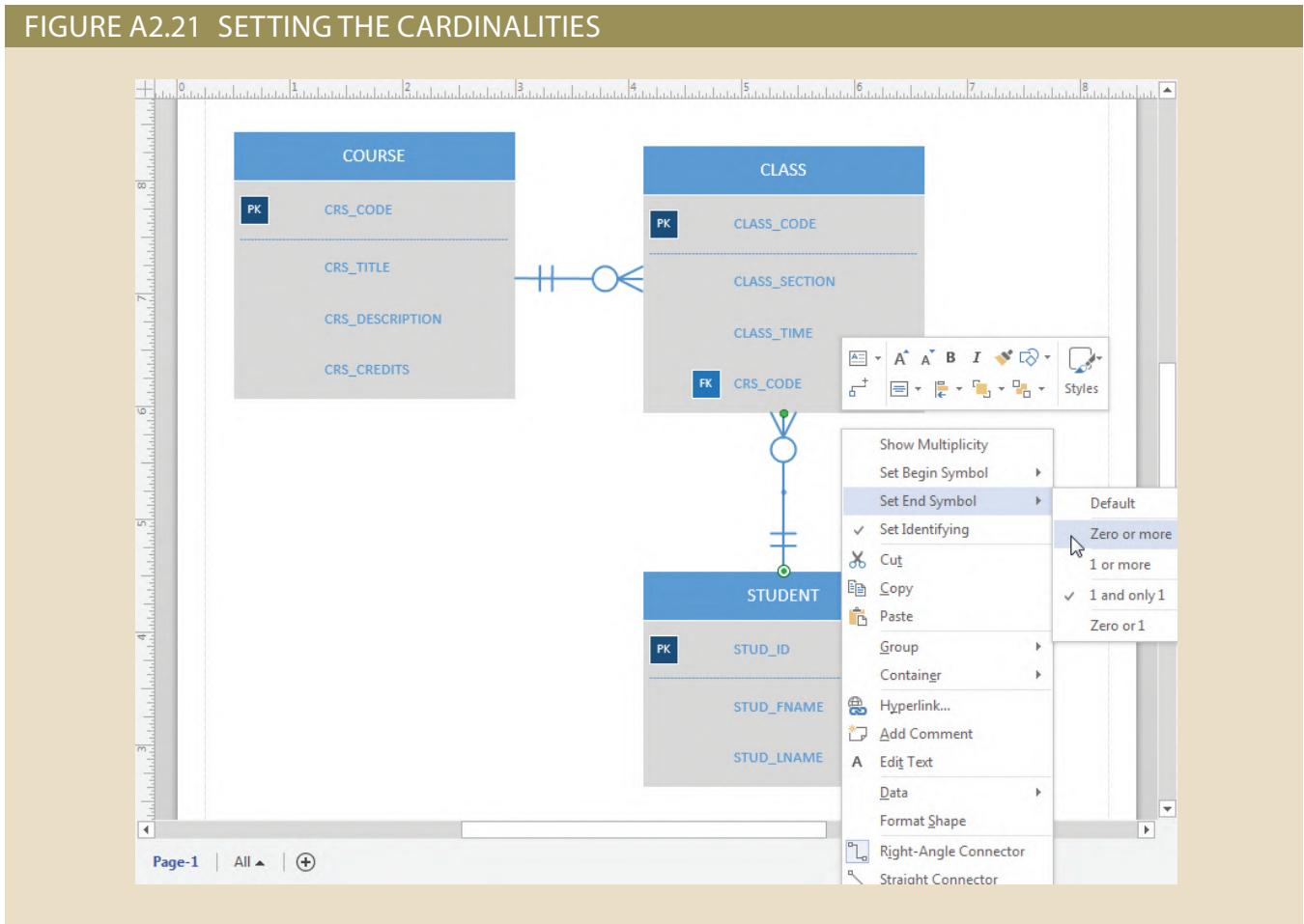
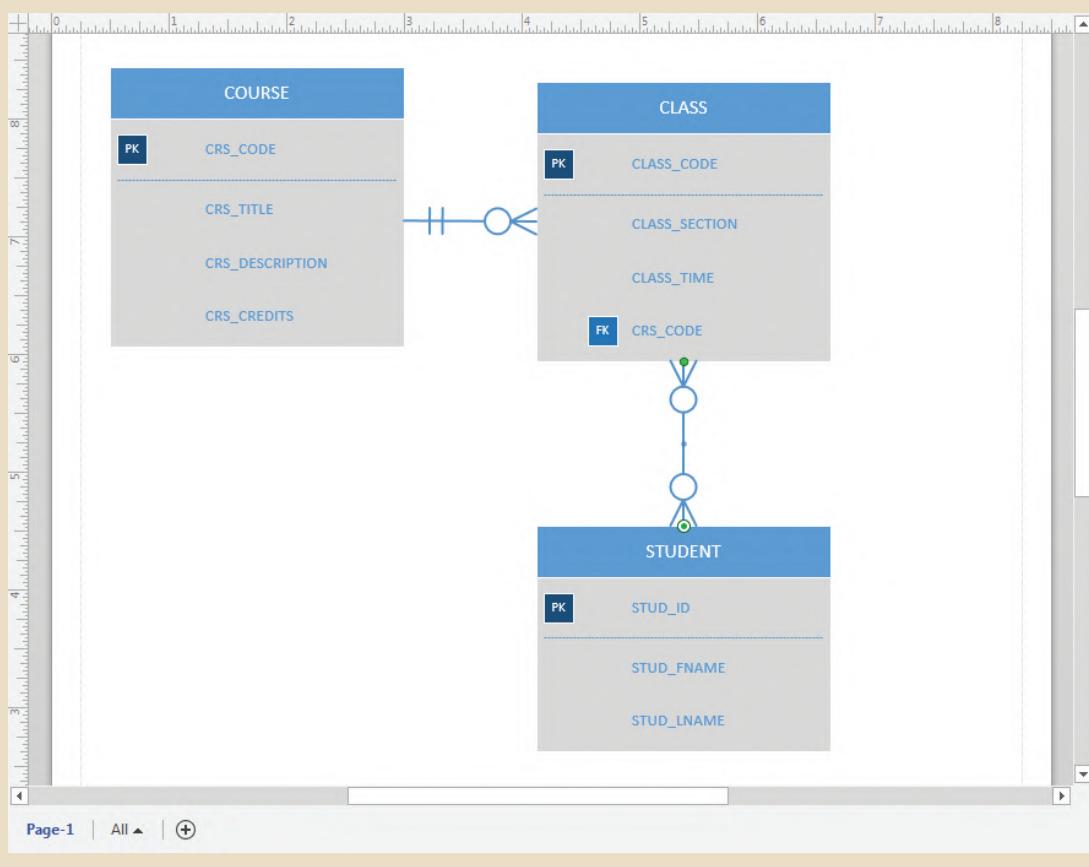
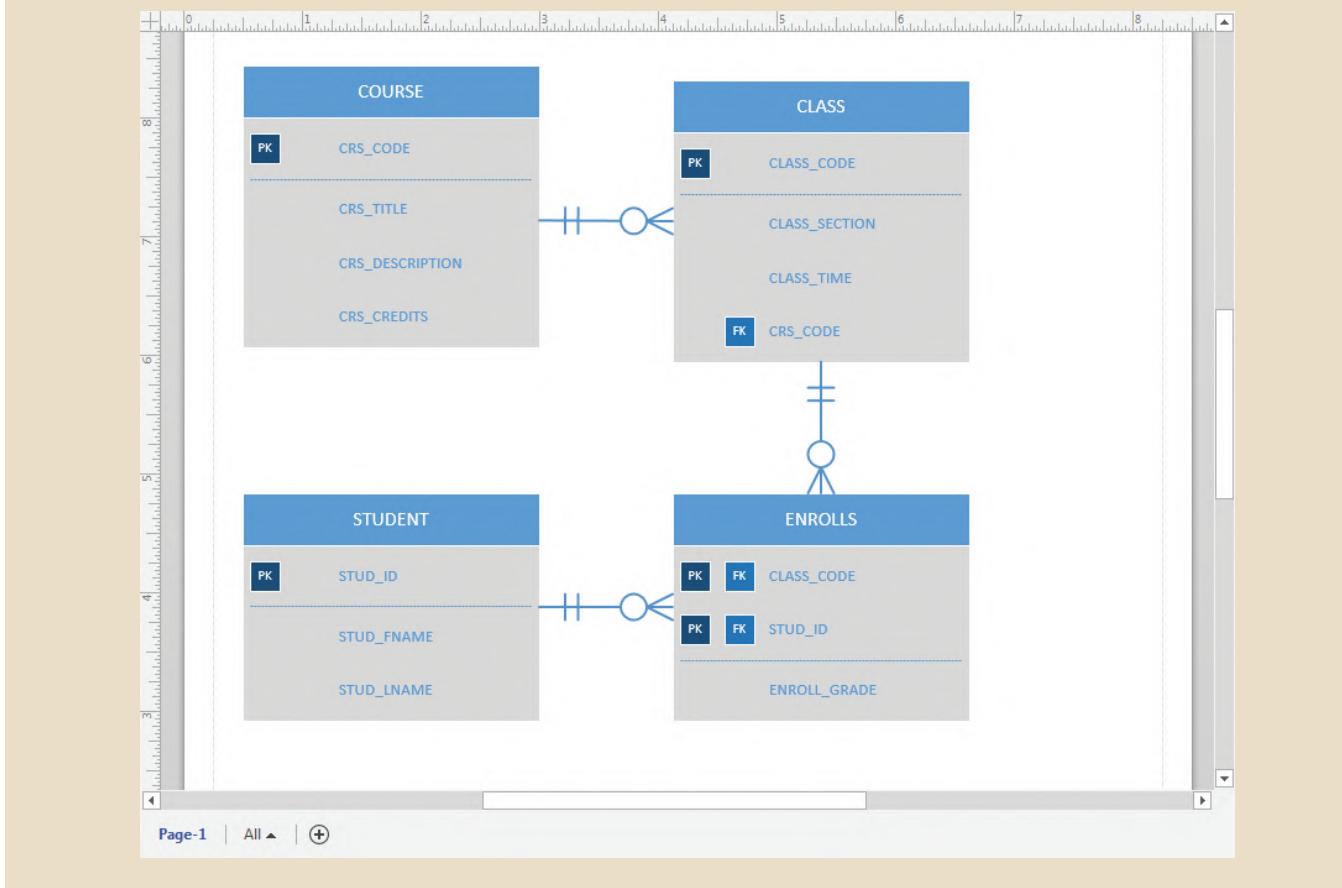


FIGURE A2.22 COMPLETED SECOND RELATIONSHIP



Notice in Figure A2.22 that there is not a foreign key to implement the many-to-many relationship between CLASS and STUDENT. As discussed in Chapter 4, Entity Relationship Modeling, many-to-many relationships are appropriate in conceptual data models. Recall that M:M relationships cannot be directly implemented in a relational database without the use of an associative or composite entity, and part of the process of converting a conceptual model to a logical model is the decomposition of M:M relationships into 1:M relationships. For practice, create the ENROLLS composite entity shown in Figure A2.23. Use CLASS_CODE and STUD_ID as a composite primary key in the ENROLLS entity, and be certain to mark them both as foreign keys as well.

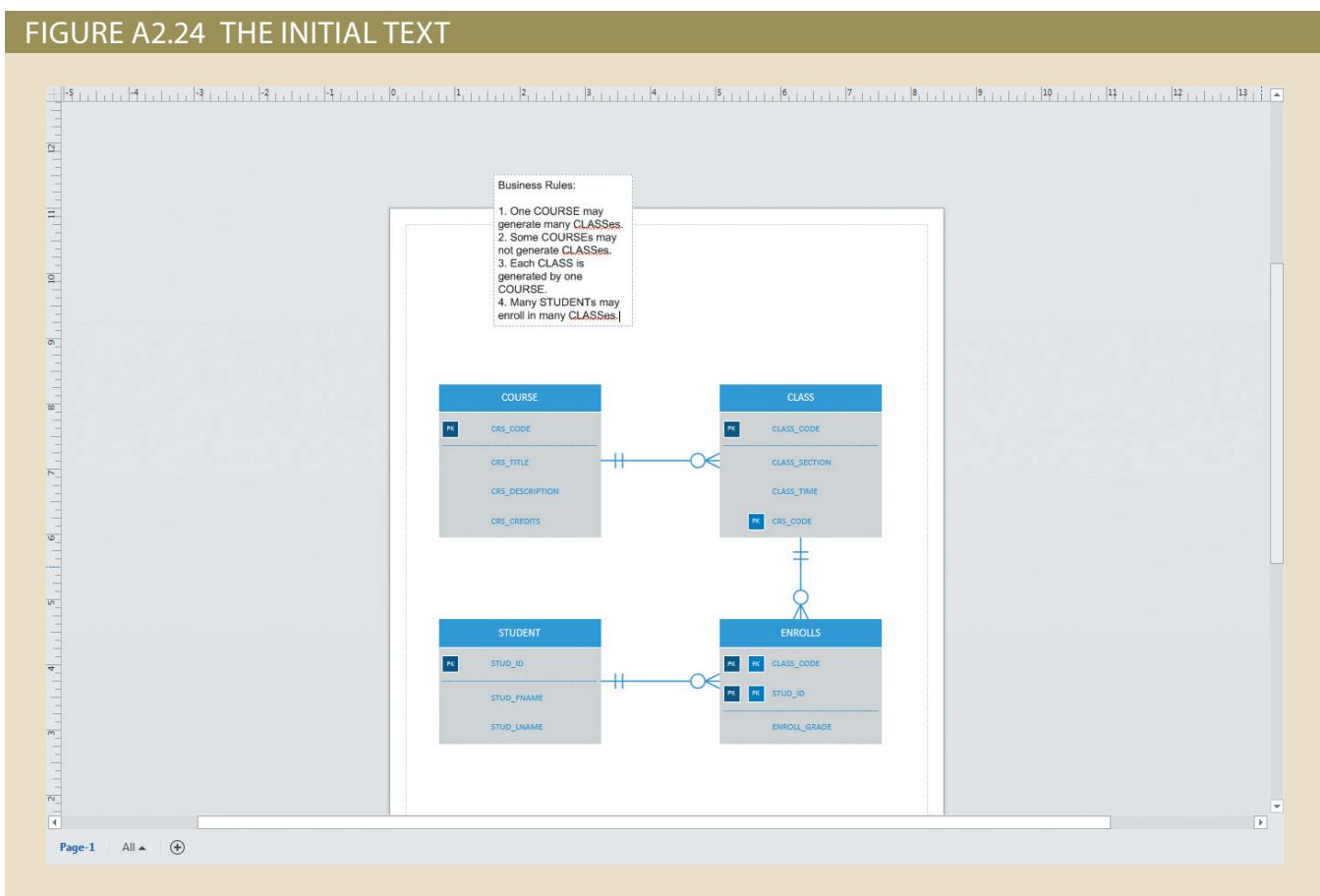
FIGURE A2.23 FINAL DATA MODEL WITH COMPOSITE ENTITY



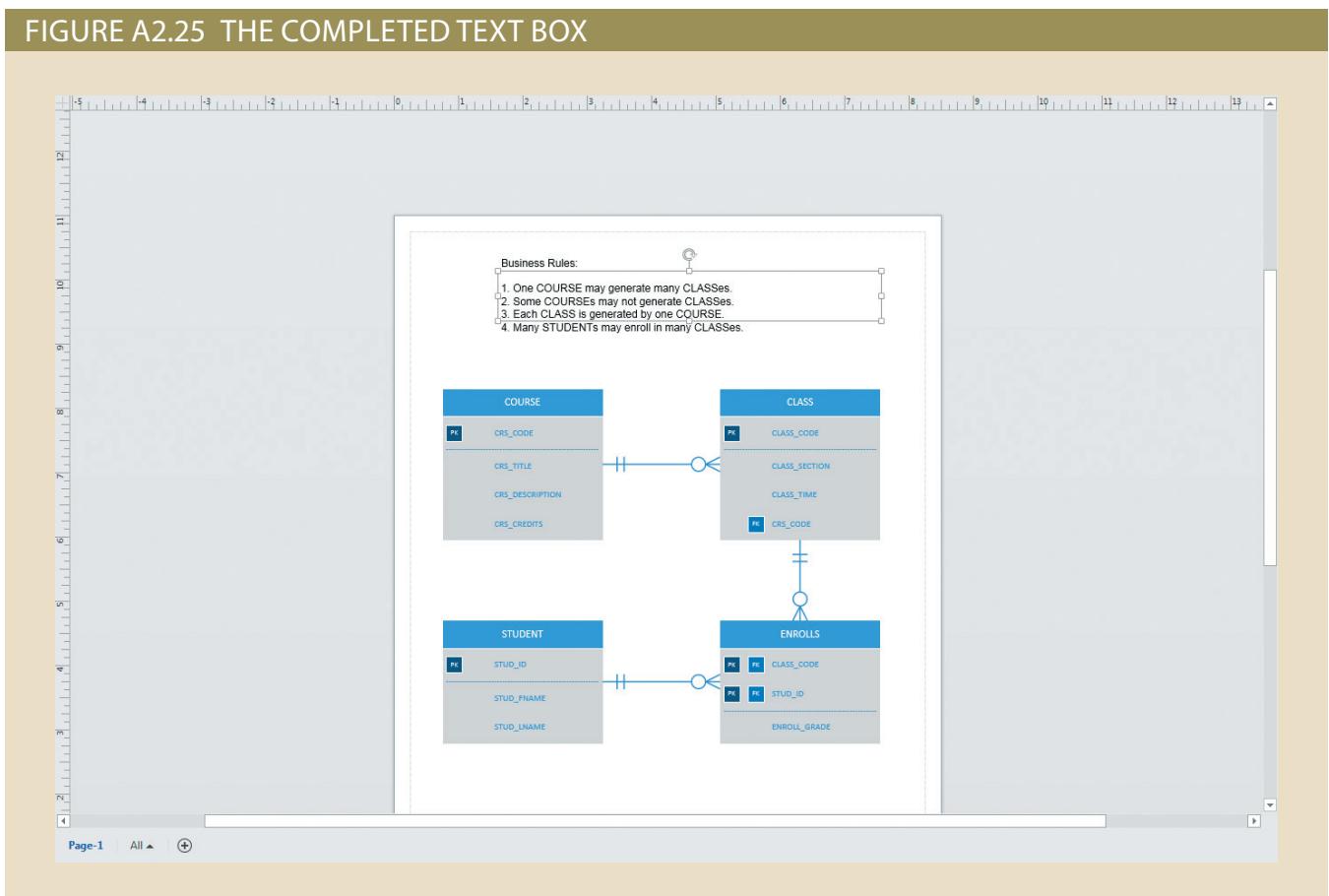
A2-4 Placing Text on the Grid

To help document the ERD, it may be helpful to place explanatory notes on the grid. Make sure that you have not selected any object by clicking a blank area of the screen. Change the **Zoom** to 75% on the **VIEW** tab on the Ribbon; then select the **Text tool** (marked **A Text**) shown at the top of the screen in the Tools group on the **HOME** tab. You will see the effect of your selection when you note the cursor's new look. Select the text format to suit your needs—left alignment, black font color, and a font face and size of Arial 12pt have been selected in Figure A2.24. After making these selections, enter the text as shown. (You can modify any text format such as the font, size, color, and justification later.)

FIGURE A2.24 THE INITIAL TEXT



To move the text box, you must first make sure that the Text tool has been deselected, and the **Pointer Tool** is selected. Clicking the text box produces a set of small squares (handles) shown on the text box perimeter. After the text box has been selected, you can drag and drop it as you would any other object on the screen. In fact, the text box behaves like any other Windows object. For example, you can change the size of the text box by dragging its perimeter in or out. Resize the text box so that the text format matches Figure A2.25.

FIGURE A2.25 THE COMPLETED TEXT BOX

Don't forget to save your Visio file before you exit. As with all Windows applications, you will be reminded to save the file if you try to close it without first saving it.

Appendix B

The University Lab: Conceptual Design

Preview

The pieces of the database design puzzle come together in this appendix and Appendix C, The University Lab: Conceptual Design Verification, Logical Design, and Implementation. You will develop a conceptual database design by using the ideas and techniques presented in Chapter 4, Entity Relationship (ER) Modeling; Chapter 5, Advanced Data Modeling; Chapter 6, Normalization of Database Tables; and Chapter 9, Database Design.

You will see the evolution of a database system, starting with the results of the database initial study and moving through a conceptual design's initial ER diagram. In Appendix D, Converting an ER Model into a Database Structure, you will see how a conceptual design is evaluated and transformed into a logical design that can be implemented in any relational DBMS environment.

Data Files and Available Formats

[MS Access](#) [Oracle](#) [MS SQL](#) [My SQL](#)

[MS Access](#) [Oracle](#) [MS SQL](#) [My SQL](#)

There are no data files for this appendix.

Data Files Available on cengagebrain.com

B-2 Appendix B

Many years of teaching database design have taught the authors a valuable lesson: If you have never stepped through a complete example of database design, chances are that you will not be able to successfully design and implement a database system.

“I hear it and I forget it, I see it and I remember it, I do it and I learn it.”

—Old Chinese proverb

The example will be the automation of a large university computer lab. Because the design detailed in this appendix is based on a real project, you will confront a few real-world problems and develop some important analytical skills.

A well-functioning system represents the culmination of several small steps. To follow the steps, but avoid getting lost in details, use Table B.1 as your map. Table B.1 shows that this appendix will take you through the first phase of a conceptual database design, through its initial ER model. The remaining database design tasks outlined in Table B.1 will be completed in Appendix C.

TABLE B.1

DATABASE DESIGN MAP FOR THE UNIVERSITY COMPUTER LAB (UCL)

DATABASE LIFE CYCLE PHASE	OUTPUT	SECTION
Database initial study	UCL objectives Organizational structure Description of operations ¹ Problems and constraints System objectives Scope and boundaries	B-1a B-1b B-1c B-1d B-1e B-1f
Database design		
Conceptual design	Information sources and users Information needs: user requirements The initial ER model Defining attributes and domains	B-2a B-2b B-2c B-1–B-2
Continued in Appendix C		
Logical design	Continued in Appendix C Normalization ER model verification	C-2 C-3
Physical design	Tables Indexes and views Access methods	C-4a C-4b C-5
Implementation		
	Creation of databases Database loading and conversion System procedures	C-6a C-6b C-6c
Testing and evaluation		
	Performance measures Security measures Backup and recovery procedures	C-7a C-7b C-7c
Operation		
	Database is operational Operational procedures	C-8a C-8b

¹The term *description of operations* is sometimes used as a synonym for the database initial study. However, the use of that synonym is appropriate only when the “operations” encompass the organization’s entire data environment, rather than just the transaction component of the data environment. This appendix will use the label “description of operations” in its more restrictive sense.

Many of the small steps in Table B.1 might appear to be trivial at first glance. Don't be tempted to overlook or rush through them. Those little details may make the difference between design success and failure. Later, it will be much easier to discard unnecessary details than to address omissions.

Database design is "detail" work. The details in this example should give you a better grasp of a design process that sometimes appears to be disorganized.

B-1 The Database Initial Study

The database initial study is basically a detailed description of an organization's current and proposed database system environments. Therefore, the database initial study must include a careful accounting of the organization's objectives, its structure, its operations, its problems and constraints, the system's objectives, the system's scope and boundaries, the information sources and users, and the end-user requirements.

A real-world database initial study is likely to have hundreds of pages because detail and accuracy are essential. The need for such detail and accuracy is obvious when you realize that the database design is based on the business rules derived from the database initial study. If the database initial study lacks detail and/or accuracy, the business rules are likely to be incomplete or inaccurate. It follows that the database design based on such business rules is destined to fail.

The information contained in the database initial study is, to a large extent, the product of interviews with key end users. Those people are the system's main beneficiaries and must be identified carefully. The key users of the University Computer Lab application developed in this appendix are:

- The assistant dean (dean) of the College of Business.
- The computer lab director (CLD), who is charged with the Lab's operational management.
- The computer lab assistants (LAs), who are charged with the Lab's daily operations.
- The computer lab secretary (CLS), who assists in the Lab's general administrative functions.
- The computer lab's graduate assistants (GAs), who work under the lab director to provide technical support and training to faculty and staff using the College of Business resources.

In the interest of brevity, only a few excerpts of the numerous interviews that were undertaken for this project will be shown.

B-1a UCL Objectives

The University Computer Lab (UCL) is in a central location on campus and is accessible by all university students regardless of major. The UCL provides access to many resources, including 200 computers, laser printers, and scanners, to all university members. The UCL provides service and support to a group of users composed of faculty, staff, and students. The Lab's objectives are to:

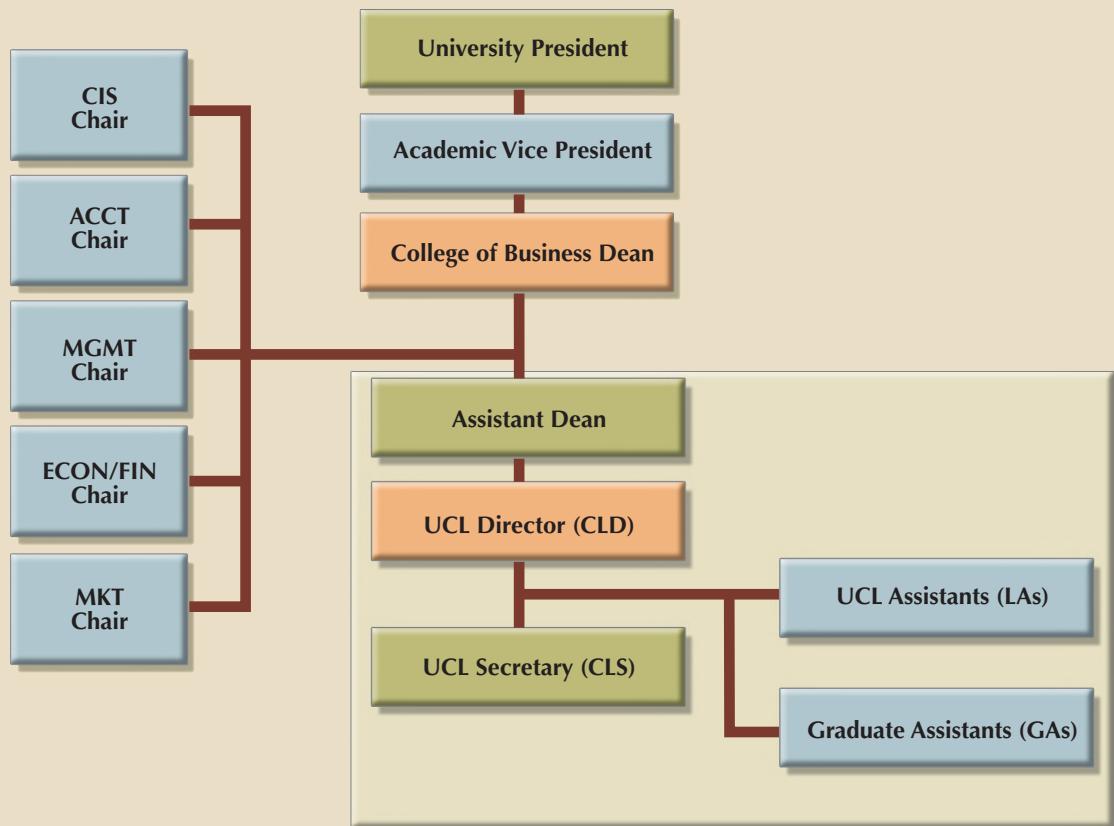
- Provide users with controlled access to the UCL's assets, such as computers, printers, supplies, application software, and software documentation.
- Guide users working with the UCL's assets and provide general problem-solving services. Those services are primarily designed to help users with basic computing operations, such as disk formatting, file copying, (approved), software installation, and basic startup and shutdown procedures.

B-1b Organizational Structure

Understanding the UCL's organizational structure helps the designer define the organization's lines of communication and establish appropriate reporting requirements. (See Figure B.1.)

The computer lab director (CLD) manages all of the UCL's operational functions. The CLD is assisted by the computer lab secretary (CLS). Graduate assistants (GAs) and undergraduate students work in the Lab as lab assistants (LAs). The CLD reports to the assistant dean of the College of Business, who reports to the College of Business dean, who, in turn, reports to the university's academic vice president, who reports to the university president. Although most of the university's chain of command for the College of Business is shown in Figure B.1, the design will focus exclusively on the UCL operations. However, because the other four department chairs receive periodic lab usage summaries, provide feedback to the UCL's director, and contribute to the UCL's funding based on lab usage, they are included in Figure B.1.

FIGURE B.1 THE UCL'S ORGANIZATIONAL STRUCTURE



CIS
ACCT
MGMT

= Computer Information Systems
= Accounting
= Marketing/Management

MKT = Marketing
UCL = University
Computer Lab



Note

The structure shown in Figure B.1 omits details that do not affect the UCL's database design. For example, neither the purchasing department nor the other university colleges and their departments have been shown within the organizational structure because they are not within the UCL's reporting channels. For the same reason, other components of the university's organizational structure are not included in Figure B.1.

Figure B.1 is useful in these ways:

- It facilitates communication between the system's end user(s) and the system's designer(s). Knowledge of the organizational structure helps you define information requirements. (Who needs what information, in what form, and when?)
- It helps system end users specify and clarify areas of responsibility. (Where do I fit into the picture? What is my job?)

Knowing the complete organizational structure is important even when a system is designed for only one component because the system might be expanded later to include other parts of the structure. The designer must keep in mind that different departments might have different, and sometimes conflicting, views of the data and/or the system requirements. The job of the database designer is to develop a common and shared view of data within the organization.

B-1c Description of Operations

Once the UCL's objectives and organizational structure are defined, it is time to study the operations. The UCL has six types of operations. They are organized as inventory/storage/order management, equipment maintenance and repair management, equipment check-out and check-in management, lab assistant payroll management, lab reservations management, and lab access management.

Inventory/Storage/Order Management The UCL's items are classified as hardware, software, literature, and supplies.

- *Hardware* includes computers, terminals, printers, and so on.
- *Software* includes all application programs, such as spreadsheets, word-processing software, statistical software, and database software.
- *Literature* includes reference texts and software manuals.
- *Supplies* include all consumables, such as printer ribbons and paper.

Each inventory item is classified by inventory type, and inventory type is used to group all similar items. The inventory type defines a four-part hierarchy: the inventory category, the class, the type, and the subtype. Table B.2 illustrates the hierarchy.

TABLE B.2

INVENTORY TYPE HIERARCHY

ITEM	CATEGORY	CLASS	TYPE	SUBTYPE
Computer, Intel i5, 2.3 GHz	Hardware	Computer	Desktop	Dual core
Laser printer paper, 8.5" x 11"	Supply	Paper	Laser	8.5" x 11"
CDs, blank rewritable	Supply	CD	Blank	R/W

The database designer(s) and the end users must work together to develop a complete and implementable definition of the appropriate inventory type hierarchy. That collaboration also yields appropriate identification codes and descriptions for each inventory type.

The inventory type also plays an important role in the way inventory items and quantities are recorded. For example, some inventory items do not require individual component tracking. Those inventory items, called **nonserialized items**, include laser printer paper, CDs, and other nondurable supplies. The term *nonserialized* means that the items do not require tracking by an assigned serial number or code. (Keeping track of individual reams of laser printer paper hardly seems appropriate.) On the other hand, durable inventory items, such as computers and printers, require careful tracking with serial numbers or other codes. Therefore, durable inventory items are referred to as **serialized items**.

The assignment of serial numbers or codes enables end users to track an item's location, user, status, and other relevant information. Keep in mind that although hardware is *usually* considered to be a serialized item, the end users and organizational policies create the business rules that define the extent of the serialized and nonserialized classifications.

The inventory's items are updated when:

- An ordered item is received.
- An item is checked out of inventory or checked into inventory by a lab user.
- A consumable item (such as paper or an ink cartridge) is withdrawn from inventory for use.
- The CLD must adjust the inventory. For example, if a physical inventory check reveals that a box of paper is missing, the quantity on hand for that item must be adjusted.

University regulations specify that if a requisition is issued for an amount exceeding \$500, a university-wide committee must approve the purchase. Once approved, the requisition is sent to the purchasing department for bidding and purchasing.



Note

Generic Rule

The university-wide committee requires the CLD to request items without specifying a specific brand and/or vendor unless the CLD can document compatibility problems. You will discover later that such a generic requirement has an effect on the entity attribute selection. For example, you must define equipment by inventory type, as follows:

- Category: Hardware
- Class: Computer
- Type: Desktop
- Subtype: Dual Core

A sample requisition for the proposed purchase of five computers would be written this way: Five (5) computers with the following characteristics: latest generation Intel processor, 19" LCD monitor, minimum 500 GB hard disk, Gigabyte Ethernet card, latest Windows operating system, and MS Office suite.

An exception to the generic rule is made only when an item is purchased under state contract. Approved items not purchased under the state contract are sent out for bids. The purchasing department sends a purchase order to the vendor who makes the winning bid. A copy of the purchase order is sent to the UCL. After receiving the item, the UCL issues a payment authorization to the university accounts payable department for payment of the purchase order.

When the item is received, it might be placed in the UCL to be used or it might be stored. There are several storage locations; each can contain many different kinds of items, and each type of item can be stored in several different locations. For example, three printers might be distributed by storing one in location A and placing the other two in the UCL for immediate use. Supplies are withdrawn from storage as needed.

Equipment Maintenance and Repair Management Computer equipment occasionally malfunctions. Defective equipment is usually repaired by the CLB. If the problem cannot be solved in-house, the equipment is sent to the vendor for repair.

If a piece of equipment requires maintenance, the CLD generates an entry in the Bad Equipment Log. If the equipment must be returned to the vendor for repair, the CLD makes an entry in the Hardware Returned for Service Log.

Equipment Check-Out and Check-In Management Although the Lab's budget and the general administrative responsibility are assigned to the College of Business, any university student, professor, or staff member can use the Lab's facilities. The designer asks the following questions to identify constraints:



Q&A
DESIGNER:

"May equipment be borrowed from the Lab?"

END USER:

"Only professors or staff members may borrow equipment from the Lab. In order to keep a record of equipment location and use, the CLD must check out the equipment. The professor who wants to borrow the equipment must fill out the appropriate form before removing any equipment. The check-out form requires the user to supply a date-out and an estimated date-in. If the equipment has not been returned by the date-in deadline, a notice is sent to the professor whose name appears on the check-out form. Student manuals and data disks may not be borrowed; they are for use only in the Lab."

Lab Assistant Payroll Management The UCL pays lab assistants (LAs) on an hourly basis and keeps track of the total hours worked by each LA during each 14-day pay period. Each LA is assigned a work schedule (the dates and times each LA must work) and must submit a time sheet (showing the hours actually worked) before a paycheck can be issued. The CLD reviews the time sheets and sends them to the payroll department for further processing. Graduate assistants (GAs) are paid a monthly stipend and work a fixed number of hours per week; they are not included in payroll calculations.

Lab Reservations Management The UCL can be reserved by faculty members for teaching purposes. A faculty member fills out a reservation form to reserve the Lab, specifying the date, time, department, and course number of the class to be taught. If an

instructor reserves the Lab for a small class, students not enrolled in that class may use the remaining unoccupied computers *at the instructor's discretion*. Appropriate questions here would be as follows:



Q&A

DESIGNER:

"Are limits placed on how often a faculty member can reserve the Lab?"

END USER:

"No, but given the Lab's limited resources, this may be the time to define limits."



Q&A

DESIGNER:

"How far ahead of time must the Lab be reserved?"

END USER:

"A faculty member must reserve the Lab at least one calendar week ahead of time."



Q&A

DESIGNER:

"Is the lead time OK?"

END USER:

"Yes."

Each reservation covers only one class; the Lab can be used by only one class during its reservation period. Reservations are handled on a first-come, first-served basis and must be approved by the CLD.



Q&A

DESIGNER:

"Is there a daily limit on the number of reserved hours?"

END USER:

"There is currently no policy governing the number of daily Lab reservations. Given the heavy student demand for Lab time, especially during periods when class Lab projects are due, we should place limits on the amount of reserved time. We propose to limit reserved time to one hour in the morning and one hour in the afternoon."

Computer Lab Access Management The UCL is used by students, faculty, and staff members. Upon entering the UCL, the user signs the users' log, located at the LA's desk, and leaves a (valid) University ID card with the LA. When the user leaves the UCL, the LA makes sure that all items checked out by the user (for example, manuals and instructors' data disks) have been returned. If all items have been returned, the LA returns the ID to the user and the user signs out in the log. As long as the UCL is open, there are no time restrictions placed on the user, except when the UCL is reserved for a class.

As you start to understand the operations taking place, you begin to create a Volume of Information Log that estimates the amount of data the system will manage. Table B.3 is an example of such a log. It shows the types of information and the number of entries you expect in designated periods of time.

TABLE B.3
A SAMPLE VOLUME OF INFORMATION LOG

TYPE OF INFORMATION	EXPECTED NUMBER OF ENTRIES PER PERIOD
Lab assistants	14 per semester
Work schedule	8 hours per workday per lab assistant
Hours worked	1 (total hours summary) entry per pay period per lab assistant
Users	
Faculty	300
Students	15,000
Staff	650
Reservations	4 per week
Daily lab users	570 per day
Orders	20 per month
Items ordered	3 per order
Inventory types	15
Locations	5
Repairs	20 per month
Vendors	40

B-1d Problems and Constraints

Once you understand the UCL's operations, you must take stock of the current system's shortcomings. Detailed interviews with key users are likely to reveal operational problems. As you catalog the problems, you should also begin to examine possible causes: poor, inadequate, or absent operational procedures; lack of operational controls; or improper application of existing procedures. Problem-source identification helps the designer develop adequate solutions to problems.

Problems can be *common* (systemwide) or *specific* (pertaining only to portions of the system). The following common problems are identified by UCL key users:

- The manual system is never up to date and yields a constant stream of errors, especially in inventory.
- There is too much data duplication and data inconsistency.
- The manual system does not generate useful information. It's too impractical (time-consuming) to generate reports.

- The system does not allow ad hoc queries.
- The CLD spends too much time manually processing data.
- The lack of a computerized inventory system makes data management difficult. Those data management shortcomings lead to lack of control and restrict the CLD's ability to manage the UCL equipment effectively.

Specific problem areas must be targeted. In the case of the UCL, the following operational problems are identified:

Inventory/Storage/Order Management

- The CLD does not have ready access to crucial inventory management data; for example, what items have been ordered, from what vendor they were ordered, and what items have been ordered but have not been received.
- The UCL needs to know the available stock and average use of supplies, such as paper and printer ink cartridges, to effectively manage the supply inventory, to determine optimal order quantities, and to place necessary orders.
- The CLD does not always know the actual location of an item at any given time. The current system hinders the CLD's ability to track inventory by category, by location, or by manufacturer.

Equipment Maintenance and Repair Management

- The CLD cannot easily generate a repair and maintenance history for each piece of equipment.
- The CLD cannot easily determine the status of items currently subject to maintenance procedures.

Equipment Check-Out/Check-In Management

- The CLD lacks timely and correct information about the Lab's assets: what equipment is checked out, to whom it was checked out, when it was checked out, and so on. Item activity summaries are not available.

Lab Assistant Payroll Management

- The CLD spends too much time reconstructing summaries of hours worked by each LA. The summaries are useful in determining work assignments. The summaries are also necessary to adjust the UCL budget.
- The CLD cannot easily estimate student workloads. Such estimates are necessary to help the CLD distribute work schedules more equitably among the LAs.

Lab Reservations Management

- The manual reservation system is inadequate; it takes too long to check whether desired dates and times are available and to complete the required paperwork.
- The current system does not provide statistical information useful for scheduling Lab reservations.

Computer Lab Access Management

- The user log is not properly maintained.
- Some students do not return certain items. Given inadequate user log entries, too often the LAs do not detect this problem. Items have been lost from the Lab as a result of this lack of control.

- Security problems, ranging from unauthorized network access and unauthorized software installation/deletion to the disappearance of manuals, are also a major concern, and they appear to be increasing.

Given the large number of documented problems, the conclusion is that the current manual system is inadequate. The paperwork tends to be overwhelming, and although reams of data are collected, the data are not readily available. What's more, transforming the data into useful information is usually too time-consuming to be practical.

Problems are solved within two sets of constraints: *operational constraints* imposed by organizational policy and *economic constraints* imposed by the organization's finances.

A well-designed database system should be able to address most of the Lab's stated problems. Consequently, the constraints within which the system is to be designed must be carefully defined.

Time Frame

- The College of Business wants the new system to be fully operational within three months.

Hardware and Software

- The new system must be developed (to the extent possible) with existing UCL hardware and software. The system must be operated on the UCL's existing local area network.

Distributed Aspects and Expandability

- The new system must operate within a multiuser environment.
- The system's operation will be independent of existing administrative systems on campus.

Cost

- The development costs of the new system must be minimal. To reduce expenses and to provide CIS majors with an educational bonus, the system must be developed by CIS majors. To minimize development costs, the design and implementation will be undertaken as a class project under the direction of a faculty member.
- The new system will use no more than two additional terminals to enable the UCL secretary and the CLD to access the system.
- The system must operate without requiring additional personnel in the department.
- Considering budgetary constraints, the College of Business has set aside \$9,500 for the new system's unavoidable expenses.

B-1e System Objectives

After identifying the problems and constraints, the designer and end users cooperate to establish the proposed new system's objectives, giving priority to problems that key users deem most significant. Two sets of objectives are defined for the UCL. First are *general objectives*, which define the overall system requirements. They are as follows:

- Improve operational efficiency, thereby increasing the UCL's capacity and the UCL's ability to expand its operations.
- Provide useful information for planning, control, and security.

Second are *specific objectives*, which define the system component requirements. They are described below.

Inventory/Storage/Order Management

- Provide better control of purchase orders, allowing the CLD to check open orders and purchases.
- Monitor the stock of supply items.
- Control inventory by type (group) as well as by individual item.
- Provide quick and efficient information about the location and status of individual items.
- Provide timely information about the use of supplies and generate the statistical information required to guide the timing and extent of future purchases.

Equipment Maintenance and Repair Management

- Monitor the maintenance history of each item.
- Keep track of items that have been returned to the vendor for repair or replacement.

Equipment Check-Out/Check-In Management

- Keep track of the items that are checked out.
- Monitor the items' check-out time.
- Generate usage statistics for reference purposes.

Lab Assistant Payroll Management

- Provide scheduling and workload information.
- Provide work summaries for each LA.

Lab Reservations Management

- Decrease the time spent processing a reservation.
- Produce reservation schedules.
- Generate statistical summaries by department, faculty, staff member, and date (to be used for planning purposes).

Computer Lab Access Management

- Provide tighter control over users and resources in the Lab.
- Reduce the sign-in time.
- Provide information about peak use times (to be used for scheduling purposes).

B-1f Scope and Boundaries

For legal and practical design reasons, the database designer (and, indeed, the entire development team) cannot work on a system whose operational extent has not been carefully defined and limited—that is, the designer must not work on an unbounded system. If the system limits have not been defined, the designer may be legally required to expand the system indefinitely. In addition, an unbounded system environment will not contain the built-in constraints that make its use practical in a real-world environment.

To define the UCL's database scope and boundaries, the designer must answer the following questions:

1. *What will be the extent of the system?* The database design will cover only the UCL portion of the organizational chart presented in Figure B.1. It will be independent of other database systems currently used on campus.
2. *What operational areas will be covered by the system?* The University Computer Lab system will cover six operational areas (see Section B-1c) and will address the specific objectives listed in Section B-1e. In other words, the system will be limited to addressing the following operational areas:
 - a. Inventory/storage/order management.
 - b. Equipment maintenance and repair management.
 - c. Equipment check-out/check-in management.
 - d. Lab assistant payroll management.
 - e. Lab reservations management.
 - f. Computer lab access management.
3. *What design and implementation strategy should be adopted to bring the system online within the specified time constraints?* To maximize the system's design efficiency, the operational areas should be organized into system modules. A **module** is a design segment that can be implemented as an autonomous unit. Modules may be linked to produce a system. Modules are especially useful because their existence makes it possible to implement and test the system in stages.
4. *What modules must be included in the system?* The operational areas discussed in Question 2 can be classified under two headings: Lab management and inventory management. Therefore, the two modules shown in Table B.4 are appropriate. Note that each module is composed of named processes. For example, the Lab Management System module contains the ACCESS, RESERVATION, and PERSONNEL processes.

module

A design segment that can be implemented as an autonomous unit.

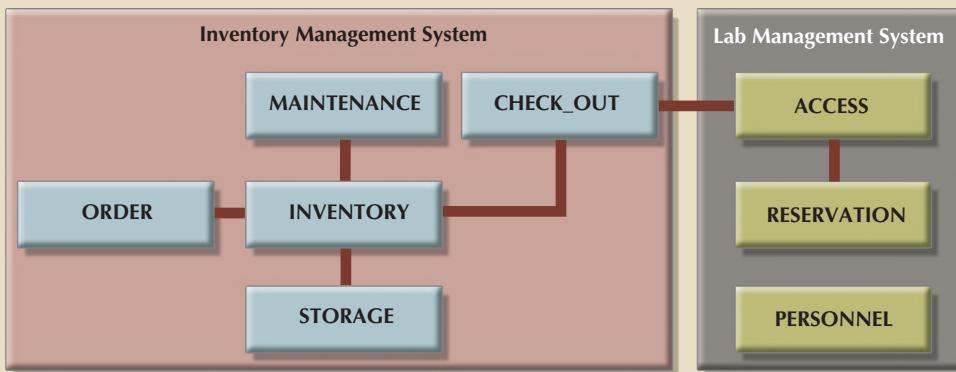
TABLE B.4

REQUIRED UCL SYSTEM MODULES

MODULE	OPERATIONAL AREA	PROCESS NAME
Lab Management System	Computer lab access Reservations Lab assistants' payroll	ACCESS RESERVATION PERSONNEL
Inventory Management System	Inventory Order Storage Equipment maintenance and repair Equipment check-out and check-in	INVENTORY ORDER STORAGE MAINTENANCE CHECK_OUT

5. *How do the modules interface?* The Inventory Management System module's INVENTORY process is the system's key component; its existence enhances the CLD's ability to monitor the Lab's operation and to control the Lab's administrative functions. Figure B.2 shows that the Inventory Management System module interfaces with the Lab Management System module through the CHECK_OUT process.

FIGURE B.2 THE UNIVERSITY COMPUTER LAB MANAGEMENT SYSTEM



Although the INVENTORY process will be independent of other special-purpose inventory systems used on campus, it will use the purchasing department's inventory item classifications. Those classifications facilitate item referencing and querying when users are communicating with purchasing. In addition, using the classifications makes it easy to integrate with a campuswide inventory control system in the future.

The INVENTORY process must permit:

- Registering new inventory types and individual items.
- Keeping track of an item's location, classification, and usage.

The INVENTORY process will interface with the ORDER, STORAGE, MAINTENANCE, and CHECK_OUT processes.

The ORDER process tracks types of inventory items that are ordered from vendors. The system will be designed to track the purchase orders and requisitions placed by the UCL. The ORDER process will interface with the INVENTORY process.

The MAINTENANCE process will track the in-house repairs performed on items, as well as track items returned to the vendor for repair. The MAINTENANCE process also interfaces with the INVENTORY process because items found in inventory may have a repair history.

The CHECK_OUT process will track the items that are checked out by the Lab's users: faculty, staff, and students.

The ACCESS process will help the CLD track the Lab's users. The ACCESS process will interface with the CHECK_OUT process because some items are checked out by students, faculty, or staff members.

The RESERVATION process will track Lab reservations made by faculty or staff members. The process interfaces with:

- ACCESS (because faculty members reserve the Lab).
- PERSONNEL (because an LA records the reservation).

The PERSONNEL process will facilitate the CLD's ability to monitor the LAs' work schedules and actual hours worked. This process interfaces with the RESERVATION process because LAs record Lab reservations.

B-2 Database Design Phase: Conceptual Design

To develop a good conceptual design, you must be able to gather information that lets you accurately identify the entities and describe their attributes and relationships. The entity relationships must accurately reflect real-world relationships.

B-2a Information Sources and Users

The initial study phase generated much useful information from the system's key users. The conceptual design phase must be begun by confirming good information sources. The confirmation process recertifies key users and carefully catalogs all current and prospective end users. In addition, the confirmation process targets the current system's paper flow and documentation, including data and report forms. No document in the paper trail is considered irrelevant at this stage. If the paper exists, somebody must have thought it was important at some point. For the UCL, the following have been confirmed:

- Assistant dean.
- Computer lab director (CLD).
- Computer lab secretary (CLS).
- Computer lab assistants (LA) and graduate assistants (GA).
- Students, faculty, and staff who use the Lab's resources.
- All currently used computer lab forms, file folders, and report forms.

It is not surprising that a list of prospective system users tends to be a duplicate of at least a portion of the list of information sources:

- The CLD (who is also the UCL system administrator) will manage the system, enter data into the database, and define reporting requirements.
- The LA and the GA are the primary UCL system users and will enter data into the database.
- The CLS is a system user and will query and update the database.

You should create a summary table to identify all system sources and users. You can use that table for cross-checking, thereby enabling you to audit sources and users more easily. The UCL system summary is shown in Table B.5. Note that the summary table also identifies the proposed system modules, processes, and interfaces discussed in the previous section.

TABLE B.5

DATA SOURCES AND USERS

MODULE	PROCESS	SOURCES	USERS	INTERFACE
Inventory Management System	INVENTORY			
	Inventory data	Inventory forms, CLD	CLD, CLS, Dean*	Order
	Item data	Inventory forms	CLD, CLS, Dean	Maintenance
	Withdrawal	Inventory forms	CLD, CLS, Dean	Check-out
	Repairs	Bad equipment log	CLD, CLS, Dean	Maintenance
	Check-out	Check-out forms	CLD, CLS, Dean	Check-out
	Location	Inventory forms	CLD, CLS, Dean	Storage
	ORDER			
	Order data	Order forms	CLD, CLS, Dean	Inventory
	Items ordered	Order forms	CLD, CLS, Dean	Inventory
	Items received	Order forms, Inventory	CLD, CLS, Dean	Inventory
	Inventory type	Inventory forms, CLD	CLD, CLS, Dean	Inventory
	Vendors	Order forms	CLD, CLS, Dean	Inventory
	STORAGE			
	Location data	Inventory forms	CLD, CLS	Inventory
	Item data	Inventory forms	CLD, CLS	Inventory
	MAINTENANCE			
	Repair	Bad equipment log	CLD, CLS, GA	Inventory
	Item data	Inventory forms	CLD, CLS, GA	Inventory
	Vendor data	Inventory forms	CLD, CLS, GA	Inventory
	CHECK-OUT			
	Item data	Inventory forms	CLD, CLS, LA, GA	Inventory
	Users	Check-out log	CLD, CLS, LA, GA	Access
Lab Management System	ACCESS			
	User	Lab usage log	CLD, LA	Reservation, Check-out
	RESERVATION			
	Reservation data	Lab reservation forms	CLD, CLS, LA	Access
	PERSONNEL			
	Lab assistants	Lab assistants form	CLD, CLS	Personnel
	Work schedule	Work schedule form	CLD, CLS, LA	Personnel
	Hours worked	Time sheet forms	CLD, CLS, LA	Personnel

CLD = Computer lab director CLS = Computer lab secretary LA = Lab assistant GA = Graduate assistant

* Although the dean is not an active system user, (s)he uses the system reports for decision making.

B-2b Information Needs: User Requirements

A design must match relevant user requirements. *Relevant requirements* are based on the proposed level of information-generating efficiency. The summary of all relevant UCL user requirements yields a general *systems requirements* description, as follows:

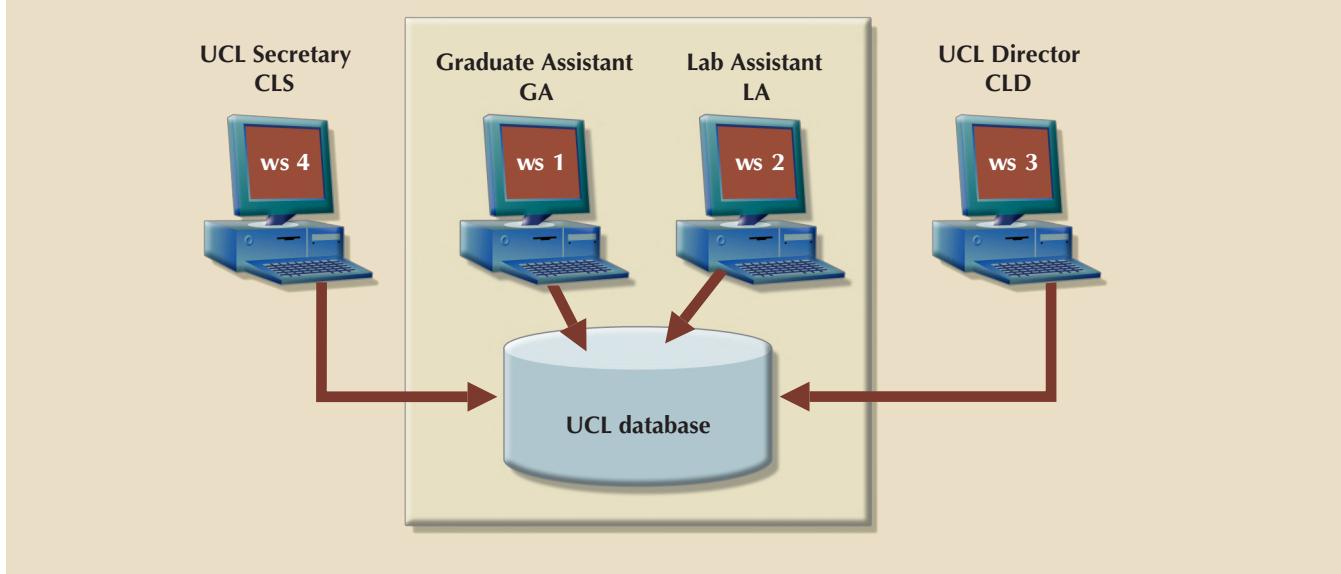
1. *The system must be easy to use.* A menu-driven interface might be most appropriate.
2. *The system must provide security measures* by using passwords and access rights.
3. *The system must be fully integrated,* thus eliminating redundant data entry and redundant updates. The system must ensure database integrity.
4. *Users must be able to access the system concurrently from several workstations.* The workstation location and use must conform to the setup shown in Table B.6. Figure B.3 depicts the University Computer Lab Management System (UCLMS) setup.

TABLE B.6

WORKSTATION ASSIGNMENTS: USES AND USERS

USER	PROCESSES ACCESSED	USE	STATION ID
UCL director (CLD)	All	System administration	WS3
UCL secretary (CLS)	INVENTORY ORDER STORAGE MAINTENANCE CHECK_OUT RESERVATION PERSONNEL	Updates and queries	WS4
Lab assistants (LAs) and Graduate assistants (GAs)	ACCESS RESERVATION CHECK_OUT MAINTENANCE PERSONNEL *	Updates and queries	WS1, WS2

* Restricted access

FIGURE B.3 UCL MANAGEMENT SYSTEM SETUP SUMMARY

5. *The system processes must perform the following functions:*

- PERSONNEL process.* Maintains data for all LAs, their schedules, and their hours worked.
- INVENTORY and storage process.* Controls the stock of items by location as well as by inventory type. The system must also track consumable items by recording their usage (withdrawal). The system must track nonserialized and serialized items.
- ORDER process.* Integrates with the inventory module to establish the relationship between orders and inventory types. The system must generate information about the total orders placed and the total cost of orders by vendor, by order, and by inventory type. It also must be able to generate a grand total cost to be used for budgeting.

- d. *MAINTENANCE process.* Tracks the equipment maintenance history for all hardware. The process must also report items that have been returned to the vendor for replacement or maintenance.
 - e. *RESERVATION process.* Allows the CLD to schedule Lab reservations easily. The system must enable professors and staff to request a reservation online. The system must automatically show the schedule of reservations for the requested day, and it must accept reservations according to the departmental and/or UCL policy.
 - f. *CHECK_OUT process.* Enables the user to track items that are checked out by faculty or staff members for their temporary use.
 - g. *ACCESS process.* Tracks the UCL's usage rate. The system enables LAs to register students who want to use the Lab facilities. The system will retrieve the user identification number (ID) from bar code readers installed on the LAs' main desk computers. The system must also allow students to check out instructors' data disks and software manuals.
6. The system's *input* requirements are, to a major extent, driven by its output requirements—that is, its desired query and reporting capabilities. The reports required by the UCL are shown in Table B.7. The reporting requirements help define appropriate attributes within the entities. Precise report format specifications are a crucial part of the conceptual design process.

TABLE B.7

UCLMS REPORTS

NUMBER	REPORT	DESCRIPTION	USERS
1	Inventory movements	Inventory movements by date and type	CLD, CLS
2	Inventory	Inventory by inventory type	CLD, CLS
3	Location inventory	Inventory of items by location	CLD, CLS
4	Orders	Orders by date, vendor, and status	Dean, CLD
5	Open orders	Open orders by date and vendor	Dean, CLD
6	Orders payable	Orders received but not paid	Dean, CLD
7	Payment history	Orders paid by date and vendor	Dean, CLD
8	Maintenance	Maintenance history by date and item	CLD, GA, LA
9	Check-out	Items checked out by date and user	CLD, CLS
10	LA schedule	Lab assistants schedule	CLD, CLS, GA, LA
11	LA hours worked	Hours worked by lab assistants	CLD, CLS, GA, LA
12	Reservation schedule	Reservations by date and user	CLD, CLS, LA
13	UCL usage statistics	Computer lab usage statistics	Dean, CLD, chairs

B-2c Developing the Initial Entity Relationship Model

From the database initial study and conceptual design preparations, you can identify an initial set of entities. Those entities represent the most important information system objects as viewed by the end user and the designer. Some of the entities represent real-world objects, such as user, lab assistant, item, location, or vendor. Others represent information about entities, such as work schedule, hours worked, repairs, Lab use log, or reservations. The UCL will use the entities shown in Table B.8.

TABLE B.8

INITIAL UCL ENTITIES BASED ON THE INITIAL STUDY

ENTITY NAME	ENTITY DESCRIPTION	ENTITY TYPE
USER	User data: includes administration, faculty, and students	
LAB_ASSISTANT	Lab assistant data: includes graduate assistants	
WORK_SCHEDULE	Lab assistant work schedule data: hours each lab assistant is assigned to work	
HOURS_WORKED	Lab assistant hours worked data: actual hours worked per each pay period for each lab assistant	Weak
LOG	Daily users of the UCL: one entry for each visitor	
RESERVATION	Lab reservation details	
INV_TYPE	Inventory types	
ITEM	Item details	
LOCATION	Storage locations	
REPAIR	Repair data by item	
VENDOR	Vendor details	
ORDER	Order details	

The designer and the end user must agree on the entities. The designer then defines the relationships among the entities, basing them generally on the description of operations (Section B-1c). More specifically, the entity relationships are based on business rules that have been derived from the careful description of operational procedures.

Business rules must be both identified and verified. The UCL database designer conducts a series of interviews with key system users: the University Computer Lab director, who is responsible for the operational administration, and the assistant dean of the computer information systems department, who is responsible for the system's general administration. After the appropriate business rules are identified and incorporated into the ER model, the designer "reads" the model to those individuals to verify its accurate portrayal of the actual and/or proposed operations. The designer also "reads" the ER model to end users to verify that it accurately describes their actions and activities. The verification process may yield additional entities and relationships.

The UCL ER modeling process yields the following summary of business rules, entities, and relationships:

Business Rule 1 Each item belongs to only one inventory type, and each inventory type may have zero, one, or many items belonging to it.

To clarify this business rule, look at the sample data shown in Table B.9. Note that an inventory type is a classification that includes all items within a given category. For example, the Dell Dimension and the Toshiba are both personal computers.

TABLE B.9

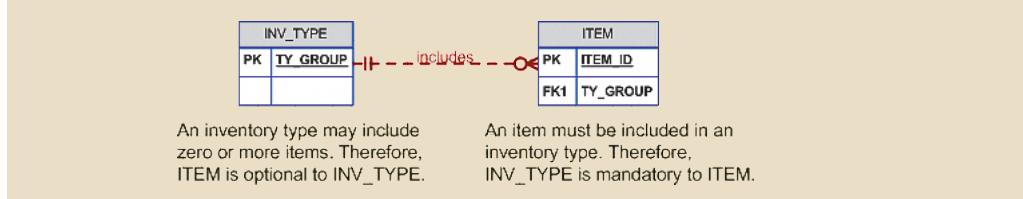
EXAMPLES OF INVENTORY TYPES

INVENTORY TYPE			ITEM		
CATEGORY	CLASS	TYPE	SUBTYPE	ITEM ID	DESCRIPTION
Hardware	Personal computer	Desktop	Dual Processor	3233452	Dell Dimension, 2048 MB RAM, 200 GB hard drive
Hardware	Personal computer	Laptop	Pentium	3312455	Toshiba 1024 MB RAM, 160 GB hard drive
Hardware	Printer	Laser	BW	312246	HP LaserJet IV
Hardware	Printer	Ink-jet	Color	313225	HP 592e color printer
Hardware	Printer	Laser	Color	316757	Xerox Network printer
Supply	Paper	Single sheet	8.5 x 11		Laser printer paper
Supply	CD	Blank	R/W		Recordable CD
Supply	Cartridge	Ink-jet	Color		Color ink-jet cartridge

As you examine the entries in Table B.9, note that each individual item belongs to only one inventory type.

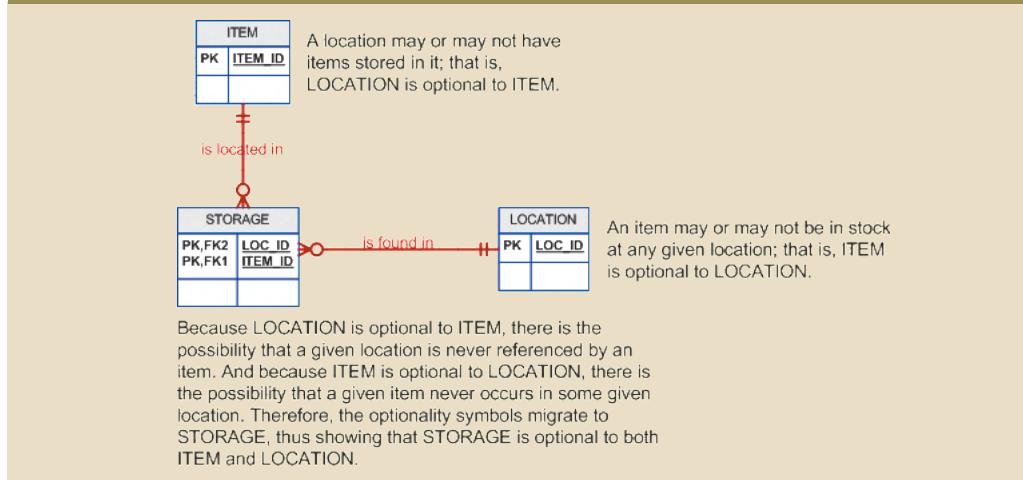
The first business rule leads to the ER model segment shown in Figure B.4.

FIGURE B.4 THE ER MODEL SEGMENT FOR BUSINESS RULE 1

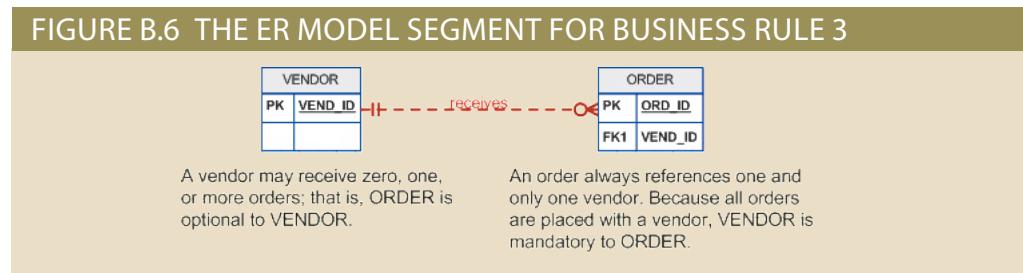


Business Rule 2 An item may be put in use upon its arrival, or it may be stored. In other words, an item might not be stored at all. Some items, such as printer cartridges, are part of a generic grouping and may be stored in more than one location. Therefore, some items could be stored in zero, one, or many locations. Each storage location might store zero, one, or many items. (See Figure B.5.)

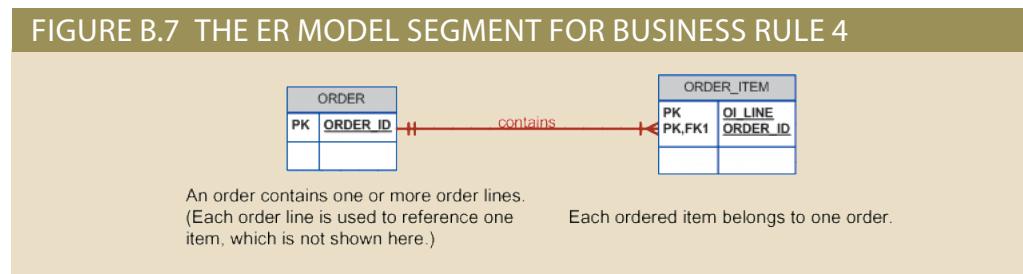
FIGURE B.5 THE ER MODEL SEGMENT FOR BUSINESS RULE 2



Business Rule 3 An order references only one vendor, and each vendor may have zero, one, or many orders. (See Figure B.6.)



Business Rule 4 Each order contains one or many ordered items, and each ordered item line belongs to only one order. (See Figure B.7.)

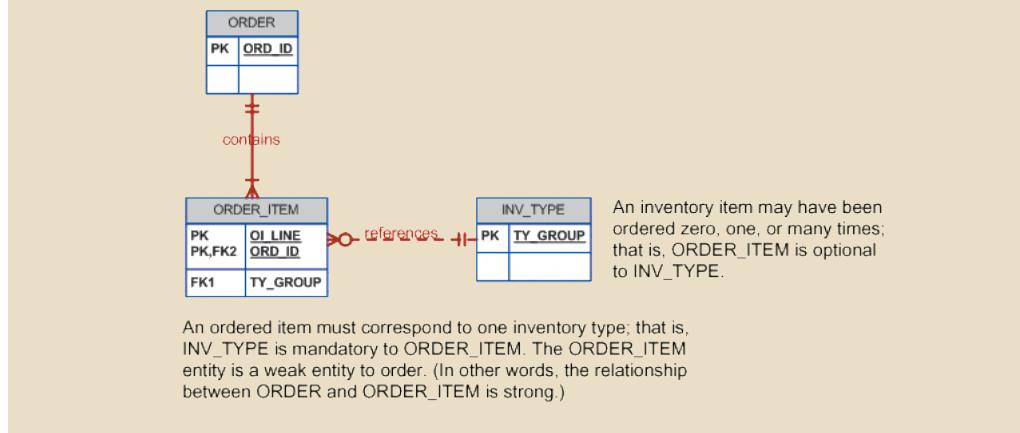


Business Rule 5 Each ordered item line corresponds to one *inventory* type, and each inventory type can be referenced by one or many order item lines. (See Figure B.8.)

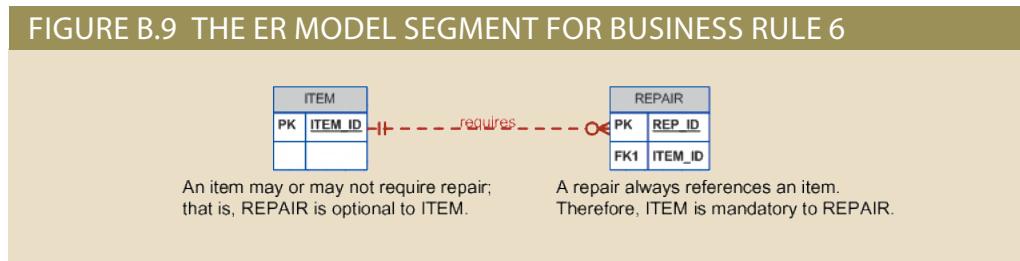
Example:

ORDERED ITEM	PENTIUM COMPUTER	
Inventory type	Category:	Hardware
	Class:	Computer
	Type:	Desktop
	Subtype:	Dual Processor
Item	3233452	Serial number
		Dell

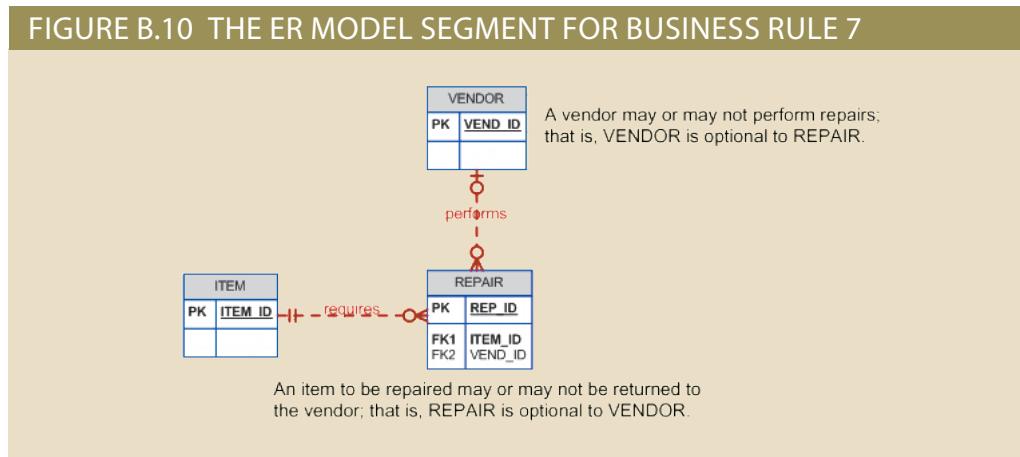
FIGURE B.8 THE ER MODEL SEGMENT FOR BUSINESS RULE 5



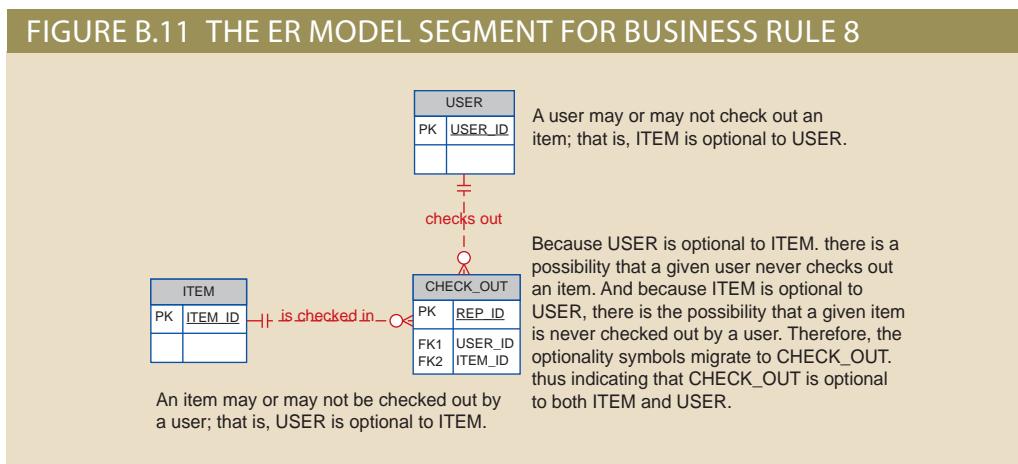
Business Rule 6 Each item may require zero, one, or many repairs, and each repair entry refers to only one item. (See Figure B.9.)



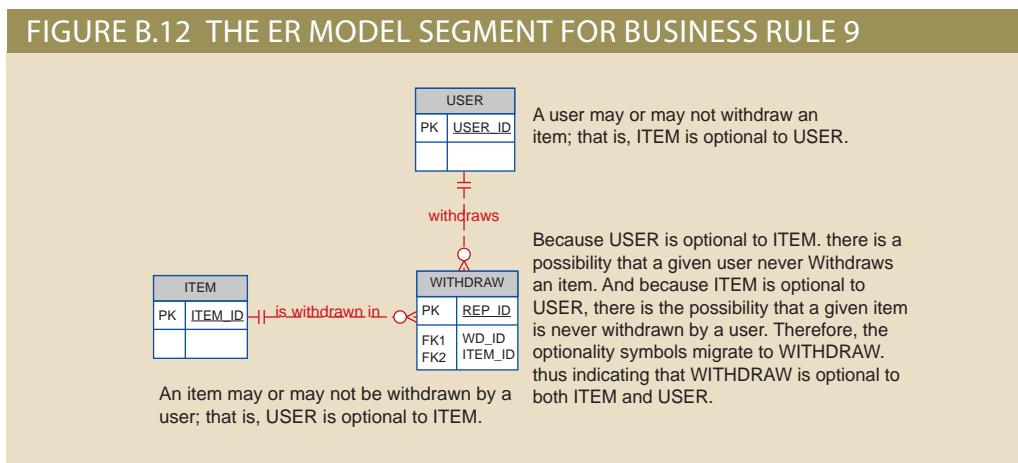
Business Rule 7 Each item to be repaired may or may not be returned to the vendor (the CLD repairs some of them), and each vendor may have zero, one, or many repair items returned. (See Figure B.10.)



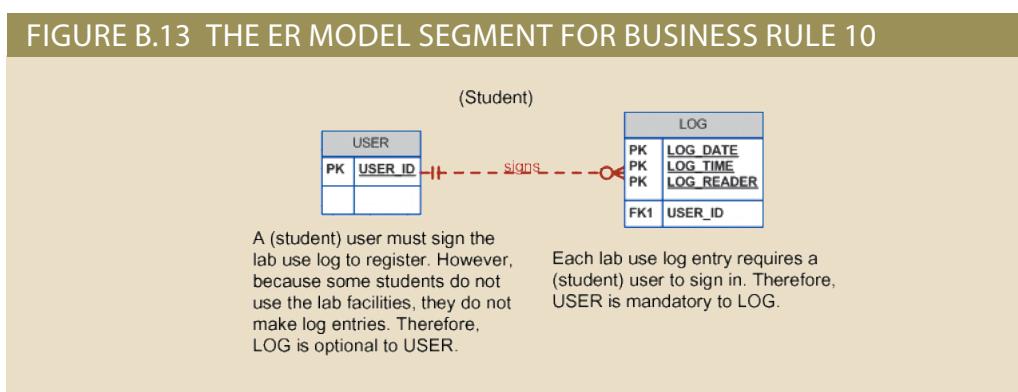
Business Rule 8 Each user may check out zero, one, or many items, and each item may be checked out by zero, one, or many users during the semester. (See Figure B.11.)

FIGURE B.11 THE ER MODEL SEGMENT FOR BUSINESS RULE 8

Business Rule 9 Each (faculty or staff) user may withdraw zero, one, or many items, and each item may be withdrawn by zero, one, or many users during the semester. (See Figure B.12.)

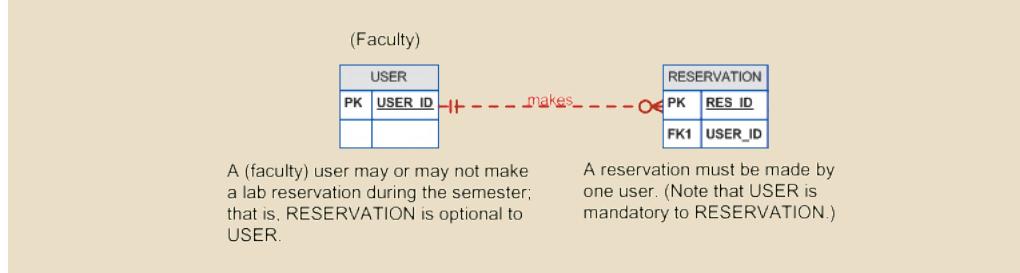
FIGURE B.12 THE ER MODEL SEGMENT FOR BUSINESS RULE 9

Business Rule 10 Each (student) user may sign into the user log many times during the semester, and each user log entry is made by only one (student) user. (See Figure B.13.)

FIGURE B.13 THE ER MODEL SEGMENT FOR BUSINESS RULE 10

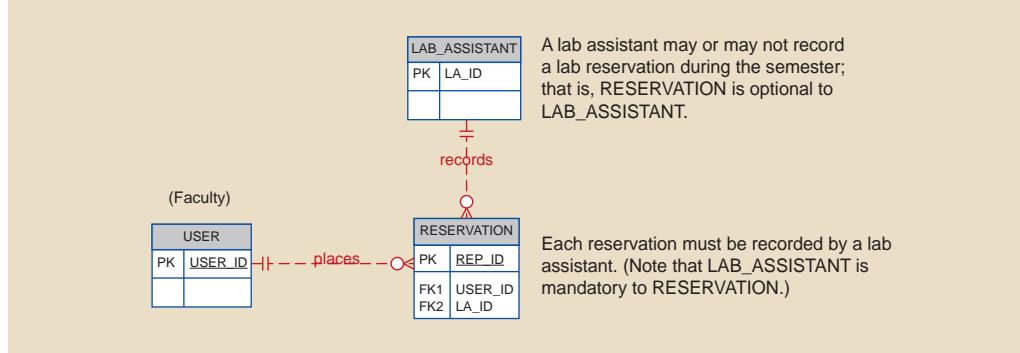
Business Rule 11 Each (faculty) user may place zero, one, or many reservations during the semester, and each reservation is placed by one faculty member. (See Figure B.14.)

FIGURE B.14 THE ER MODEL SEGMENT FOR BUSINESS RULE 11



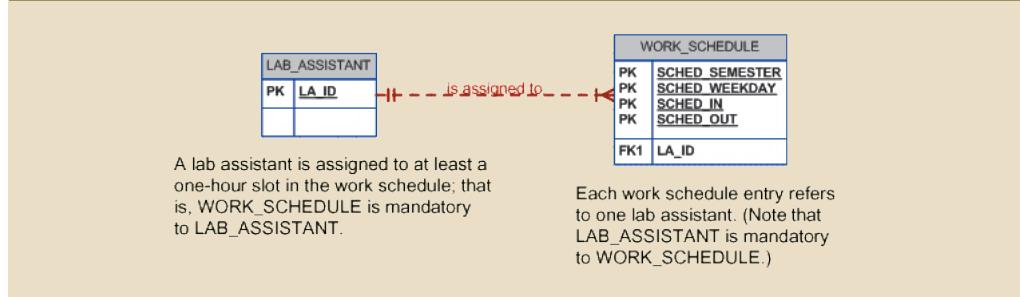
Business Rule 12 Each reservation is recorded by an LA, and each LA may record zero, one, or many reservations during the semester. (See Figure B.15.)

FIGURE B.15 THE ER MODEL SEGMENT FOR BUSINESS RULE 12



Business Rule 13 Each LA is assigned to work at least one day in each week's work schedule, and each work schedule assignment is made for one LA. (See Figure B.16.)

FIGURE B.16 THE ER MODEL SEGMENT FOR BUSINESS RULE 13



Business Rule 14 Each LA accumulates hours worked during each two-week pay period, and each “hours worked” entry is associated with one LA. (See Figure B.17.)

FIGURE B.17 THE ER MODEL SEGMENT FOR BUSINESS RULE 14

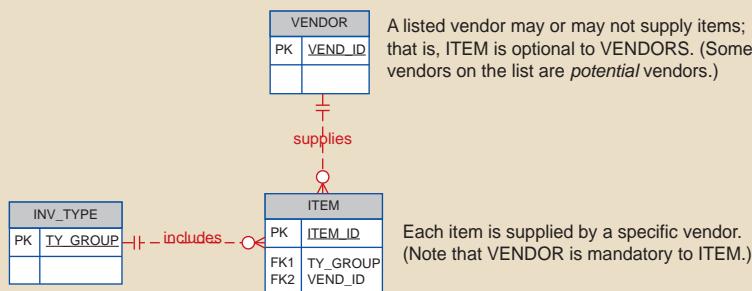


A lab assistant accumulates one or more hours worked per pay period.

Each "hours worked" entry refers to one and only one lab assistant.

Business Rule 15 Each item is supplied by a specific vendor, and each vendor may supply several different items. (See Figure B.18.)

FIGURE B.18 THE ER MODEL SEGMENT FOR BUSINESS RULE 15



(The relationship between ITEM and INV_TYPE was illustrated in Figure B.4, Business Rule 1.)

A listed vendor may or may not supply items; that is, ITEM is optional to VENDORS. (Some vendors on the list are *potential* vendors.)

Each item is supplied by a specific vendor. (Note that VENDOR is mandatory to ITEM.)



Note

Remember that a student can check out items only while (s)he is in the Lab. While such a constraint is written as a business rule, this restriction cannot be represented in the ER diagram; instead, it must be reflected in the program code to conform to the UCL's operational procedures.

Although Tables B.8 and B.10 contain similar information, they reflect different stages in the entity relationship modeling process. Table B.8 shows the initial entity information that is derived from the UCL description of operations. That description is the source of the UCL's business rules. Business rules often generate questions that cause additional entities, relationships, and attributes to be identified. In addition, as the entities and their relationships generate ERD segments, the modeling process may uncover the need for additional entities and/or relationships. The entity information presented in Table B.10 reflects the results of this dynamic modeling process.

Table B.10 summarizes the proposed UCL management system's entities. The ER components identified thus far come together in an ER diagram. Figure B.19 represents the database as seen by the end users and designer at this point.

TABLE B.10

UCL ENTITIES BASED ON THE BUSINESS RULES

ENTITY NAME	ENTITY DESCRIPTION	ENTITY TYPE
USER	User data	
LAB_ASSISTANT	Lab assistant data	
WORK_SCHEDULE	Lab assistant work schedule data	
HOURS_WORKED	Lab assistant hours worked data	Weak
LOG	Daily users of the UCL	
RESERVATION	Lab reservations data	
INV_TYPE	Inventory type data	
ITEM	Items data	
CHECK_OUT	Item check-out data	
WITHDRAW	Supply withdrawal data	
LOCATION	Location in which item is stored	
STORAGE	Item storage data	Composite
REPAIR	Repair data	
VENDOR	Vendor data	
ORDER	Order data	
ORDER_ITEM	Items ordered data	Weak



Note

Business rules are generated from many sources, such as multiple end users, forms, and manuals. Therefore, business rules are not generated in any particular order. For example, you began this appendix's business rule summary by specifying the inventory business rules, you shifted to the end users and their Lab activities, and then you returned to inventory business rules. Those business rules were then converted into ER segments, which were then placed in the framework shown in Figure B.19. If necessary, you can start anywhere in the ER diagram and organize the business rules to match a path you trace through the design. Or you can group the business rules to match the processes. Although this business rule rearrangement may appeal to your desire for organization, it is not required.

Also keep in mind that different end users tend to view data relationships at different levels. For example, note that the M:N relationship between ITEM and LOCATION is represented by a composite entity named STORAGE. (Note that the STORAGE entity's PK consists of the PKs of the related tables, thus making STORAGE a composite entity.) Compare that relationship implementation with one for the M:N relationship between ORDER and INV_TYPB. In the latter case, the ORDER_ITEM entity's PK is composed of OI_LINE and ORD_ID, thereby making the ORDER_ITEM entity weak. Note that the original business rule expressing the M:N relationship between ITEM and LOCATION may be written as:

An ITEM may be stored in many LOCATIONS, and each LOCATION may be used to store many ITEMS.

However, that M:N relationship gives rise to two 1:M relationships that are expressed by these two business rules:

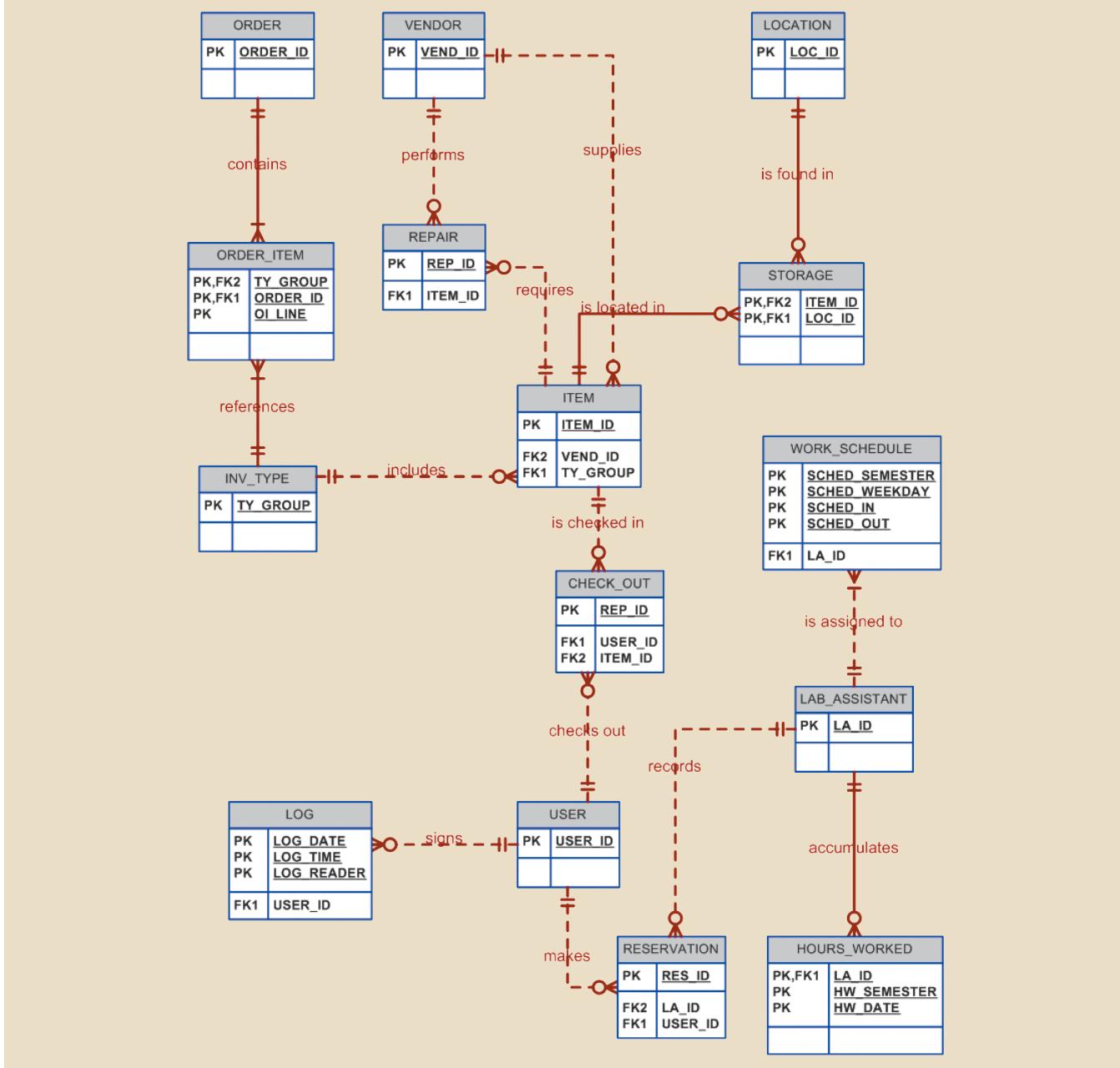
1. Each ITEM may be found one or more times in STORAGE, and each STORAGE (location) *may contain many ITEMS.*

2. Each LOCATION may be referenced one or more times in STORAGE, and each STORAGE entry references one and only one LOCATION.

In short, the database designer must integrate the design components while keeping in mind the following:

- The design is based on multiple information sources.
- The order in which the business rules are developed and yield ER segments is immaterial.
- Different end users view relationships from different perspectives. Thus, the designer must make professional judgments about the way in which those perspectives are reflected in the database design.

FIGURE B.19 THE UCL MANAGEMENT SYSTEM'S INITIAL ERD



Key Terms

module, B-13

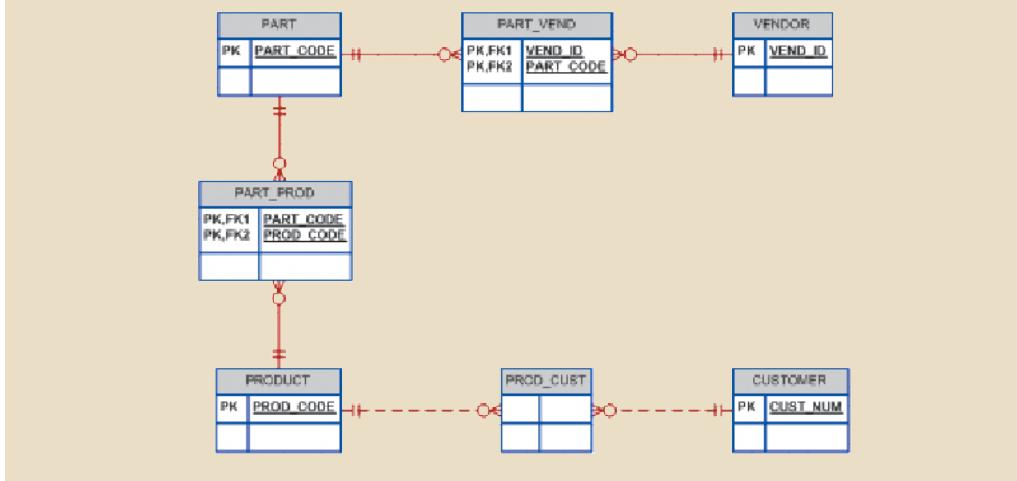
nonserialized items, B-6

serialized items, B-6

Review Questions

1. What factors relevant to database design are revealed during the initial study phase?
2. Why is the organizational structure relevant to the database designer?
3. What is the difference between the database design scope and its boundaries? Why is the scope and boundary statement so important to the database designer?
4. What business rule(s) and relationships can be described for the ERD shown in Figure QB.4?

FIGURE QB.4 THE ERD FOR QUESTION 4



5. Write the connectivity and cardinality for each of the entities shown in Question 4.
6. What is a module, and what role does a module play within the system?
7. What is a module interface, and what does it accomplish?

Problems

1. Modify the initial ER diagram presented in Figure B.19 to include the following entity supertype and subtypes: The University Computer Lab **USER** may be a *student* or a *faculty member*.
2. Using an ER diagram, illustrate how the change you made in Problem 1 affects the relationship of the **USER** entity to the following entities:
 - a. LOG
 - b. RESERVATION
 - c. CHECK_OUT
 - d. WITHDRAW

3. Create the initial ER diagram for a car dealership. The dealership sells both new and used cars, and it operates a service facility. Base your design on the following business rules:
- A salesperson can sell many cars, but each car is sold by only one salesperson.
 - A customer can buy many cars, but each car is sold to only one customer.
 - A salesperson writes a single invoice for each car sold.
 - A customer gets an invoice for each car (s)he buys.
 - A customer might come in only to have a car serviced; that is, one need not buy a car to be classified as a customer.
 - When a customer takes in one or more cars for repair or service, one service ticket is written for each car.
 - The car dealership maintains a service history for each car serviced. The service records are referenced by the car's serial number.
 - A car brought in for service can be worked on by many mechanics, and each mechanic can work on many cars.
 - A car that is serviced may or may not need parts. (For example, parts are not necessary to adjust a carburetor or to clean a fuel injector nozzle.)
4. Create the initial ER diagram for a video rental shop. Use (at least) the following description of operations on which to base your business rules.

The video rental shop classifies movie titles according to their type: comedy, western, classical, science fiction, cartoon, action, musical, or new release. Each type contains many possible titles, and most titles within a type are available in multiple copies. For example, note the summary in the following table of the relationship between video rental type and title.

TYPE	TITLE	COPY
Musical	My Fair Lady	1
	My Fair Lady	2
	Oklahoma!	1
	Oklahoma!	2
	Oklahoma!	3
Cartoon	Dilly Dally & Chit Chat Cat	1
	Dilly Dally & Chit Chat Cat	2
	Dilly Dally & Chit Chat Cat	3
Action	Amazon Journey	1
	Amazon Journey	2

Keep the following conditions in mind as you design the video rental database:

- The movie type classification is standard; not all types are necessarily in stock.
- The movie list is updated as necessary; however, a movie on that list might not be ordered if the video shop owner decides that the movie is not desirable for some reason.
- The video rental shop does not necessarily order movies from all vendors on the vendor list; some vendors on the vendor list are merely potential vendors from whom movies may be ordered in the future.
- Movies classified as new releases are reclassified to an appropriate type after they have been in stock for more than 30 days. The video shop manager wants to have an end-of-period (week, month, year) report for the number of rentals by type.

- If a customer requests a title, the clerk must be able to find it quickly. When a customer selects one or more titles, an invoice is written. Each invoice can contain charges for one or more titles. All customers pay in cash.
 - When a customer checks out a title, a record is kept of the check-out date and time and the expected return date and time. When rented titles are returned, the clerk must be able to check quickly whether the return is late and to assess the appropriate late return fee.
 - The video store owner wants to generate periodic revenue reports by title and by type. The owner also wants to generate periodic inventory reports and track titles on order.
 - The video store owner, who employs two (salaried) full-time and three (hourly) part-time employees, wants to keep track of all employee work time and payroll data. Part-time employees must arrange entries in a work schedule, while all employees sign in and out on a work log.
5. Suppose a manufacturer produces three high-cost, low-volume products: P1, P2, and P3. Product P1 is assembled with components C1 and C2; product P2 is assembled with components C1, C3, and C4; and product P3 is assembled with components C2 and C3. Components may be purchased from several vendors, as shown in the following table.

VENDOR	COMPONENT SUPPLIED
V1	C1, C2
V2	C1, C2, C3, C4
V3	C1, C2, C4

Each product has a unique serial number, as does each component. To track product performance, careful records are kept to ensure that each product's components can be traced to the component supplier.

Products are sold directly to final customers; that is, no wholesale operations are permitted. The sales records include the customer identification and the product serial number. Using the preceding information, do the following:

- a. Write the business rules governing the production and sale of the products.
- b. Create an ER diagram capable of supporting the manufacturer's product/component tracking requirements.
6. Create an ER diagram for a hardware store. Make sure you cover (at least) store transactions, inventory, and personnel. Base your ER diagram on an appropriate set of business rules that you develop. (*Note:* It would be useful to visit a hardware store and conduct interviews to discover the type and extent of the store's operations.)
7. Use the following brief description of operations as the source for the next database design.

All aircraft owned by ROBCOR require periodic maintenance. When maintenance is required, a maintenance log form is used to enter the aircraft's identification number, the general nature of the maintenance, and the maintenance starting date. A sample maintenance log form is shown in Figure PB.7A.

FIGURE PB.7A THE MAINTENANCE LOG FORM

ROBCOR Aircraft Service page 1 of 1

Log #: 2155 Aircraft: 2155W Date in: 18-Jan-2016
Checked in by: George D. Ramsey (115)



Squawk summary

1. Left mag rough on run-up _____
2. Nose gear shimmies at taxi speeds _____
3. Left main gear door does not close flush with wing panel _____
4. Gear struts do not maintain proper pressure _____
5. Left engine vibrates when power is pulled back to 20 in. manifold pressure _____
6. _____
7. _____
8. _____
9. _____
10. _____

Aircraft release date: 19-Jan-2016 Released by: Bea L. Patterson (109)

FIGURE PB.7B THE MAINTENANCE LINE FORM

ROBCOR Aircraft Service		page 1 of 1			
		Log #: 2155		© noppasit TH/Shutterstock.com	
Item	Action Description	Time	Part	Units	Mechanic
1	Performed run-up Rough mag reset	0.8	None	0	112
2	Cleaned #2 bottom plug, left engine	0.9	None	0	112
3	Replaced nose gear shimmy dampener	1.3	P-213342A	1	103
4	Replaced left main gear door oleo strut seal	1.7	GR/311109S	1	112
5	Cleaned and checked gear strut seals	1.7	None	0	116
6					
7					
8					

Parts used in any maintenance action must be signed out by the mechanic who used them, thus allowing ROBCOR to track its parts inventory. Each sign-out form contains a listing of all parts associated with a given maintenance log entry. Therefore, a parts sign-out form contains the maintenance log number against which the parts are charged. In addition, the parts sign-out procedure is used to update the ROBCOR parts inventory. A sample parts sign-out form is shown in Figure PB.7C.

Mechanics are highly specialized ROBCOR employees, and their qualifications are quite different from those of an accountant or a secretary, for example.

Given this brief description of operations and using the Chen ER methodology, draw the fully labeled ER diagram. Make sure you include all appropriate relationships, connectivities, and cardinalities.

8. You have just been employed by the ROBCOR Trucking Company to develop a database. To gain a sense of the database's intended functions, you spent some time talking to ROBCOR's employees and you examined some of the forms used to track driver assignments and truck maintenance. Your notes include the following observations:

- Some drivers are qualified to drive more than one type of truck operated by ROBCOR. A driver may, therefore, be assigned to drive more than one truck type during some period of time. ROBCOR operates several trucks of a given type. For example, ROBCOR operates two panel trucks, four half-ton pick-up trucks, two single-axle dump trucks, one double-axle truck, and one 16-wheel truck. A driver with a chauffeur's license is qualified to drive only a panel truck and a half-ton pick-up truck and, thus, may be assigned to drive any one of six trucks. A driver with a commercial license with an appropriate heavy equipment endorsement may be assigned to drive any of the nine trucks in the ROBCOR fleet. Each time a driver is assigned to drive a truck, an entry is made in a log containing

the employee number, the truck identification, and the sign-out (departure) date. Upon the driver's return, the log is updated to include the sign-in (return) date and the number of driver duty hours.

FIGURE PB.7C THE PARTS SIGN-OUT FORM

ROBCOR Aircraft Service				
page 1 of 1				
Log #: 2155				
Form sequence #: 24226				
 <small>© noppasit TH/Shutterstock.com</small>				
Part	Description	Units	Unit Price	Mechanic
P-213342A	Nose gear shimmy dampener, PA31-350/1973	1	\$189.45	112
GR/311109S	Left main gear door oleo strut seal, PA31-350/1973	1	\$59.76	103

- If trucks require maintenance, a maintenance log is filled out. The maintenance log includes the date the truck was received by the maintenance crew. The truck cannot be released for service until the maintenance log release date has been entered and the log has been signed off by an inspector.
- All inspectors are qualified mechanics, but not all mechanics are qualified inspectors.
- Once the maintenance log entry has been made, the maintenance log number is transferred to a service log in which all service log transactions are entered. A single maintenance log entry can give rise to multiple service log entries. For example, a truck might need an oil change as well as a fuel injector replacement, a brake adjustment, and a fender repair.
- Each service log entry is signed off by the mechanic who performed the work. To track the maintenance costs for each truck, the service log entries include the parts used and the time spent to install the part or to perform the service. (Not all service transactions involve parts. For example, adjusting a throttle linkage does not require the use of a part.)
- All employees are automatically covered by a standard health insurance policy. However, ROBCOR's benefits include optional copaid term life insurance and disability insurance. Employees may select both options, one option, or no options.

Given those brief notes, create the ER diagram. Make sure you include all appropriate entities and relationships and define all connectivities and cardinalities.

Appendix C

The University Lab: Conceptual Design Verification, Logical Design, and Implementation

Preview

This appendix will *verify* the ER model developed in Appendix B, “The University Lab: Conceptual Design.” **Verification** represents the link between the database modeling and design activities and the database applications design. Therefore, the verification process requires that you identify and define all database transactions (insert, update, delete, and outputs) and be flexible enough to support expected enhancements and modifications.

The verification process will also enable the designer to find and eliminate unnecessary data redundancies, to help ensure database integrity, to discover appropriate enhancements, and to verify that all stated end-user requirements are met. The verification process includes the integration of all of the different end-user views of the database, each with its own set of requirements and transactions. In this appendix, as in the real world, the verification process leads to modifications in the initial ER model. This verification process makes use of normalization procedures that are usually considered to be part of the *logical design* phase. However, in a real-world environment, the verification process generally uses modeling and normalization procedures *concurrently*. The modifications may include the creation and/or deletion of new entities, additional attributes in existing entities, and relationships.

Before the verification process can begin, you must identify and define all attributes and domains for each entity in the initial ER model and normalize the entities. You must also select a proper primary key and place foreign keys to link the entities. After completing the verification process, you finish the design process by formulating the logical and physical models.

verification

The process of refining a conceptual data model into a detailed design that is capable of supporting all required database transactions, and input and output requirements.

Data Files and Available Formats

MS Access **Oracle** **MS SQL** **My SQL**

MS Access **Oracle** **MS SQL** **My SQL**

There are no data files for this appendix.

Data Files Available on cengagebrain.com

C-1 Completing the Conceptual and Logical Database Designs

The conceptual database blueprint developed in Appendix B is still in a rough-draft format. Although it helps you define the basic characteristics of the database environment, the design lacks the details that allow you to implement it effectively. Using an analogy, if an architect's blueprint shows a wall, it is important to know whether that wall will be made of board, brick, block, or poured concrete and whether that wall will bear a load or merely act as a partition. In short, *detail matters*.

Before continuing, you might find it helpful to review the database life cycle (DBLC) in Chapter 9 Database Design. Doing this will help you evaluate what is accomplished in Appendix B and determine what remains to be done. In Appendix B, you completed:

- Phase 1 (the database initial study) of the database life cycle (DBLC).
- The initial pass through the DBLC's Phase 2 (the database design phase). That is, in Appendix B, you identified, analyzed, and refined the business rules; identified the main entities; and identified the relationships among those entities.

In this appendix, you will complete the conceptual and logical designs for the University Computer Lab's database. The physical design elements will be presented, and you will examine the issues to be confronted in the implementation phase. Table C.1 shows the specific tasks addressed.

TABLE C.1

TASKS ADDRESSED IN THIS CHAPTER

TASK	SECTION
Entity relationship modeling and normalization	Section C-2
Data model verification	Section C-3
Logical design	Section C-4
Physical design	Section C-5
Implementation	Section C-6
Testing and evaluation	Section C-7
Operation	Section C-8

The initial ER diagram in Appendix B (Figure B.19) will serve as the starting point. In other words, you will use the initial design as the basis for attribute definition, table normalization, and model verification to see if the design meets processing and information requirements. Keep in mind that the activities described are often concurrent and iterative. That is, they often take place simultaneously and are often repeated. For example, the definition of entities and their attributes is subject to normalization, which can generate additional entities and attributes, which are subject to normalization. If done properly, that process will yield an ER model whose entities, attributes, and relationships are capable of supporting the end-user data, information, and processing requirements.

To facilitate the completion of the conceptual model, you will use two modules, each supporting a functional area of the University Computer Lab. Those two modules, first introduced in Appendix B, Table B.4, are the

- Lab Management System, which reflects the Lab's daily operations. This module targets the Lab's users, the people who work in the Lab, and the scheduling of Lab resources. This module allows the computer lab director (CLD) to track the Lab's resources by user type, department, and so on. Such tracking will be an important resource when the Lab's budget is written.
- Inventory Management System, in which the equipment, supplies, orders, and repairs are tracked. (For example, equipment sent out for repair is temporarily *removed from* inventory, while repaired equipment is *returned to* inventory.) This module also allows the CLD to track equipment that is temporarily checked out for use by faculty members and staff.

A list of the entities identified during this process, as well as the attribute prefixes used, is shown in Table C.2.

TABLE C.2

THE UCL DESIGN ORGANIZATION

MODULE	ENTITIES	ATTRIBUTE PREFIX
Lab Management System	USER LOG LAB_ASSISTANT WORK_SCHEDULE HOURS_WORKED RESERVATION RES_SLOT	USER_ LOG_ LA_ SCHED_ HW_ RES_ RSLOT_
Inventory Management System	INV_TYPE ITEM STORAGE LOCATION REPAIR VENDOR ORDER ORDER_ITEM WITHDRAW WD_ITEM CHECK_OUT CHECK_OUT_ITEM INV_TRANS TR_ITEM	TY_ ITEM_ STOR_ LOC_ REP_ VEND_ ORD_ OI_ WD_ WI_ CO_ CI_ TRANS_ TI_

As you compare the entities listed in Table C.2 with the initial database design shown in Appendix B, you will note that several new entities have been introduced. For example, RES_SLOT has emerged because a single RESERVATION might trigger more than one reservation date and time. For example, on 11-Jan-2018, a professor might make three Lab reservations: from 8:00 a.m.–8:50 a.m. and from 1:00 p.m.–1:50 p.m. on 23-Jan-2018 and from 6:00 p.m.–8:40 p.m. on 25-Jan-2018. Therefore, there is a 1:M relationship between RESERVATION and RES_SLOT. The new entities will be discussed as you develop each module within the system. You will also discover that some of the entities shown in Table C.2 will be replaced by other entities during the revision process.

C-2 Completing the Conceptual Design: Entities, Attributes, and Normalization

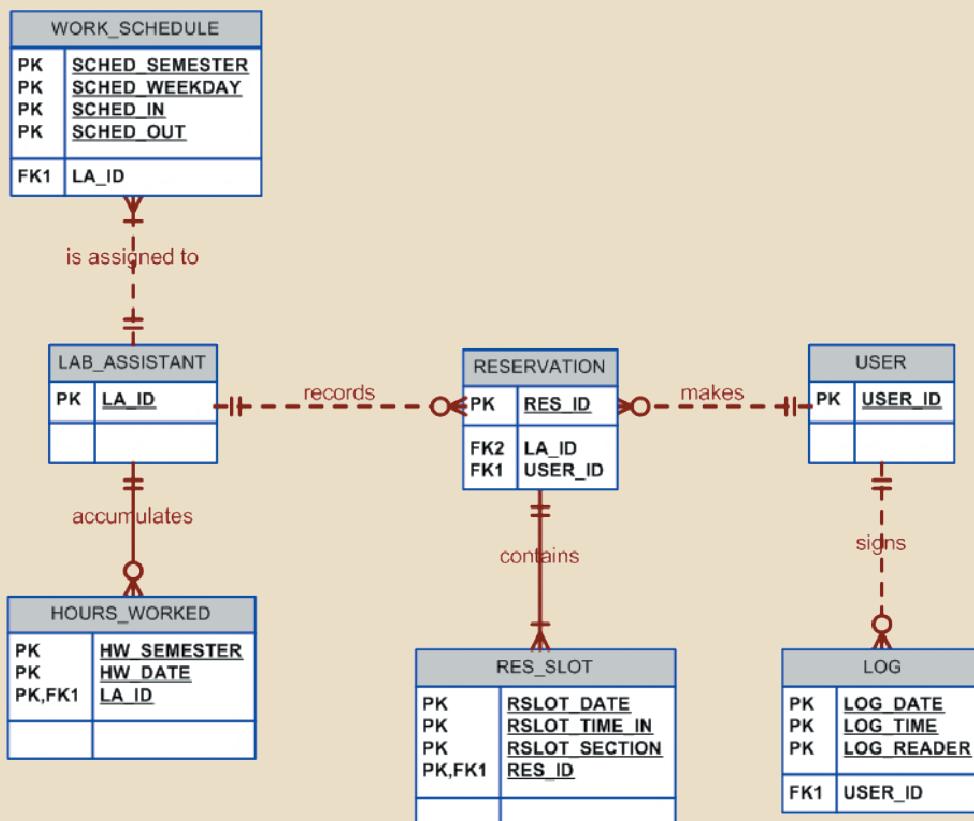
Section C-1 described two system modules. Each module's entities and their attributes will be defined next. Even as they are defined, entities and attributes are subject to the revisions that are often triggered by normalization. In other words, normalization is treated as an integral part of the ER modeling process. Therefore, functional dependencies are monitored carefully. Normalization techniques are used to discover new entities and some practical ways to evaluate their functions. The entities and their attributes are also subject to revision as they are evaluated in terms of end-user requirements.

The normalization techniques will not be covered again in this appendix. The structures presented here, however, have all been subjected to proper evaluation of their normalization levels. Your knowledge of the normalization principles and techniques will be necessary as you create and revise the entities and their attributes. As the revised model is developed, keep in mind the often conflicting requirements of design elegance, information requirements, and processing speed.

C-2a The Lab Management System Module

Before examining the structure of each of the Lab Management System module components, let's look at the ER segment presented in Figure C.1.

FIGURE C.1 THE LAB MANAGEMENT SYSTEM MODULE'S ER SEGMENT



The ER segment in Figure C.1 will be used as a map to track where you are in the process and where you are going. Using Figure C.1 and Table C.2 as a guide, let's begin by examining the USER entity's characteristics, shown in Table C.3.

TABLE C.3

THE USER ENTITY

ATTRIBUTE NAME	CONTENTS	ATTRIBUTE TYPE (*)	PRIMARY KEY (PK) AND/OR FOREIGN KEY (FK)	REFERENCES
USER_ID	User identification code		PK	
DEPT_CODE	Department code			
USER_TYPE	User type: Fac = Faculty Staff = Staff Stu = Student			
USER_CLASS	User class: UG = Undergraduate GR = Graduate Fac = Faculty Staff = Staff			
USER_GENDER	M = Male F = Female			

* The attribute type may be Composite (C), Derived (D), or Multivalued (M).



Note

To let you focus on the relationships among the entities, all of the ERDs used in this appendix will show only the PK and FK attributes for each of the entities. You may, of course, add the additional attributes that will be defined for you in each of the summary tables.

If you use Visio Professional or any similar CASE tool to design the database, remember that you create only an entity's PK attribute at the *entity* level if its *relationship* to its parent entity is non-identifying. You never define the FK at the *entity* level for any entity, no matter what its relationship(s) to other entities. Instead, first create the *entities* and their PK attributes—as long as none of those PK attributes is inherited from related entities. If you use Visio Professional, attaching the relationship lines will ensure that the following actions are taken:

- All of the FKs will be inserted into the entities to properly reflect the relationships that you have defined for those entities.
- All of the PK components that are inherited from related entities will be properly inserted into each entity that requires the use of such inherited PK attribute(s).
- The FKs will always inherit the attribute characteristics from the PKs to which they point. That means you will never see an “incompatible data type” error message when you try to implement the design.
- The software will automatically check for inconsistent relationships, circular relationships, and incorrectly defined relationships. Therefore, you will, in effect, have a built-in design quality control feature.

Most CASE tools provide such PK/FK design services, and you should make use of those services when they are available. After all, the objective is to produce a clean design that can be implemented successfully. Once you know that the design contains no logical or implementation-level flaws, you can add the remaining attributes.

Naturally, if you are doing the preliminary design using pencil and paper—you are using a design tool that does not provide the just-described services—you will have to write all of the PK attributes and FK attributes at the entity level so you can see what the implementation implications are. In effect, you will be serving the same role as the “update foreign keys” function you’ll find in most advanced database design software.

As you examine Table C.3, note that the DEPT_CODE attribute has been added, which lets the CLD track which departments use the Lab facilities. That information is important because departments share the Lab's budgeting equation; departments using the Lab more frequently contribute more to its operation than departments using the Lab less frequently. Therefore, the ability to count Lab use by department code is important. Although the DEPT_CODE is clearly a foreign key to a DEPARTMENT entity, there is no information requirement at this time for more detailed departmental data. Therefore, DEPARTMENT will not be included in this design, and the DEPT_CODE in the USER entity has not been designated as an FK at this point.



Note

The USER entity was initially created to prototype the system and to make sure that a working database could be supplied within this appendix. In the USER table, USER_ID, DEPT_CODE, and USER_TYPE can be used to summarize Lab usage for budgeting purposes.

As in most real-world database designs, the University Computer Lab Management System's (UCLMS) actual user data are part of an existing external database. That external database is controlled and maintained by the university. Its data entry procedures and structures are different from those addressed in the UCLMS design. For example, the Lab's user data are entered through a magnetic card reader that targets many more variables than are included here. Adding those variables to the design shown in this appendix would not add insight into the crucial design verification process on which you want to focus. For the same reason, the entities DEPARTMENT and COLLEGE were not included. (The inclusion of the DEPT_CODE in USER is sufficient to track Lab usage without having to access departmental and college details.)

Also, the data used in this appendix constitute a very small subset of the actual data. For example, the real system currently records over 30,000 Lab-use entries per semester, and that tally is growing rapidly. Finally, to conform to privacy requirements, all data values have been simulated.

Note that the USER table structure produces some redundancy. For example, USER_CLASS clearly determines the USER_TYPE. If you know that USER_CLASS = UG, you also know that USER_TYPE is Stu. On the other hand, USER_TYPE is not a determinant of USER_CLASS because Stu can mean either UG or GR. In any case, you now know that the table is in 2NF. To eliminate the 2NF condition, you could combine USER_TYPE and USER_CLASS into a single attribute represented by a string to portray Stu/UG, Stu/GR, Fac, and Staff. However, because the university requires a report that shows Lab usage summaries by faculty, staff, and students, the current table structure is desirable. Additionally, the report requires a breakdown by various student subcategories (graduate/undergraduate, male/female). Real-world database design often requires a trade-off between information efficiency and design purity. Some sample USER data are shown in Figure C.2.

As you examine the data shown in Figure C.2, you should note that when the USER_TYPE is Fac or Staff, the USER_CLASS is also Fac or Staff. That duplication serves reporting requirements well because it enables you to generate USER_CLASS

summaries easily. Finally, note that hyphens have been used in the USER_ID data. Social Security numbers are read more easily when hyphens are used, and the cost of including hyphens in the string is only 2 bytes per entry. Data storage is cheap and getting cheaper, so the extra 2 bytes per USER_ID entry do not create much of a burden. On the other hand, if the data search is keyed to the User ID, the search speed is enhanced when the dashes are not included in an alphanumeric attribute. Modern database systems do provide the designer with an option to use input masks for presentation purposes (9-99-9999, rather than 9999999) *without* storing the dashes in the database table. As always, database professionals are expected to use sound judgment to balance competing requirements.

FIGURE C.2 SAMPLE USER DATA

USER_ID	DEPT_CODE	USER_TYPE	USER_CLASS	USER_GENDER
1-11-1111	CIS	Fac	Fac	F
1-11-1112	CIS	Fac	Fac	M
1-11-1113	ACCT	Staff	Staff	F
1-11-1114	ACCT	Fac	Fac	F
1-11-1115	BIOL	Staff	Staff	M
1-11-1116	MKT/MGT	Fac	Fac	M
1-11-1117	CIS	Fac	Fac	M
1-11-1118	SOC	Stu	GR	F
1-11-1119	ACCT	Fac	Fac	F
1-11-1120	ACCT	Fac	Fac	F
1-11-1121	CIS	Fac	Fac	F
1-11-1122	CIS	Stu	UG	F
1-11-1123	CIS	Stu	UG	F

Following the module layout in Table C.2, the LOG entity is examined next, represented by the LOG table. Each time a USER accesses the Lab facilities, that user's identification is read into the log by one of two magnetic card readers.



Note

To prototype the system and keep the database self-contained, the USER entry procedure has been modified. For example, if a USER_ID does not match a record in the USER table, the USER table is updated by the addition of the new user. Keep in mind that the real system must provide security, so it must refuse entry to a user whose identification does not match an existing record in the externally managed student database.

The LOG entity details are shown in Table C.4.

TABLE C.4 THE LOG ENTITY				
ATTRIBUTE NAME	CONTENTS	ATTRIBUTE TYPE (*)	PRIMARY KEY (PK) AND/OR FOREIGN KEY (FK)	REFERENCES
LOG_DATE	Log-in (system) date		PK	
LOG_TIME	Log-in (system) time		PK	
LOG_READER	Magnetic card reader number		PK	
USER_ID	User identification (ID)		FK	USER

* The attribute type may be Composite (C), Derived (D), or Multivalued (M).

Table C.4 shows a composite primary key based on LOG_DATE, LOG_TIME, and LOG_READER. The assumption is that it is impossible for any magnetic card reader to record the same time (to the second) for more than one entry because it takes a few seconds to complete the magnetic card swipe. If LOG_READER is not part of the primary key, it is possible that two different card readers swiped at the same time would record the same time and, thus, violate the entity integrity requirement.

The LOG's sample data are shown in Figure C.3.

FIGURE C.3 SAMPLE LOG DATA

LOG_DATE	LOG_TIME	LOG_READER	USER_ID
17-Jan-2014	2:24:32 PM	1	1-11-1138
17-Jan-2014	2:24:39 PM	2	1-11-1125
17-Jan-2014	2:25:41 PM	2	1-11-1123
17-Jan-2014	2:25:44 PM	1	1-11-1129
17-Jan-2014	2:25:58 PM	2	1-11-1143
17-Jan-2014	2:26:03 PM	1	1-11-1115
17-Jan-2014	2:26:28 PM	2	1-11-1112
17-Jan-2014	2:29:19 PM	1	1-11-1136
17-Jan-2014	2:30:35 PM	1	1-11-1114
17-Jan-2014	2:32:09 PM	1	1-11-1130

As you examine the LOG data in Figure C.3, it might occur to you that adding an attribute such as LOG_ID would eliminate the need for a composite primary key. You might also argue that such a LOG_ID attribute would be redundant because the combination of LOG_DATE, LOG_TIME, and LOG_READER already performs the primary key function. That's true enough. But the existence of a candidate key is not structurally damaging, and a single-attribute primary key decreases system overhead by diminishing the primary key index requirements. Here is yet another example of the many decisions that the database designer must make. (As you can tell, the decision was made to stick with the composite primary key.) Try to answer questions such as these: Is the attribute necessary or useful? If it is useful, what is the

cost of creating and using it? What function does it have that cannot be well served by other attributes? As you can see, database design requires the use of professional judgment.

The CLD manages a group of lab assistants (LAs). The University's policy is to limit the Lab staffing for daily operations to graduate assistants (GAs), student workers (SW), and work study students (WS). The GAs are limited to a 20-hour work week, the SWs are limited to a 10-hour work week, and the WSs are limited to a 4-hour work week. The LA attributes are summarized in Table C.5.

TABLE C.5

THE LAB_ASSISTANT ENTITY

ATTRIBUTE NAME	CONTENTS	ATTRIBUTE TYPE (*)	PRIMARY KEY (PK) AND/OR FOREIGN KEY (FK)	REFERENCES
LA_ID	Lab assistant identification		PK	
LA_NAME	Lab assistant name	C		
LA_PHONE	Lab assistant campus phone	C		
LA_SEMESTER	Most recent working semester	C		
LA_TYPE	Lab assistant classification: GA = Graduate assistant SW = Student worker WS = Work study student			
LA_HIRE_DATE	Date hired			

* The attribute type may be Composite (C), Derived (D), or Multivalued (M).

As you examine Table C.5, remember that information requirements often determine the degree to which composite entities are decomposed. For example, it is likely that the CLD will want to generate a phone list to simplify contacting LAs. Therefore, the decomposition of the LA_NAME into its component first name, last name, and initial is appropriate. On the other hand, it is unlikely that much will be gained by decomposing a phone number such as 4142345 into the 414 exchange number and its 2345 extension. Although information needs are generally better served by greater atomism, the needless proliferation of attributes increases complexity without generating appropriate return benefits. Similarly, the LA_SEMESTER is expressed by entries such as SPRING12 to indicate the most recent semester during which the LA was working. End-user reporting requirements indicate that little would be gained by decomposing that entry into the SPRING semester designation and the 12 year designation.

A few of the LAB_ASSISTANT records are shown in Figure C.4 to illustrate the data entries.

FIGURE C.4 SAMPLE LAB_ASSISTANT DATA

LA_ID	LA_LNAME	LA_FNAME	LA_INITIAL	LA_PHONE	LA_SEMESTER	LA_TYPE	LA_HIRE_DATE
1-11-2001	Jones	James	C	4123234	SPRING18	GA	07-Jan-2018
1-11-2002	Smith	Anne	G	4123245	SPRING18	SW	10-Jun-2018
1-11-2003	Hernandez	Maria	M	4123245	SPRING18	GA	07-Jan-2018
1-11-2004	Inum	Idong	J	4123234	SPRING18	WS	07-Jan-2018
1-11-2005	Jamerson	George	D	4126789	SPRING18	SW	13-Jul-2017
1-11-2006	Patterson	Herman	W	4127890	SPRING18	GA	07-Jan-2018
1-11-2007	Troyana	Emily	H	4121121	FALL17	SW	04-Jan-2018
1-11-2008	Evans	Peter	G	4123234	SPRING18	GA	07-Jan-2018
1-11-2009	Vann	Evangeline	D	4121121	SPRING18	SW	07-Jan-2018



Note

The LA_SEMESTER attribute enables the CLD to check whether an LA is available for assignment during the current semester. Some LAs work in the Lab one semester, perform cooperative duties the next semester, and return to their Lab assignment the following semester. Thus, a FALL17 entry would indicate that the LA's last work assignment was during the Fall of 2017. Because LAB_ASSISTANT records are purged (and archived) after the LA's graduation, termination, or resignation, an LA_SEMESTER designation other than the current semester's indicates the last semester during which the LA worked.

To keep track of the LA work schedules, the CLD keeps a scheduling sheet like the one shown in Table C.6. Table C.6 defines the attributes of the WORK_SCHEDULE entity, which is shown in Table C.7.

TABLE C.6

THE LAB ASSISTANT WORK-SCHEDULING SHEET

TIME SLOT	MONDAY	TUESDAY	WEDNESDAY	THURSDAY	FRIDAY	SATURDAY	SUNDAY
06–08	1. Jones (GA)	Thomas (GA)	Gabril (GA)	Evans (GA)	Hernando (GA)		
	2. Jamerson (WS)	Chung (SW)	Chung (SW)	Tabrin (GA)	Mustava (GA)		
	3. Hernando (GA)	Womack (SW)	Thomas (GA)	Jones (GA)	Tabrin (GA)		
	4. Vann (SW)	Dalton (SW)	Smith, C (SW)	Smith, C (SW)	Rommel (SW)		
08–10	1. Jones (GA)	Thomas (GA)	Gabril (GA)	Evans (GA)	Hernando (GA)		
	2. Hernandez (GA)	Porter (WS)	Chung (SW)	Tabrin (GA)	Mustava (GA)		
	3. Jamerson (WS)	Womack (SW)	Thomas (GA)	Dalton (SW)	Tabrin (GA)		
	4. Vann (SW)	Chung (SW)	Smith, C (SW)	Smith, C (SW)	Rommel (SW)		

TABLE C.6

THE LAB ASSISTANT WORK-SCHEDULING SHEET (CONTINUED)

TIME SLOT		MONDAY	TUESDAY	WEDNESDAY	THURSDAY	FRIDAY	SATURDAY	SUNDAY
10-12	1.	Hernandez (GA)	Jones (GA)	Gabril (GA)	Jones (GA)	Hernando (GA)		
	2.	Troyana (SW)	Porter (WS)	Troyana (SW)	Tabrin (GA)	Antony (SW)		
	3.	Jamerson (WS)	Willis (GA)	Willis (GA)	Antony (SW)	Tabrin (GA)		
	4.	Morris (SW)	Womack (SW)	Antony (SW)	Dalton (SW)	Rommel (SW)		
12-02	1.	Evans (GA)	Jones (GA)	Gabril (GA)	Jones (GA)	Kallen (GA)	Jones (GA)	Tabrin (GA)
	2.	Trayana (SW)	Vann (SW)	Troyana (SW)	Tabrin (GA)	Evans (GA)	Dalton (SW)	Mustava (GA)
	3.	Willis (GA)	Willis (GA)	Willis (GA)	Antony (SW)	Mustava (GA)		
	4.	Highlon (SW)	Womack (SW)	Antony (SW)	Smith, C (SW)	Rostav (SW)		
02-04	1.	Evans (GA)	Hernando (GA)	Kallen (GA)	Kallen (GA)	Kallen (GA)	Gabril (GA)	Tabrin (GA)
	2.	Inum (WS)	Vann (SW)	Troyana (SW)	Tabrin (GA)	Willis (GA)	Dalton (SW)	Mustava (GA)
	3.	Jones (GA)	Morris (SW)	Morris (SW)	Mustava (GA)	Mustava (GA)	Batey (SW)	Kadin (SW)
	4.	Highlon (SW)	Womack (SW)	Chung (SW)	Jones (WS)	Batey (WS)	Avery (SW)	
04-06	1.	Evans (GA)	Hernando (GA)	Kallen (GA)	Kallen (GA)	Kallen (GA)	Gabril (GA)	Sorals (GA)
	2.	Kadin (SW)	Vann (SW)	Sorals (GA)	Thomas (GA)	Willis (GA)	Dalton (SW)	Mustava (GA)
	3.	Winston (SW)	Morris (SW)	Morris (SW)	Jones, A (WS)	Jones, A (SW)	Batey (SW)	
	4.	Rostav (SW)	Avery (SW)	Avery (SW)	Highlon (SW)	Jones (GA)	Rommel (SW)	
06-08	1.	Evans (GA)	Hernando (GA)	Kallen (GA)	Kallen (GA)	Kallen (GA)	Gabril (GA)	Sorals (GA)
	2.	Kadin (SW)	Thomas (GA)	Sorals (GA)	Sorals (GA)	Willis (GA)	Aaron (SW)	Mustava (GA)
	3.	Winston (SW)	Avery (SW)	Thomas (GA)	Jones, A (WS)	Aaron (SW)	Rommel (SW)	
	4.	Rostav (SW)	Winston (SW)	Avery (SW)	Kadin (SW)	Batey (SW)		
08-10	1.	Evans (GA)	Hernando (GA)	Kallen (GA)	Sorals (GA)	Kallen (GA)	Gabril (GA)	Sorals (GA)
	2.	Kadin (SW)	Thomas (GA)	Sorals (GA)	Thomas (GA)	Willis (GA)	Aaron (SW)	Witte (SW)
	3.	Highlon (SW)	Avery (SW)	Thomas (GA)	Jones, A (WS)	Aaron (SW)	Winston (SW)	
	4.	Rostav (SW)	Winston (SW)	Winston (SW)	Rostav (SW)	Rommel (SW)		
10-12	1.	Casey (GA)	Casey (GA)	Casey (GA)	Casey (GA)	Casey (GA)	Gabril (GA)	Sorals (GA)
	2.	Thompson (SW)	Thompson (SW)	Thompson (SW)	Karpov (SW)	Karpov (SW)	Witte (SW)	Witte (SW)
	3.							
	4.							

TABLE C.7

THE WORK_SCHEDULE ENTITY

ATTRIBUTE NAME	CONTENTS	ATTRIBUTE TYPE (*)	PRIMARY KEY (PK) AND/OR FOREIGN KEY (FK)	REFERENCES
SCHED_SEMESTER	Semester ID	C	PK	
SCHED_WEEKDAY	Schedule weekday		PK	
SCHED_IN	Time slot start		PK	
SCHED_OUT	Time slot end			
SCHED_SLOT	Weekday slot number (Value range 1–4)		PK	
LA_ID	Lab assistant ID		FK	LAB_ASSISTANT

* The attribute type may be Composite (C), Derived (D), or Multivalued (M).

The WORK_SCHEDULE's primary key is a composite key, created by the combination of SCHED_SEMESTER, SCHED_WEEKDAY, SCHED_IN, and SCHED_SLOT. The requirement is that any given LA cannot have two of the same scheduled starting times for the same weekday. For example, an LA cannot have two starting times of 10 a.m. on Monday. The SCHED_IN and SCHED_OUT entries are based on a 24-hour time clock and range from 0600 to 2400.

A second requirement is that there can be no more than four lab assistants assigned to work during the same time slot. Unfortunately, the initial primary key selection doesn't fit those requirements well. Fortunately, the current design shortcomings can be fixed by adopting a three-pronged approach, as follows:

1. Create a primary key composed of SCHED_SEMESTER, SCHED_WEEKDAY, SCHED_IN, and SCHED_SLOT.
2. Create a unique index based on SCHED_SEMESTER, SCHED_WEEKDAY, SCHED_IN, and LA_ID.
3. Create a data validation rule to specify that the SCHED_SLOT values must be 1, 2, 3, or 4.

Option 1 can be implemented by declaring the PK components when the table is created. Option 2 can be implemented through the CREATE INDEX command. Option 3 requires the use of application code to enforce the validity of SCHED_SLOT values. If you are using Oracle, you can implement option 3 by using a trigger. (If you are using MS Access, you can use a data validation rule to implement option 3.) The implementation of options 1 and 3 will ensure that a maximum of four lab assistants will be working at any given scheduled time. Option 2 ensures that no lab assistant appears more than once in any given weekday/time combination.

Sample WORK_SCHEDULE data entries are shown in Figure C.5.

FIGURE C.5 SAMPLE WORK_SCHEDULE DATA

SCHED_SEMESTER	SCHED_WEEKDAY	SCHED_IN	SCHED_OUT	SCHED_SLOT	LA_ID
SPRING18	Friday	06:00	08:00		1 1-11-2003
SPRING18	Friday	08:00	10:00		1 1-11-2003
SPRING18	Friday	10:00	12:00		1 1-11-2003
SPRING18	Friday	14:00	16:00		2 1-11-2018
SPRING18	Friday	16:00	18:00		2 1-11-2018
SPRING18	Friday	16:00	18:00		4 1-11-2001
SPRING18	Friday	18:00	20:00		2 1-11-2018
SPRING18	Friday	20:00	22:00		2 1-11-2018
SPRING18	Monday	06:00	08:00		1 1-11-2001
SPRING18	Monday	06:00	08:00		2 1-11-2005
SPRING18	Monday	06:00	08:00		3 1-11-2003

Keep in mind that the order in which the WORK_SCHEDULE attributes are stored in the table is immaterial to the relational model. However, your DBMS is very likely to index the table according to its primary key. In this case, the indexing order begins with SCHED_SEMESTER and, within that order, moves to index SCHED_WEEKDAY, SCHED_IN, and SCHED_SLOT. The data displayed in Figure C.5 conform to such an indexing order. In that case, the primary key ensures an order that is independent of the LA_ID or the assistant's last name. Once again, you see a condition that is inconsequential to the relational model, but that eventually affects how the designer evaluates the design for implementation and applications development.

The HOURS_WORKED structure, shown in Table C.8, tracks the number of hours worked by each LA during a two-week pay period.

TABLE C.8
THE HOURS_WORKED ENTITY

ATTRIBUTE NAME	CONTENTS	ATTRIBUTE TYPE (*)	PRIMARY KEY (PK) AND/OR FOREIGN KEY (FK)	REFERENCES
LA_ID	Lab assistant ID		PK, FK	LAB-ASSISTANT
HW_SEMESTER	Semester designation	C	PK	
HW_DATE	Work period ending data		PK	
HW_HOURS_WORKED	Total hours worked			

* The attribute type may be Composite (C), Derived (D), or Multivalued (M).

The HOURS_WORKED entity's primary key is a composite key and consists of LA_ID, HW_SEMESTER, and HW_DATE. The inclusion of HW_DATE as part of the primary key is required to maintain entity integrity because the combination of LA_ID and HW_SEMESTER can produce many occurrences. (Each LA works many weeks within the semester.) Also note that the HW_HOURS_WORKED attribute represents the total hours worked by the LA during the pay period *and is entered manually by the end user*. (In this case, the HW_HOURS_WORKED attribute is *not* a derived attribute. Note that there are no attributes in this table from which the HW_HOURS_WORKED attribute

could be computed. If the HOURS_WORKED entity had included HW_TIME_IN and HW_TIME_OUT attributes, the HW_HOURS_WORKED attribute values could have been calculated from the other two time attributes and would, *in that case*, have been a derived attribute.) The HOURS_WORKED data form the basis for payroll applications. A few sample data entries are shown in Figure C.6.

FIGURE C.6 SAMPLE HOURS_WORKED DATA

LA_ID	HW_SEMESTER	HW_DATE	HW_HOURS_WORKED
1-11-2001	SPRING18	15-Jan-2018	40
1-11-2003	SPRING18	15-Jan-2018	40
1-11-2005	SPRING18	15-Jan-2018	8
1-11-2007	SPRING18	15-Jan-2018	20
1-11-2009	SPRING18	15-Jan-2018	20
1-11-2017	SPRING18	15-Jan-2018	20
1-11-2018	SPRING18	15-Jan-2018	40

Although the Lab is used mostly by students doing their assignments, sections of the Lab may be reserved by faculty members for teaching purposes or by staff members for hardware and software maintenance and updates. To enable the system to handle those reservations, the initial RESERVATION Entity structure was developed, as shown in Table C.9.

TABLE C.9

THE RESERVATION ENTITY

ATTRIBUTE NAME	CONTENTS	ATTRIBUTE TYPE (*)	PRIMARY KEY (PK) AND/OR FOREIGN KEY (FK)	REFERENCES
RES_DATE	Reservation date		PK	
USER_ID	User ID (faculty/staff only)		PK, FK	USER
RES_DATES_RESVD	Date(s) reserved	M	PK	
RES_TIME_IN	Time(s) in reserved	M		
RES_TIME_OUT	Time(s) out reserved	M		
RES_USERS	Number of users during the scheduled reserved time	M		
LA_ID	Lab assistant who entered the reservation		FK	LAB_ASSISTANT

* The attribute type may be Composite (C), Derived (D), or Multivalued (M).

As you examine the RESERVATION entity's structure, note its many multivalued attributes. For example, a faculty member may reserve the Lab for several dates and, within those dates, several times per day, each time for a different number of users. Such multivalued attributes are guaranteed to create problems at the implementation stage. For example, how many RES_DATES_RESVD derivative attributes (RES_DATES_RESVD1, RES_DATES_RESVD2, RES_DATES_RESVD3) should you reserve for storing the reservation dates? When you create too many, you have many nulls. When you create too few, reservations are limited by the available attributes. And if you want to allow

additional reservation dates later, you'll have to modify the table structure. The problem is magnified by the fact that, for each reserved date, there are many possible reserved times. The number of such derivative attributes can multiply dramatically. (The authors once did a database audit in which one of the tables contained 114 attributes—and the list was growing. Not surprisingly, the database did not function very well.)

The RESERVATION structure can be split into two tables in a 1:M relationship. The first of those two tables, still named RESERVATION, represents the “1” side. Its structure is shown in Table C.10.

TABLE C.10

THE REVISED RESERVATION ENTITY

ATTRIBUTE NAME	CONTENTS	ATTRIBUTE TYPE (*)	PRIMARY KEY (PK) AND/OR FOREIGN KEY (FK)	REFERENCES
RES_ID	Reservation ID		PK	
RES_DATE	Date on which the reservation was made			
USER_ID	User ID (faculty/staff only)		FK	USER
LA_ID	Lab assistant who entered the reservation		FK	LAB_ASSISTANT

* The attribute type may be Composite (C), Derived (D), or Multivalued (M).

Some sample RESERVATION data are shown in Figure C.7.

FIGURE C.7 SAMPLE RESERVATION DATA

RES_ID	RES_DATE	USER_ID	LA_ID
523	25-Jan-2018	1-11-1111	1-11-2003
524	25-Jan-2018	1-11-1112	1-11-2003
525	26-Jan-2018	1-11-1120	1-11-2008

This new RESERVATION structure works much better. Each time an LA records a set of reservations, the date on which the reservations are made is recorded in RES_DATE. You can also track who (USER_ID) made the reservation and who (LA_ID) recorded it. The multiple occurrences of the reservations are then handled by the “Many” side in a table named RES_SLOT, whose structure is shown in Table C.11.

TABLE C.11

THE RES_SLOT (WEAK) ENTITY

ATTRIBUTE NAME	CONTENTS	ATTRIBUTE TYPE (*)	PRIMARY KEY (PK) AND/OR FOREIGN KEY (FK)	REFERENCES
RES_ID	Reservation ID		PK, FK	RESERVATION
RSLOT_DATE	Date reserved		PK	
RSLOT_TIME_IN	Reservation time in		PK	
RSLOT_TIME_OUT	Reservation time out			
RSLOT_USERS	Number of users during the scheduled reserved time			
RSLOT_LAB_SECTION	Reserved section of the lab		PK	

* The attribute type may be Composite (C), Derived (D), or Multivalued (M).

As you examine the structure in Table C.11, note that the participation of the RSLOT_LAB_SECTION makes it possible to have two reservations on the same date and time when the reservations involve different sections of the Lab. Also note that RES_SLOT is a weak entity because it is existence-dependent on RESERVATION and because one of its primary key components, RES_ID, is inherited from the RESERVATION entity.

Figure C.8 shows some sample data to illustrate the reservation process.

FIGURE C.8 SAMPLE RES_SLOT DATA

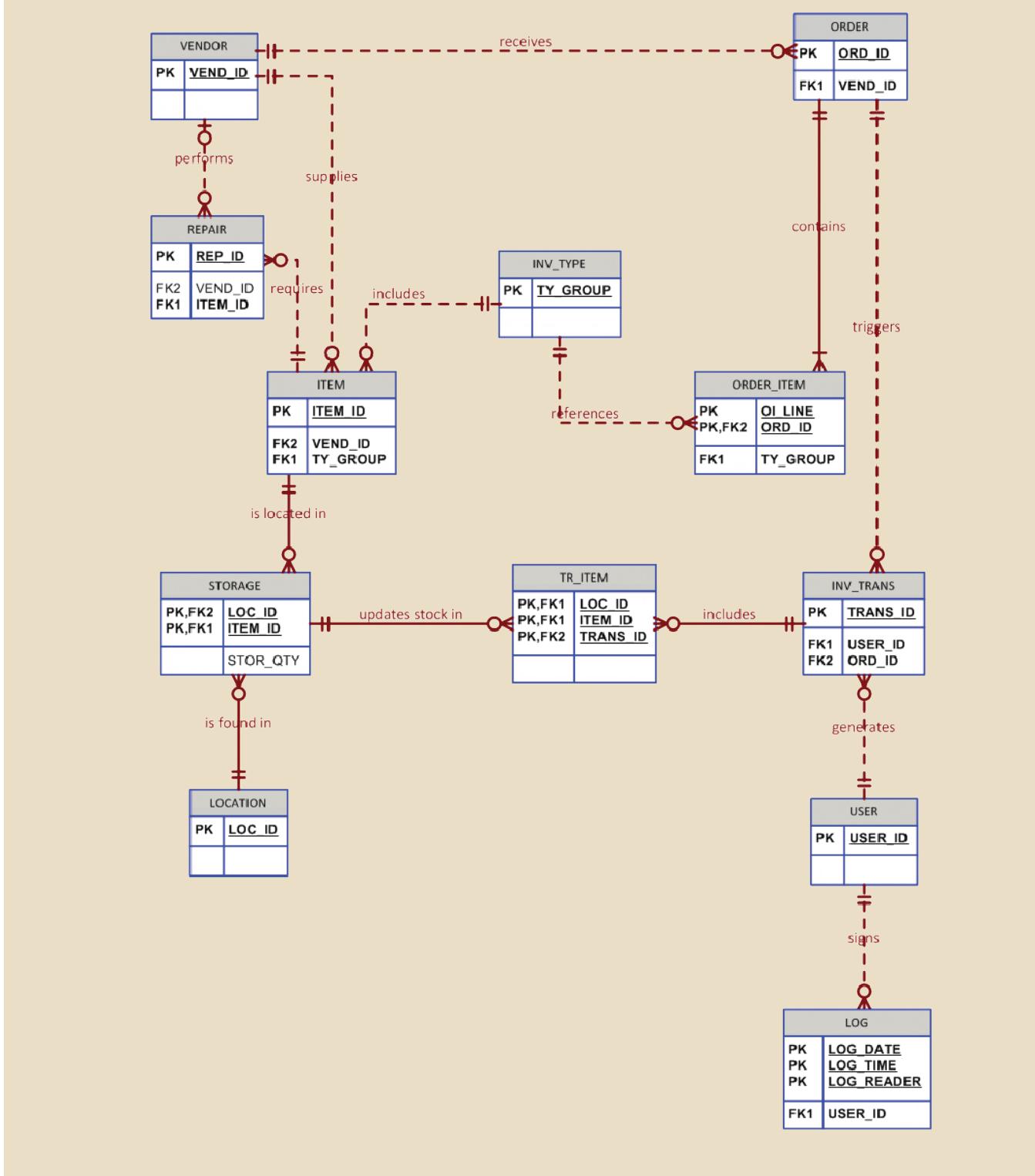
RES_ID	RSLOT_DATE	RSLOT_TIME_IN	RSLOT_SECTION	RSLOT_TIME_OUT	RSLOT_USERS
523	03-Feb-2018	8:00 AM	A	9:50 AM	23
523	10-Feb-2018	8:00 AM	A	9:50 AM	23
524	04-Feb-2018	2:00 PM	C	3:15 PM	35
525	03-Feb-2018	6:00 PM	A	8:40 PM	18
525	07-Feb-2018	10:00 AM	A	10:50 AM	24
525	10-Feb-2018	6:00 PM	B	8:40 PM	18

By examining the sample data in Figure C.8, you can easily trace the reservation process when you keep in mind the 1:M relationship between RESERVATION and RES_SLOT. Note, for example, that on January 25, 2018 (see the RESERVATION data in Figure C.7), user 1-11-1111 made reservations (see the RES_SLOT data in Figure C.8) for 23 users for February 3, 2018 from 8:00 a.m.–9:50 a.m. in Section A of the Lab and for 23 users for February 10, 2018 from 8:00 a.m.–9:50 a.m. in Section A of the Lab.

C-2b The Inventory Management Module

To help track the Inventory Management System's detailed development process, it is useful to look at its ER components, shown in Figure C.9. Using that illustration as your guide, you will find it much easier to understand the revision process. Refer to Figure C.9 often as the Inventory Management System's entities and their attributes are developed.

FIGURE C.9 THE INVENTORY MANAGEMENT MODULE'S ER SEGMENT



As you examine the ER segment in Figure C.9, you might wonder why the WITHDRAW and CHECK_OUT entities used in the initial ER diagram in Appendix B and in Table C.2 in this appendix do not appear. You will also see that a few new entities, such as INV_TRANS, have been added. Those changes are part of the ER data model verification process, and they will be discussed in detail later in this section. Also, the USER entity, which is not an explicit part of the Inventory Management System and was already discussed in Section C-2a, is the “connector” between the two entity segments. Therefore, although it will not be discussed further, the inclusion of USER makes sense in this segment, too.

The INV_TYPE entity performs an important role in the Inventory Management module. Its presence makes it easy for the CLD to generate detailed inventory summaries. (For example, how many boxes of 8.5" × 11" single-sheet paper are in stock? How many laser printers are available? How many boxes of writable CDs are on hand?) To understand the INV_TYPE’s function, you must first understand the role of a classification hierarchy, as shown in Table C.12.

TABLE C.12

AN INVENTORY CLASSIFICATION HIERARCHY

GROUP	CATEGORY	CLASS	TYPE	SUBTYPE
HWPCDTP3	Hardware (HW)	Personal computer (PC)	Desktop (DT)	Intel Core i7
WPCLTP4	Hardware (HW)	Personal computer (PC)	Laptop (LT)	Intel Core i7
WPCLTCE	Hardware (HW)	Personal computer (PC)	Laptop (LT)	Intel Core i5
WPRLSBL	Hardware (HW)	Printer (PR)	Laser (LS)	Black (BL)
WPRIJCO	Hardware (HW)	Printer (PR)	Ink-jet (IJ)	Color (CO)
SUPPSS11	Supply (SU)	Paper (PP)	Single-sheet (SS)	11-inch
HWEXVIXX	Hardware (HW)	Expansion board (EX)	Video (VI)	XX
SWDBXXXX	Software (SW)	Database (DB)	XX	XX

As you examine the classification hierarchy in Table C.12, note that three categories have been created: hardware, software, and supply. Also note that each group code precisely describes the inventory type discussed here. For example, the first group code, HWPCDTP3, describes the category “hardware” (HW), the class “personal computer” (PC), the type “desktop” (DT), and the subtype Intel Core i7. Some group codes, such as the last one in Table C.12, do not specifically identify type and subtype; rather, they use the code XX to indicate that no specification was made. You will see later that the classification hierarchy group codes can be used as primary keys to define the INV_TYPE rows and that the category, class, type, and subtype components will be stored as separate attributes to enhance the inventory reporting capabilities.

The classification hierarchy can also be presented as a tree diagram, as shown in Figure C.10.

The classification hierarchy illustrated in Table C.12 and Figure C.10 is reflected in the INV_TYPE structure shown in Table C.13 and in the INV_TYPE’s sample data illustrated in Figure C.11.

FIGURE C.10 THE INV_TYPE CLASSIFICATION HIERARCHY AS A TREE DIAGRAM

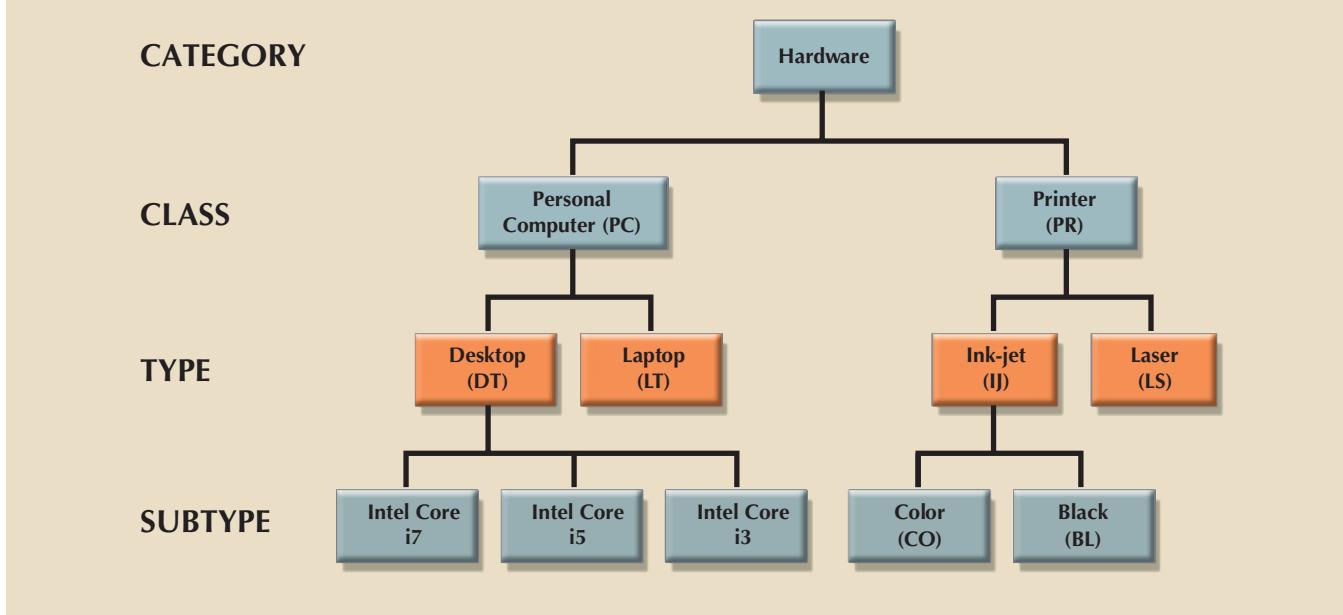


TABLE C.13

THE INV_TYPE ENTITY

ATTRIBUTE NAME	CONTENTS	ATTRIBUTE TYPE (*)	PRIMARY KEY (PK) AND/OR FOREIGN KEY (FK)	REFERENCES
TY_GROUP	Inventory group code	C	PK	
TY_CATEGORY	Inventory category			
TY_CLASS	Inventory class			
TY_TYPE	Inventory type			
TY_SUBTYPE	Inventory subtype			
TY_DESCRIPTION	Group description			
TY_UNIT	Unit of measurement (box, ream, and so on)			

*The attribute type may be Composite (C), Derived (D), or Multivalued (M).

As you examine the INV_TYPE structure in Table C.13, note that the INV_TYPE uses a single-attribute primary key (TY_GROUP), which renders the system faster and more efficient in the query mode. Although a composite primary key could have been created by combining TY_CATEGORY, TY_CLASS, TY_TYPE, and TY_SUBTYPE, such a multiple-attribute primary key would produce a more complex pointer system for the DBMS, thus slowing down the system. Yet the decomposition of TY_GROUP into TY_CATEGORY, TY_CLASS, TY_TYPE, and TY_SUBTYPE allows a greater variety of reporting summaries to be performed easily while having the benefit of a single-attribute primary key. (Remember that information requirements help drive the design process. Table C.13 and its sample data in Figure C.11 provide an appropriate illustration of that point.)

Although the INV_TYPE provides much flexibility in terms of inventory summary statements, you still must be able to reference specific units. For example, it is useful to know that there are 217 computers in inventory, but you must also be able to track each computer that was installed in a professor's office or in the Lab. The relationship between INV_TYPE and ITEM (review the ER segment in Figure C.9) provides that capability. The ITEM's entity structure is shown in Table C.14.

FIGURE C.11 SAMPLE INV_TYPE DATA

TY_GROUP	TY_CATEGORY	TY_CLASS	TY_TYPE	TY_SUBTYPE	TY_DESCRIPTION	TY_UNIT
HWBAPNXX	HW	BA	PN	XX	Bar Code Reader, Pen Type	UN
HWCAMO12	HW	CA	MO	12	External Modem Wire, 12'	UN
HWCASP08	HW	CA	SP	08	Serial Printer Cable, 8-pin	UN
HWEXHDID	HW	EX	HD	ID	Expansion Board-IDE HD ctrl.	UN
HWEXHDMF	HW	EX	HD	MF	Expansion Board-MFM HD ctrl.	UN
HWEXMEXX	HW	EX	ME	XX	Expansion Board, Memory	UN
HWEXVIXX	HW	EX	VI	XX	Expansion Board, Video	UN
HWMSXXXX	HW	MS	XX	XX	Hardware, Miscellaneous	UN
HWNCETCX	HW	NC	ET	CX	Ethernet NIC, Coax	UN
HWNCETTP	HW	NC	ET	TP	Ethernet NIC, Twisted Pair	UN
HWNCTR4M	HW	NC	TR	4M	Token Ring, NIC 4M	UN
HWPCDT48	HW	PC	DT	48	Tier 2 desktop	UN
HWPCDTCE	HW	PC	DT	CE	Tier 3 desktop computer	UN
HWPCDTM2	HW	PC	DT	M2	Tier 1 Apple computer	UN
HWPCDTP2	HW	PC	DT	P2	Desktop PC, tier 3	UN
HWPCDTP3	HW	PC	DT	P3	Desktop PC, tier 2	UN
HWPCDTP4	HW	PC	DT	P4	Desktop PC, tier 1	UN

TABLE C.14

THE ITEM ENTITY

ATTRIBUTE NAME	CONTENTS	ATTRIBUTE TYPE (*)	PRIMARY KEY (PK) AND/OR FOREIGN KEY (FK)	REFERENCES
ITEM_ID	Item identification code		PK	
TY_GROUP	Inventory group code		FK	INV_TYPE
ITEM_UNIV_ID	University inventory ID			
ITEM_SERIAL_NUM	Item (manufacturer's) serial number			
ITEM_DESCRIPTION	Item description			
ITEM_QTY	Total quantity on hand at all locations	D		
VEND_ID	Original vendor code		FK	VENDOR
ITEM_STATUS	Item status: 1 = available 2 = under repair 3 = out of order 4 = checked out			
ITEM_BUY_DATE	Purchase date			

* The attribute type may be Composite (C), Derived (D), or Multivalued (M).

Let's examine the behavior of the ITEM entity's attributes according to the three main inventory types:

- *Hardware.* If 20 computers are bought, each one will be assigned an ITEM_ID, an ITEM_UNIV_ID, and a manufacturer's ITEM_SERIAL_NUM, thus generating 20 records.
- *Supply.* If 20 boxes of laser printer paper are bought, only one record is generated because there is no need to identify each box individually. Therefore, in the case of the boxes, no university ID number is required, and the ITEM_UNIV_ID will be null. In addition, a box of paper would not have a serial number, so the ITEM_SERIAL_NUM will be null. However, because the CLD must be able to track the boxes, it is necessary to have ITEM_ID as the primary key. Naturally, you may use special codes, such as 00000, for the ITEM_UNIV_ID and the ITEM_SERIAL_NUM to avoid the use of nulls.
- *Software.* If a license is bought for 180 software copies, only one license record will exist in the ITEM entity. Individual installations can be tracked through the use of the STORAGE entity, shown in Table C.15.

TABLE C.15

THE STORAGE ENTITY

ATTRIBUTE NAME	CONTENTS	ATTRIBUTE TYPE (*)	PRIMARY KEY (PK) AND/OR FOREIGN KEY (FK)	REFERENCES
LOC_ID	Location ID		PK, FK	LOCATION
ITEM_ID	Item ID		PK, FK	ITEM
STOR_QTY	Quantity stored at this location			

*The attribute type may be Composite (C), Derived (D), or Multivalued (M).

Note also that the ITEM_QTY in Table C.14 is a derived attribute because it sums the quantity on hand *at all locations*. Keep in mind that design purity would dictate the elimination of a derived attribute. Yet its presence here reflects the end user's desire for simple and quick answers to such ad hoc questions as "How many boxes of 8.5" × 11" paper do we have at all locations?" Because the derived attribute ITEM_QTY is computed and written into the ITEM table by the application software any time there is a transaction involving the inventory, there is no chance of creating data anomalies through its inclusion.

Some sample ITEM data are shown in Figure C.12.

FIGURE C.12 SAMPLE ITEM DATA

ITEM_ID	TY_GROUP	ITEM_UNIV_ID	ITEM_SERIAL_NUM	ITEM_QTY	VEND_ID	ITEM_STATUS	ITEM_BUY_DATE
1212004	HWEXMEXX	00000	00000	2	PCJUN	2	10-May-2017
2088723	HWPCLTP3	3009765	CX-5437688	1	PCJUN	4	12-May-2017
2879954	SUPPCO11	00000	00000	84	PCPAL	1	12-May-2017
2998950	SWDBXXXX	00000	00000	20	COMPU	1	15-May-2017
3045887	HWEXVIXX	3422012	BBF-985643	1	COMPU	1	15-May-2017
3154567	SUPPSS11	00000	00000	121	PCPAL	1	15-May-2017
3210946	HWEXHDID	3215457	12Q31223D9	1	PCJUN	1	16-May-2017
3212345	HWNCTECX	3245367	TR/3255675	1	PCJUN	1	16-May-2017

It is necessary to be able to locate an item in inventory at any given time. Therefore, the item's storage location must be known. The STORAGE entity, shown in Table C.15, plays an important role.

The STORAGE sample data are shown in Figure C.13.

FIGURE C.13 SAMPLE STORAGE DATA

LOC_ID	ITEM_ID	STOR_QTY
KOM106-1	3154567	19
KOM106-1	4238130	15
KOM106-1	4238132	10
KOM203E-1	2088723	1
KOM203E-1	3212345	1
KOM203E-1	4456789	1
KOM203E-1	4456791	1
KOM203E-2	4562397	1
KOM205B-1	2879954	10
KOM205B-1	3154567	25
KOM245A-1	2879954	35
KOM245A-1	4238131	18
KOM245A-1	4238132	10
KOM245A-1	4451236	1
KOM245A-2	3154567	52
KOM245A-2	4009212	1
KOM245A-2	4112151	1
KOM245A-2	4238132	12
KOM245B-1	3154567	5
KOM245B-1	4228753	1
KOM245B-1	4358255	1
KOM245B-1	4358258	1
KOM245B-1	4451235	1
KOM245B-2	3154567	8

By tracing the ITEM_ID in STORAGE (see Figure C.13) to the ITEM_ID in ITEM (see Figure C.12) and then to TY_GROUP = SUPPSS11 in INV_TYPE (see Figure C.11), you can determine that the first record in STORAGE shows 19 boxes of single-sheet paper (ITEM_ID = 3154567) located in KOM106-1. An additional 25 boxes of single-sheet paper are located in KOM205B-1, 52 boxes are located in KOM245A-2, and 5 boxes are located in KOM245B-1.

The storage location details are stored in the LOCATION entity, shown in Table C.16. By reviewing the ER segment in Figure C.9, you will note that there is a 1:M relationship between LOCATION and STORAGE.

TABLE C.16

THE LOCATION ENTITY

ATTRIBUTE NAME	CONTENTS	ATTRIBUTE TYPE (*)	PRIMARY KEY (PK) AND/OR FOREIGN KEY (FK)	REFERENCES
LOC_ID	Location ID		PK	
LOC_DESCRIPTION	Location description (examples: faculty office, classroom, cabinet, and so on)			

* The attribute type may be Composite (C), Derived (D), or Multivalued (M).

The LOCATION entity's sample data are shown in Figure C.14.

FIGURE C.14 SAMPLE LOCATION DATA

LOC_ID	LOC_DESCRIPTION
KOM106-1	CIS Department office
KOM106-2	CIS Department office
KOM200-1	Classroom
KOM200-2	Classroom
KOM200-3	Classroom
KOM200-4	Classroom
KOM203E-1	Faculty office
KOM203E-2	Faculty office
KOM205A-1	Hall storage closet
KOM205A-2	Hall storage closet
KOM205A-3	Hall storage closet
KOM205A-4	Hall storage closet
KOM205B-1	Hall storage closet
KOM205B-2	Hall storage closet
KOM245A-1	Computer lab, section A
KOM245A-2	Computer lab, section A
KOM245A-3	Computer lab, section A
KOM245A-4	Computer lab, section A
KOM245A-5	Computer lab, section A
KOM245B-1	Computer lab, section B
KOM245B-2	Computer lab, section B
KOM245B-3	Computer lab, section B

You can create as much detail as necessary in the LOCATION entity's LOC_DESCRIPTION. For example, you can specify bins, shelves, and other details within the storage location. In fact, if you were to use that design technique in a store or plant location, you might create a series of new attributes to specify section, aisle, shelf, and bin.

At this point, you are able to:

- Provide detailed descriptions of the items in the UCL's inventory.
- Provide inventory category summaries easily and efficiently.
- Determine the item status.
- Trace the items to their storage locations.

Because inventory tracking is very important, especially at auditing time, the emerging system is already showing considerable end-user potential. The remaining Inventory Management System will be built on that solid foundation.

Items in inventory are dynamic; that is, the items don't stay in inventory forever. In fact, some items, such as supplies, have a very limited inventory life. Paper, for example, doesn't last long in a computer lab. Software becomes obsolete, as does hardware. Hardware might break down and require repair, or it might require disposal. In fact, the need for repair produces some special database handling. An item sent out for repair still belongs to the Lab, but it is not available for use. Neither is an item that has broken down but has not yet been sent out for repair. Some items can be repaired in-house, and some items are returned to the vendor for replacement. In short, repair is an ever-present issue that requires tracking. Therefore, the REPAIR entity shown in Table C.17 plays an important role in the design.

TABLE C.17

THE REPAIR ENTITY

ATTRIBUTE NAME	CONTENTS	ATTRIBUTE TYPE (*)	PRIMARY KEY (PK) AND/OR FOREIGN KEY (FK)	REFERENCES
REP_ID	Repair ID		PK	
ITEM_ID	Item identification code		FK	ITEM
REP_DATE	Date on which item needed repair			
REP_DESCRPT	Problem description			
REP_STATUS	Repair status: 1 = in repair 2 = repaired 3 = returned to vendor 4 = out of order			
VEND_ID	Vendor code		FK	VENDOR
REP_REF	Reference number supplied by vendor			
REP_DATE_OUT	Date on which item was sent out			
REP_DATE_IN	Date on which item was returned			
REP_COST	Repair cost			

* The attribute type may be Composite (C), Derived (D), or Multivalued (M).

Several of the REPAIR attributes in Table C.17 require additional explanation.

- VEND_ID is an optional foreign key because the VENDOR does not enter the repair picture until the REP_STATUS = 3. If the repair status is 3, the VEND_ID must contain a valid VEND_ID entry—that is, one that matches a vendor in the VENDOR table. If you want to avoid nulls by using a “no vendor” code, the VENDOR table that the code references must contain a “no vendor” entry to maintain referential integrity.
- Because the cost is not known until the repair has been completed, REP_COST will be \$0.00 until REP_STATUS has been changed to 2. Some repairs are done at the vendor’s expense, so it is quite possible that the REP_COST will remain \$0.00 when the REP_STATUS = 2. Also, the many repairs done in-house do not carry a charge, except the cost of replacement parts.
- REP_DESCRPT cannot be null; there must be some description of the problem that occurred.

A few REPAIR records are shown in Figure C.15.

FIGURE C.15 SAMPLE REPAIR DATA

REP_ID	ITEM_ID	REP_DATE	REP_DESCRPT	REP_STATUS	VEND_ID	REP_REF	REP_DATE_OUT	REP_DATE_IN	REP_COST
121	3045887	15-Jan-2018	Memory board failure	2	COMPU	RT-54566-674	17-Jan-2016	20-Jan-2018	0.00
122	5453234	17-Jan-2018	Token ring board failure	2	PCJUN	231156	17-Jan-2016	22-Jan-2018	54.82
123	3045887	05-Jan-2018	Video board failure	1	COMPU	RT-57455-121	07-Jan-2016		0.00
124	3244536	01-Jan-2018	Power supply short/burn	3	PCPAL	FRG-324458	09-Jan-2016		0.00

To place orders, to return equipment for repair, and so on, the system must contain VENDOR data. Table C.18 shows a simplified structure for the VENDOR entity.

As you examine the VENDOR entity's structure, keep the following points in mind:

TABLE C.18

THE VENDOR ENTITY

ATTRIBUTE NAME	CONTENTS	ATTRIBUTE TYPE (*)	PRIMARY KEY (PK) AND/OR FOREIGN KEY (FK)	REFERENCES
VEND_ID	Vendor identification code		PK	
VEND_NAME	Vendor name	C		
VEND_ADDRESS	Vendor street address			
VEND_CITY	Vendor city			
VEND_STATE	Vendor state			
VEND_ZIP	Vendor zip code			
VEND_PHONE	Vendor phone	C		
VEND_CONTACT	Vendor contact person	C		
VEND_CON_PHONE	Vendor contact phone	C		
VEND_TECH_PHONE	Vendor tech support phone	C		

* The attribute type may be Composite (C), Derived (D), or Multivalued (M).

- Because the system must be able to generate shipping labels for items returned to the vendor, it is necessary to decompose the vendor address into street address, city, state, and zip code.
- There was no end-user requirement for a vendor telephone number to be broken down by area code, nor was there a requirement for an alphabetically arranged list of contacts and technical support people. Therefore, the last four VENDOR attributes were left as composites.

A few sample VENDOR records are shown in Figure C.16.

FIGURE C.16 SAMPLE VENDOR DATA

VEND_ID	VEND_NAME	VEND_ADDRESS	VEND_CITY	VEND_STATE	VEND_ZIP	VEND_PHONE	VEND_CONTACT	VEND_CON_PHONE	VEND_TECH_PHONE
COMPU	ComputerLand	1012 Hard Drive,	San Francisco	CA	12345	8004567789	John E. Miesler	8004567785	8004567762
PCJUN	PC Junction	1234 Video Circle	Nashville	TN	23456	6153454567	Anna D. Williamson	6153454574	6153454573
PCPAL	PC Palace	4567 Ram Lane	New York	NY	34567	8002341234	Juan H. Cordova	8002341232	8002341235

Given the Lab's frequent hardware, software, and supply updates, the system's ORDER entity plays a crucial role. Its necessary attributes, required to satisfy budgeting, auditing, and various system end-user requirements, are reflected in the ORDER entity in Table C.19.

TABLE C.19

THE ORDER ENTITY

ATTRIBUTE NAME	CONTENTS	ATTRIBUTE TYPE (*)	PRIMARY KEY (PK) AND/OR FOREIGN KEY (FK)	REFERENCES
ORD_ID	Order ID code		PK	
ORD_DATE	Order date			
VEND_ID	Vendor ID code		FK	VENDOR
ORD_VEND_REF	Reference number supplied by the vendor (optional)			
ORD_PO_NUM	Purchase order number			
ORD_TOT_COST	Total order cost, including shipping and handling			
ORD_STATUS	Order status: OPEN = open order REC = received CANCEL = canceled order PAID = paid order			
ORD_FUND_TYPE	Order funding source: BUS = College of Business budget			
USER_ID	Person who requested the order		FK	USER

* The attribute type may be Composite (C), Derived (D), or Multivalued (M).

Appendix B stated that each order contains one or many ordered items. Using the normalization rules, the order was split into two entities: ORDER and ORDER_ITEM. The ORDER entity is related in a one-to-many relationship with the ORDER_ITEM entity. (Review Figure C.9 to see the precise relationship between ORDER and ORDER_ITEM.) The ORDER entity (the “1” side) contains general data about the order; the ORDER_ITEM entity (the “M” side) contains the items in the order.

The ORDER entity’s ORD_STATUS reflects reporting requirements. Clearly, it is possible for an order to have been received and not yet paid for, so a distinction must be made between REC and PAID. Although canceled orders have no impact on inventory movements, it is important to keep track of them. For example, before you make the payment on a bill, it would be wise to find out if the order has been canceled. The ORD_FUND_TYPE lets you know to which budget you should charge the payments. And because accountability is an ever-present factor, you must be able to track (through USER_ID) the person who originated the order.

Information requirements might also determine on which side of the 1:M relationship the data is stored. For example, it is quite possible for only part of an order to arrive at a given time. Let’s say that the order consisted of 12 computers and 25 boxes of paper. All 25 boxes of paper but only 8 computers might have arrived. Because you must trace those portions of the order that are complete, the receipt date must be stored on the “M” side of the 1:M relationship. Using a similar approach, you must decide where to store the USER_ID attribute. For example, if information requirements demand that you get a precise listing of who ordered what specific item in any one order, the USER_ID must be stored on the “M” side. On the other hand, if you merely need to know who placed the

entire order, the USER_ID is stored on the “1” side. Given the latter scenario, you may then list the person who requested the specific item within an order as part of each order line description.

The placement of the VEND_ID depends on a simple business rule. In this case, it is quite reasonable to assume that each order is placed with a single vendor. (Just think how many checks you would have to write if each order line were tied to a different vendor within the same order.) Therefore, the VEND_ID is written on the “1” side of the relationship between the order and its order lines. The revised ORDER structure, shown in Table C.19, shows what decisions were made in that design.

To illustrate the data placement, some ORDER sample data are shown in Figure C.17.

FIGURE C.17 SAMPLE ORDER DATA

ORD_ID	ORD_DATE	VEND_ID	ORD_VEND_REF	ORD_PO_NUM	ORD_TOT_COST	ORD_STATUS	ORD_FUND_TYPE	USER_ID
121	06-Feb-2018	PCJUN	320021	21234	17335.13	OPEN	CIS	1-11-1117
122	06-Feb-2018	PCJUN	320213	21234	10698.20	PAID	CIS	1-11-1128
123	10-Feb-2018	COMPU	320322	21234	1751.95	PAID	CIS	1-11-1120
124	10-Feb-2018	PCPAL	000000	21234	2397.74	REC	BUS	1-11-1113
125	14-Feb-2018	COMPU	320345	21234	640.98	REC	CIS	1-11-1128

The “M” side of the relationship between orders and their components will be stored in the ORDER_ITEM table. Thus, each ORDER references one or more ORDER_ITEM records, but each ORDER_ITEM entry refers to a single ORDER record. The ORDER_ITEM’s structure is shown in Table C.20.

TABLE C.20

THE ORDER_ITEM ENTITY

ATTRIBUTE NAME	CONTENTS	ATTRIBUTE TYPE (*)	PRIMARY KEY (PK) AND/OR FOREIGN KEY (FK)	REFERENCES
ORD_ID	Order number		PK, FK	ORDER
OI_LINE	Order line number		PK	
TY_GROUP	Inventory group		FK	INV_TYPE
OI_DESCRIPTION	Item description			
OI_UNIT_COST	Unit cost			
OI_QTY_ORD	Order quantity			
OI_COST	Total line cost	D		
OI_QTY_RECVD	Quantity received			
OI_DATE_IN	Last date received			

* The attribute type may be Composite (C), Derived (D), or Multivalued (M).

Using two entities, ORDER and ORDER_ITEM, results in some additional advantages, as follows:

- The ORDER_ITEM entity will contain an OI_LINE attribute to represent the order in which the INV_TYPE (items) are actually entered. If ORD_ID and TY_GROUP are used instead, any change will produce a different listing each time an item is added to

an order. In that case, if an end user were to review a previous order, it might have an arrangement quite different from that of the original entries and would likely confuse the end user. By keeping the OI_LINE attribute in ORDER_ITEM, that problem is eliminated.

- The OI_DESCRIPTION has a special purpose. Although application software can be used to read the description found in INV_TYPE when the end user enters the TY_GROUP code, that description can still be modified to clarify the order details. (See the sample data in Figure C.18) Because this description does not serve the purpose of a foreign key, the redundancy does not create structural problems. Additionally, the descriptive material can prove to be helpful if questions arise later about the precise nature of the order.

The sample data for the ORDER_ITEM are shown in Figure C.18.

FIGURE C.18 SAMPLE ORDER_ITEM DATA

ORD_ID	OI_LINE	TY_GROUP	OI_DESCRIPTION	OI_UNIT_COST	OI_QTY_ORD	OI_COST	OI_QTY_RECVD	OI_DATE_IN
121	1	HWNCTR4M	Ethernet card, NIC (BAS112B addition)	184.23	1	184.23	1	10-Feb-2018
121	2	HWPCTP3	DELL Desktop, 800GB HD, 160GB RAM	2395.00	5	11975.00	2	
121	3	HWPCLTP3	DELL Inspiron, 200GB HD, 4GB RAM, 14" display	2587.95	2	5175.90	2	10-Feb-2018
122	1	HWPRLSBL	HP Laserjet 4550, 64MB RAM	1999.99	5	9999.95	5	11-Feb-2018
122	2	SUPPSS11	Laser printer paper, 5,000 sheet box	19.95	35	698.25	35	11-Feb-2018

As you examine the ORDER and ORDER_ITEM data presented in Figure C.17 and Figure C.18, respectively, you can easily trace all orders and their components. For example, as you look at ORDER's ORD_ID = 121 and trace it through ORDER_ITEM's ORD_ID = 121, you can draw the following conclusions:

- The order consisted of three items: one network card, five Dell desktop computers, and two Dell laptop computers. (Note that the ORDER_ITEM's OI_LINE numbers range from 1 to 3 for ORD_ID = 121.)
- The order was written on February 10, 2018 to vendor PCJUN.
- The order is still open because the ORDER_ITEM's second order line (OI_LINE = 2) shows that only two of the five computers have been received. (Note that the ORDER_ITEM's OI_DATE_IN is null.)

Keeping track of the items in inventory is a challenge in the UCL's environment because there are two different types of transactions. Some items, such as paper, ink-jet cartridges, and other consumables, are withdrawn from inventory as they are used. For example, if a faculty member needs a box of paper for the classroom or the office, the stock of "boxes of paper" is simply decreased by one box. However, faculty and staff might also check out items *temporarily*, such as flat panel displays for use in the classroom or laptop computers to take to remote classes for demonstration purposes. In that case, the checked-out item remains in inventory, but its availability status and location change. You must be able to track each item's user and location. When the item is returned, another (check-in) transaction changes the availability status and location again.

To examine the inventory transactions, let's begin with some simple withdrawals. The following scenario covers four transactions, recorded as withdrawals 325, 326, 327, and 328. (You can evaluate the transaction components by reviewing the previously shown sample data. For example, you know that user 1-11-1111 is a CIS faculty member from examining the USER table. You can find item 4238131 in the ITEM table; you determine that this item is a laser printer cartridge by looking at the INV_TYPE table, and so on.)

- Transaction 325: CIS faculty member 1-11-1111 withdrew a laser printer cartridge (ITEM_ID = 4238131) from the computer lab (KOM245A-1) on February 4, 2018.
- Transaction 326: CIS staff member 1-11-1128 withdrew three laser printer cartridges from KOM245A-1, five boxes of single-sheet 8.5" × 11" paper (ITEM_ID = 3154567) from KOM245B-1, and two boxes of 3.5" disks (ITEM_ID = 4238132) from KOM245A-1 on February 4, 2018.
- Transaction 327: CIS staff member 1-11-1130 withdrew one box of 3.5" floppy disks from KOM106-1 on February 7, 2018.
- Transaction 328: CIS faculty member 1-11-1109 withdrew one box of single-sheet 8.5" × 11" paper from KOM106-1 and one black ink-jet cartridge (ITEM_ID = 4238130) from KOM106-1 on February 8, 2018.

Before those transactions can be tracked, there must be a set of database tables to support them. At this point, you cannot track the transactions because the database reflects withdrawals as the M:N relationship between USER and ITEM shown in Figure C.19, Panel A. For example, a qualified user can withdraw many boxes of paper, and a box of paper in inventory can be withdrawn by any number of qualified users.

Because the M:N relationship cannot be properly implemented in a relational database design, your first thought might be to transform the “withdraws” relationship into a composite entity named WITHDRAW, shown in Figure C.19, Panel B. That entity's structure is illustrated in Table C.21.

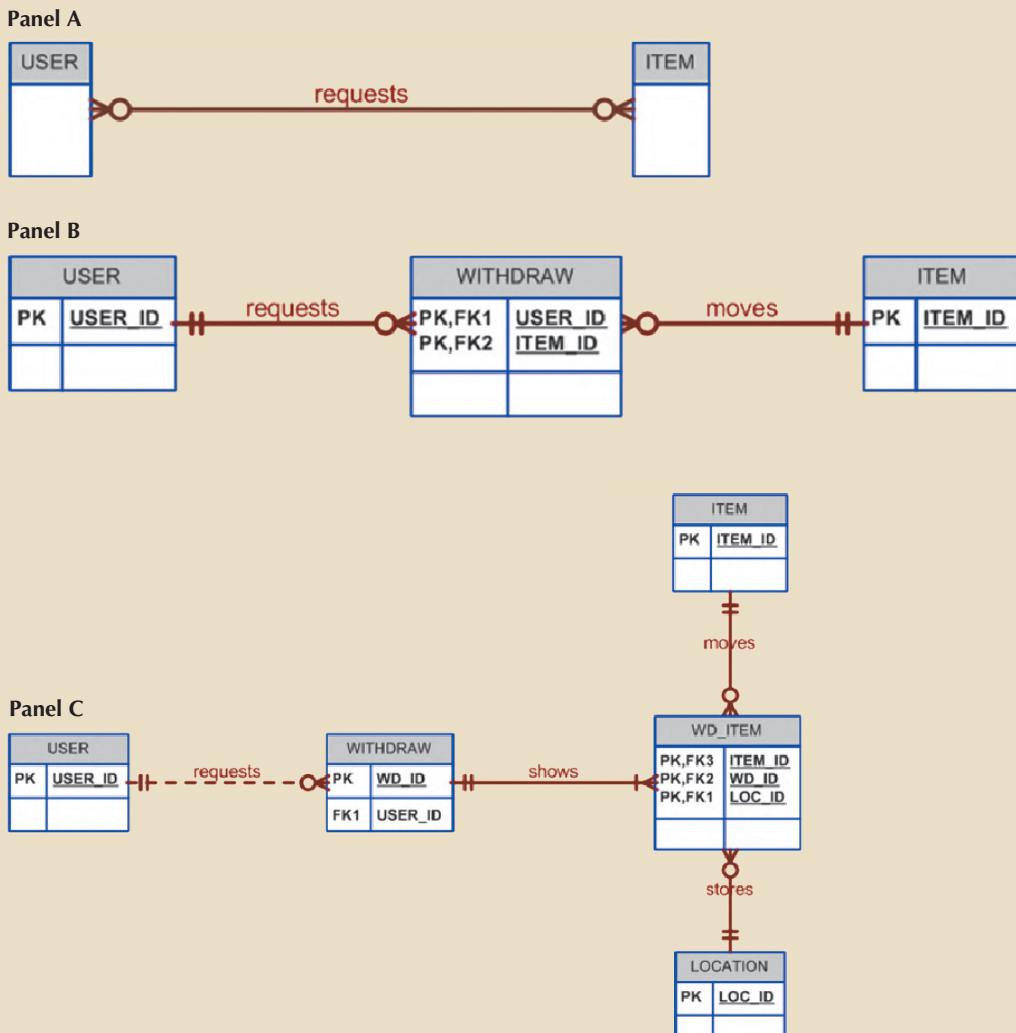
TABLE C.21

THE WITHDRAW ENTITY

ATTRIBUTE NAME	CONTENTS	ATTRIBUTE TYPE (*)	PRIMARY KEY (PK) AND/OR FOREIGN KEY (FK)	REFERENCES
WD_DATE	Withdrawal date		PK	
USER_ID	User ID (faculty or staff)		PK, FK	USER
ITEM_ID	Item ID for withdrawn item	M	PK, FK	ITEM
LOC_ID	Location ID	M	PK, FK	LOCATION
WD_QTY	Quantity withdrawn	M		

*The attribute type may be Composite (C), Derived (D), or Multivalued (M).

FIGURE C.19 THE WITHDRAW REVISION PROCESS



The **WITHDRAW** entity in Table C.21 seems to have all of the required attributes. In addition, Figure C.19, Panel B indicates that the addition of the **WITHDRAW** entity certainly has transformed the M:N relationship between **USER** and **ITEM** into two sets of 1:M relationships. Yet in spite of the design's improvement, **WITHDRAW** will not perform its intended functions well. Although its components help tie **USER**, **ITEM**, and **LOCATION** together, it contains three multivalued attributes. To eliminate those multivalued attributes, the **WITHDRAW** entity in Table C.21 can be decomposed into the two entities shown in Figure C.19, Panel C.

Using Figure C.19, Panel C as a guide, you can revise the **WITHDRAW** entity to eliminate the multivalued attributes and place them in **WD_ITEM**. Those two entities are shown in Tables C.22 and C.23 and their sample data are shown in Figures C.20 and C.21, respectively. (The data trace the withdrawal scenario presented at the beginning of this discussion.)

TABLE C.22**THE REVISED WITHDRAW ENTITY**

ATTRIBUTE NAME	CONTENTS	ATTRIBUTE TYPE (*)	PRIMARY KEY (PK) AND/OR FOREIGN KEY (FK)	REFERENCES
WD_ID	Withdrawal number		PK	
WD_DATE	Withdrawal date			
USER_ID	User ID (faculty or staff)		FK	USER

*The attribute type may be Composite (C), Derived (D), or Multivalued (M).

FIGURE C.20 SAMPLE WITHDRAW DATA

WD_ID	WD_DATE	USER_ID
325	04-Feb-2018	1-11-1111
326	04-Feb-2018	1-11-1128
327	07-Feb-2018	1-11-1130
328	08-Feb-2018	1-11-1109

TABLE C.23**THE WD_ITEM (WEAK) ENTITY**

ATTRIBUTE NAME	CONTENTS	ATTRIBUTE TYPE (*)	PRIMARY KEY (PK) AND/OR FOREIGN KEY (FK)	REFERENCES
WD_ID	Withdrawal ID		PK, FK	WITHDRAW
ITEM_ID	Item ID for withdrawn item		PK, FK	ITEM
LOC_ID	Location ID		PK, FK	LOCATION
WD_QTY	Quantity withdrawn			

*The attribute type may be Composite (C), Derived (D), or Multivalued (M).

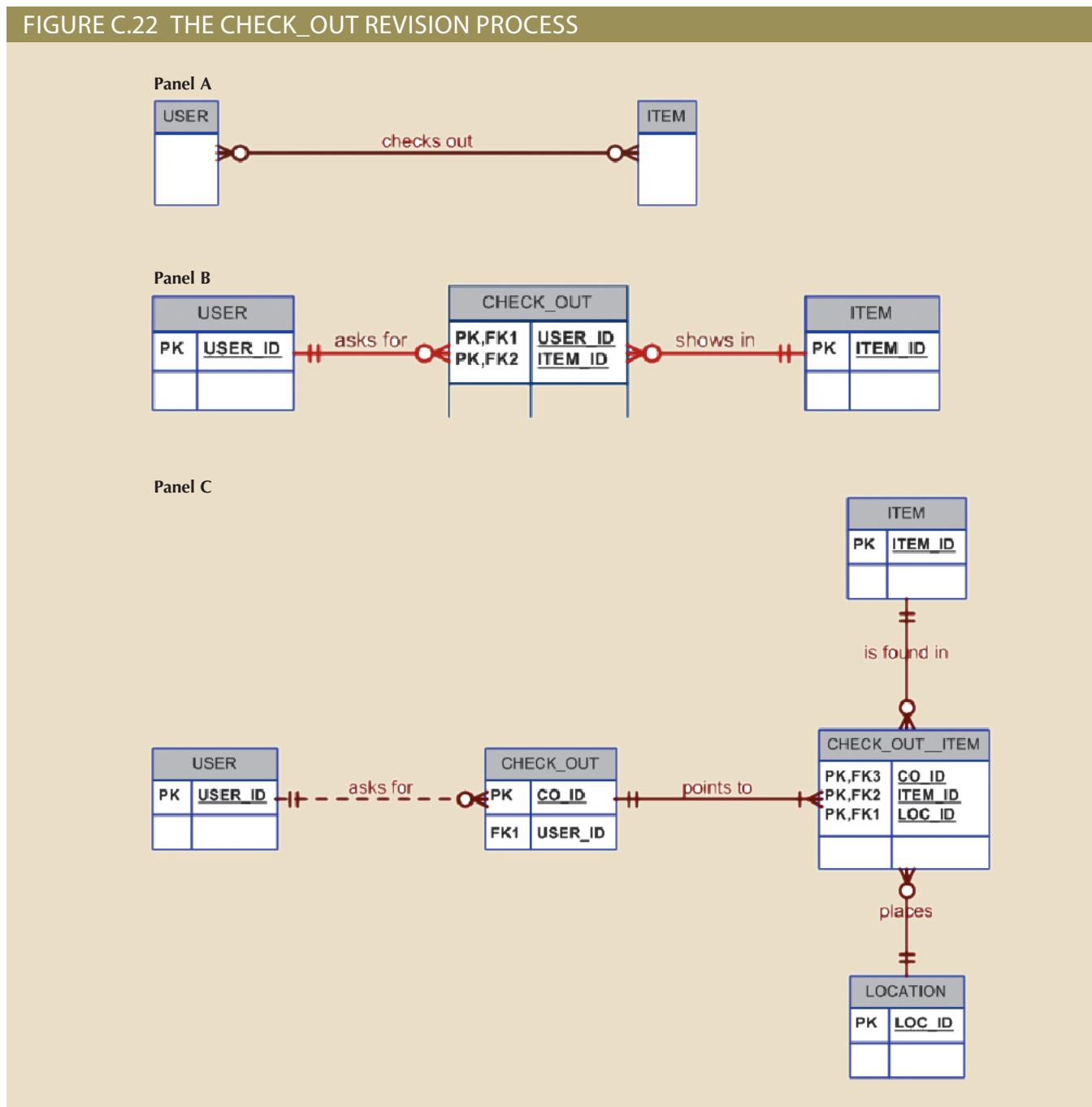
FIGURE C.21 SAMPLE WD_ITEM DATA

WD_ID	ITEM_ID	LOC_ID	WI_QTY
325	4238131	KOM245A-1	1
326	4238131	KOM245A-1	3
326	3154567	KOM245B-1	5
326	4238132	KOM245A-1	2
327	4238132	KOM106-1	1
328	3154567	KOM106-1	1
328	4238130	KOM106-1	1

WITHDRAW and WD_ITEM are capable of supporting the required withdrawal transactions, so this revision could be incorporated into the final design. However, another revision will be made later to standardize all inventory transactions.

Because the check-out transactions are subject to the same basic process as the withdrawal transactions, Figure C.22 illustrates that their design revisions mirror those of the withdrawal revision process. The difference between WITHDRAW and CHECK_OUT is that the latter yields two expected transactions for each item: one when the item is checked out and one when the item is returned.

FIGURE C.22 THE CHECK_OUT REVISION PROCESS



Because the check-out revision process basically mirrors that of the withdrawal revision, the discussion will simply note Figure C.22, Panels A and B, without providing any further revision details.

Using Figure C.22, Panel C as a design guide, Table C.24 defines the CHECK_OUT structure.

TABLE C.24

THE CHECK_OUT ENTITY

ATTRIBUTE NAME	CONTENTS	ATTRIBUTE TYPE (*)	PRIMARY KEY (PK) AND/OR FOREIGN KEY (FK)	REFERENCES
CO_ID	Check-out ID		PK	
CO_DATE	Check-out date			
USER_ID	User ID		FK	USER

* The attribute type may be Composite (C), Derived (D), or Multivalued (M).

To see the check-out process in action, let's trace the following transactions. (You will also see the difference between withdrawal and check-out when you look at the CO_ITEM's structure and sample data.)

- CO_ID = 101: Accounting faculty member 1-11-1117 checked out a laptop computer (4228753) from KOM245A-1 on February 2, 2018 and returned it on February 3, 2018.
- CO_ID = 102: CIS faculty member 1-11-1128 checked out a laptop computer (4358255) from KOM245B-1 and a projector panel (4358258) from KOM245B-1 on February 3, 2018. Only the laptop computer was returned on February 4, 2018.
- CO_ID = 103: CIS faculty member 1-11-1128 checked out the laptop computer (4228753) that was returned by the Accounting faculty member in transaction 101, from KOM245A-1 on February 3, 2018. The CIS faculty member has not yet returned the laptop.
- CO_ID = 104: CIS staff member 1-11-1112 checked out a laptop computer (4112151) from KOM245A-2 on February 4, 2018 and returned it on February 5, 2018.

Note how those transactions are reflected in Figures C.23 and C.24.

FIGURE C.23 SAMPLE CHECK_OUT DATA

CO_ID	CO_DATE	USER_ID
101	02-Feb-2018	1-11-1117
102	03-Feb-2018	1-11-1128
103	03-Feb-2018	1-11-1128
104	04-Feb-2018	1-11-1112

Each of the CHECK_OUT records will point to the transaction details—that is, the “Many” side, represented by the CHECK_OUT_ITEM entity shown in Table C.25.

TABLE C.25

THE CHECK_OUT_ITEM (WEAK) ENTITY

ATTRIBUTE NAME	CONTENTS	ATTRIBUTE TYPE (*)	PRIMARY KEY (PK) AND/OR FOREIGN KEY (FK)	REFERENCES
CO_ID	Check-out ID		PK	
ITEM_ID	Item ID		PK, FK	ITEM
LOC_ID	Location ID		PK, FK	LOCATION
COI_QTY	Check-out item quantity			
COI_DATE_IN	Date the item was returned			

* The attribute type may be Composite (C), Derived (D), or Multivalued (M).

The scenario is completed as shown in the sample data in Figure C.24.

FIGURE C.24 SAMPLE CHECK_OUT_ITEM DATA

CO_ID	ITEM_ID	LOC_ID	COI_QTY	COI_DATE_IN
101	4228753	KOM245A-1	1	03-Feb-2018
102	4358255	KOM245B-1	1	04-Feb-2018
102	4358258	KOM245B-1	1	
103	4228753	KOM245A-1	1	
104	4112151	KOM245A-2	1	05-Feb-2018

As you examine Figure C.24, note that it accurately portrays the transactions described earlier. Because item 4358258 in transaction 102 and item 4228753 in transaction 103 have not yet been returned, their COI_DATE_IN values are null. As was true in the case of the withdrawal process, you can now support the check-out and check-in transactions. However, you will discover in the next section that the inventory transaction process can be streamlined further.

C-3 Verifying the ER Model

Let's look at what you have accomplished. At this point, you have identified:

- *Entity sets, attributes, and domains.*
- *Composite attributes.* Such attributes may be (and usually are) decomposed into several independent attributes.
- *Multivalued attributes.* You implemented them in a new entity set in a 1:M relationship with the original entity set.
- *Primary keys.* You ensured primary key integrity.
- *Foreign keys.* You ensured referential integrity through the foreign keys.
- *Derived attributes.* You ensured the ability to compute their values.
- *Composite entities.* You implemented them with 1:M relations.

Although you have made considerable progress, much remains to be done before the model can be implemented.

To complete the UCL conceptual database design, you must *verify* the model. Verification represents the link between the database modeling and design activities, database implementation, and database application design. Therefore, the verification process is used to establish that:

- The design properly reflects the end-user or application views of the database.
- All database transactions—inserts, updates, deletes—are defined and modeled to ensure that the implementation of the design will support all transaction-processing requirements.
- The database design is capable of meeting all output requirements, such as query screens, forms, and report formats. (Remember that information requirements may drive part of the design process.)
- All required input screens and data entry forms are supported.
- The design is sufficiently flexible to support expected enhancements and modifications.

In spite of the fact that you were forced to revise the ER diagram initially depicted in Appendix B's Figure B.19, it is still possible that:

- Some of the relationships are not clearly identified and may even have been misidentified.
- The model contains design redundancies. (Consider the similarity between the WITHDRAW and CHECK_OUT entities.)
- The model can be enhanced to improve semantic precision and to better represent the operations in the real world.
- The model must be modified to better meet user requirements (such as processing performance or security).

The following few paragraphs will demonstrate the verification process for some of the application views in the Inventory Management module. (This verification process should be repeated for all of the system's modules.)

- *Identifying the central entity.* Although the satisfaction of the UCL's end users is vital, inventory management has the top priority from an administrative point of view. The reason for that priority rating is simple: state auditors target the Lab's considerable and costly inventory to ensure accountability to the state's taxpayers. Failure to track items properly may have serious consequences; therefore, ITEM becomes the UCL's central entity.
- *Identifying each module and its components.* Table C.2 identifies the modules and their components. It is important to "connect" the components by using shared entities. For example, although USER is classified as belonging to the Lab Management module and ITEM is classified as belonging to the Inventory Management module, the USER and ITEM entities interface with both. For example, the USER is written into the LOG in the Lab Management module. USER also features prominently in the Inventory Management module's withdrawal of supplies and in the check-out/check-in processes.
- *Identifying each module transaction requirement.* You will focus your attention on one of the INVENTORY module's reporting requirements. The authors suggest that you identify other transaction requirements. Then you can validate those requirements against the UCL database for all system modules.

An examination of the Inventory Management module's reporting requirements uncovers the following problems:

- The Inventory module generates three reports, one being an inventory movement report. But the inventory movements are spread across several different entities (CHECK_OUT and WITHDRAW and ORDER). That spread makes it difficult to generate the output and reduces system performance.
- An item's *quantity on hand* is updated with an inventory movement that can represent a purchase, withdrawal, check-out, check-in, or inventory adjustment. Yet only the *withdrawals* and *check-outs* are represented in the model.

The solution to those problems is described by the database designer:

What the Inventory Management module needs is a common entry point for all movements. In other words, the system must track all inputs to and withdrawals from inventory. To accomplish that task, we must create a new entity to record all inventory movements; that is, we need an inventory transaction entity. We will name that entity INV_TRANS.

The creation of a common entry point serves two purposes:

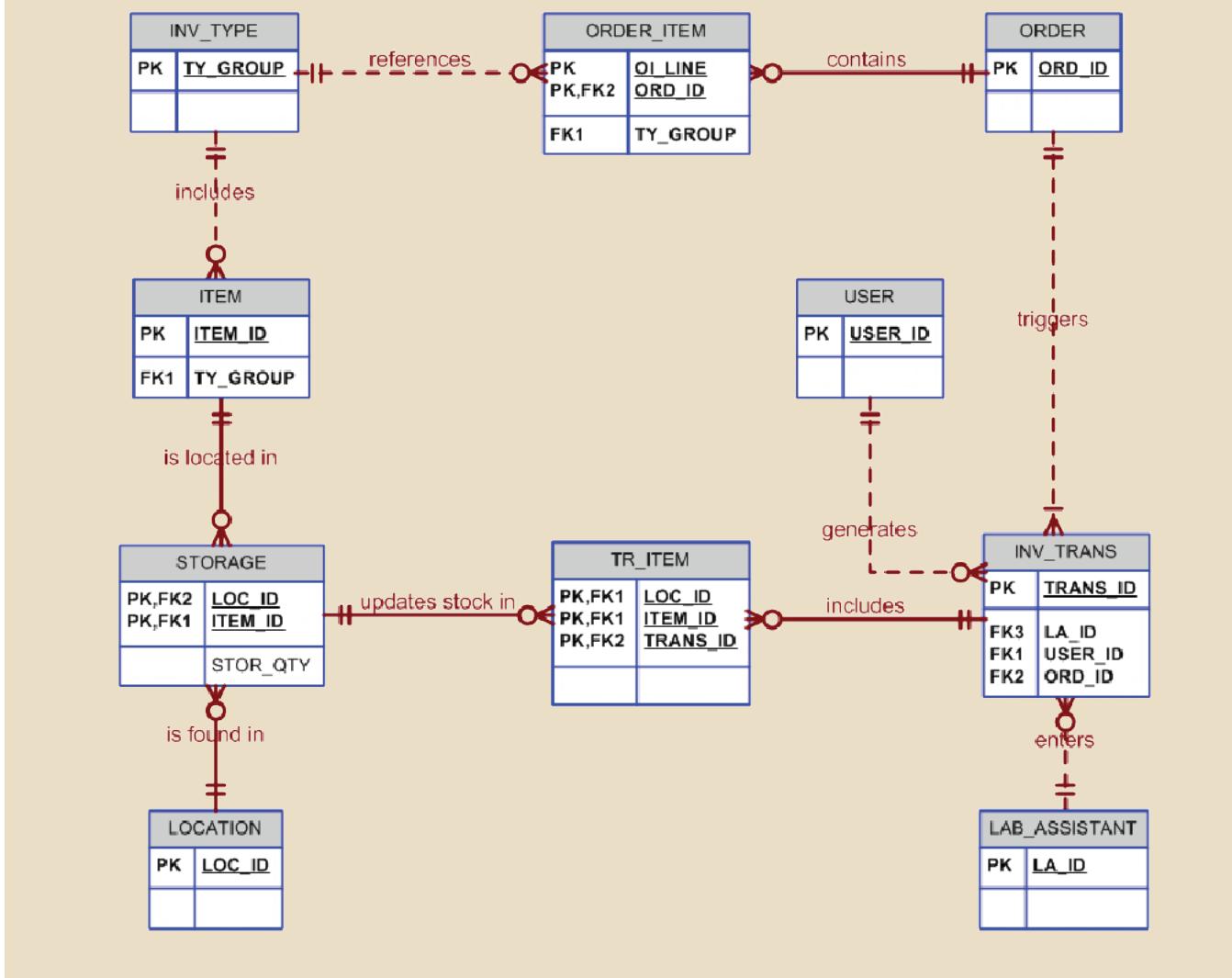
1. It standardizes the Inventory module's interface with other (external) modules. Any inventory movement (whether it adds or withdraws) will generate an inventory transaction entry.
2. It facilitates control and generation of required outputs, such as the inventory movement report.

Figure C.25 illustrates a solution to the problems just described.

The INV_TRANS entity in Figure C.25 is a simple inventory transaction log, and it will contain any inventory I/O movement. Each INV_TRANS entry represents one of two types of movement: input (+) or output (-). Each INV_TRANS entry must contain a line in TR_ITEM for each item that is added, withdrawn, checked in, or checked out.

The INV_TRANS entity's existence also enables you to build additional I/O movements efficiently. For example, when an ordered item is received, an inventory transaction entry (+) is generated. That INV_TRANS entry will update the quantity received (OI_QTY_RECVD) attribute of the ORDER_ITEM entity in the Inventory Management module. The Inventory Management module will generate an inventory transaction entry (-) to register the items checked out by a user, and it will generate another inventory transaction entry (+) to register the items checked in. The withdrawal of items (supplies) will also generate an inventory transaction entry (-) to register the items that are being withdrawn. Those relationships are depicted in Figure C.25.

FIGURE C.25 THE INVENTORY TRANSACTION PROCESS



The new INV_TRANS entity's attributes are shown in Table C.26.

TABLE C.26

THE INV_TRANS ENTITY

ATTRIBUTE NAME	CONTENTS	ATTRIBUTE TYPE (*)	PRIMARY KEY (PK) AND/OR FOREIGN KEY (FK)	REFERENCES
TRANS_ID	Inventory transaction ID (This code is generated by the system.)		PK	
TRANS_TYPE	Inventory transaction type: I = input (add to inventory) O = output (subtract from inventory)			

TABLE C.26

THE INV_TRANS ENTITY (CONTINUED)

ATTRIBUTE NAME	CONTENTS	ATTRIBUTE TYPE (*)	PRIMARY KEY (PK) AND/OR FOREIGN KEY (FK)	REFERENCES
TRANS_PURPOSE	Reason for inventory transaction: PO = purchase order (add to the inventory) CC = check-out (subtract from inventory) WD = withdrawal (subtract from inventory) AD = adjustment (add to or subtract from inventory, depending on the type of adjustment)			
TRANS_DATE	Inventory transaction date			
LA_ID	Lab assistant who recorded the transaction		FK	LAB_ASSISTANT
USER_ID	Person who created the transaction		FK	USER
ORDER_ID	Order ID		FK	ORDER
TRANS_COMMENT	Comments			

* The attribute type may be Composite (C), Derived (D), or Multivalued (M).

To see how INV_TRANS in Table C.26 works, refer to the ER segment in Figure C.25 and note that its detail lines are kept in the (weak) TR_ITEM. Figure C.25 also illustrates that all of the inventory movements can now be traced. For example, any item must be stored somewhere, so its location can be accessed through STORAGE. Because INV_TRANS is related to both LAB_ASSISTANT and USER, you know who recorded the transaction and who generated it. Figure C.26 contains sample data that will allow you to trace the:

- Withdrawal transactions first examined in Figures C.20 and C.21.
- Check-in and check-out transactions first examined in Figures C.23 and C.24.
- Purchase of 2 HP laser printers and 35 boxes of paper.

FIGURE C.26 SAMPLE INV_TRANS DATA

TRANS_ID	TRANS_TYPE	TRANS_PURPOSE	TRANS_DATE	LA_ID	USER_ID	ORDER_ID	TRANS_COMMENT
325	O	WD	04-Feb-2018	1-11-2003	1-11-1117		Laser printer cartridge
326	O	WD	04-Feb-2018	1-11-2008	1-11-1128		Laser printer cartridge
327	O	WD	07-Feb-2018	1-11-2003	1-11-1128		Box of 20 3.5" floppy disks, HD/DS
328	O	WD	08-Feb-2018	1-11-2008	1-11-1111		Two reams of 8.5x11" paper & ink-jet cartridge
401	O	CC	02-Feb-2018	1-11-2008	1-11-1120		Laptop check-out
402	I	CC	03-Feb-2018	1-11-2009	1-11-1120		Laptop returned
403	O	CC	03-Feb-2018	1-11-2003	1-11-1111		Laptop & projector check-out
404	O	CC	03-Feb-2018	1-11-2009	1-11-1112		Laptop check-out
405	I	CC	04-Feb-2018	1-11-2008	1-11-1111		Laptop returned
406	O	CC	04-Feb-2018	1-11-2021	1-11-1133		Laptop check-out
407	I	CC	05-Feb-2018	1-11-2003	1-11-1133		Laptop returned
408	I	PO	11-Feb-2018	1-11-2008	1-11-1128	122	Two HP printers, 35 boxes of paper order

For example, the first INV_TRANS row reveals that on February 4, 2018, a laser printer cartridge was withdrawn from inventory by user 1-11-1117. The transaction was recorded by LA 1-11-2003, and the transaction decreased (TRANS_TYPE = O) the stock in inventory.

The transaction details in Figure C.26 are stored in TR_ITEM, so before you can examine those details, you must examine the TR_ITEM structure in Table C.27.

TABLE C.27

THE TR_ITEM (WEAK) ENTITY

ATTRIBUTE NAME	CONTENTS	ATTRIBUTE TYPE (*)	PRIMARY KEY (PK) AND/OR FOREIGN KEY (FK)	REFERENCES
TRANS_ID	Inventory transaction ID (This code is generated by the system in the INV_TRANS entity.)		PK, FK	INV_TRANS
ITEM_ID	Item ID		PK, FK	ITEM
LOC_ID	Location ID		PK, FK	LOCATION
TRANS_QTY	Quantity withdrawn			

* The attribute type may be Composite (C), Derived (D), or Multivalued (M).

By examining the sample data shown in Figure C.27, you can trace the transaction details in Figure C.26.

FIGURE C.27 SAMPLE TR_ITEM DATA

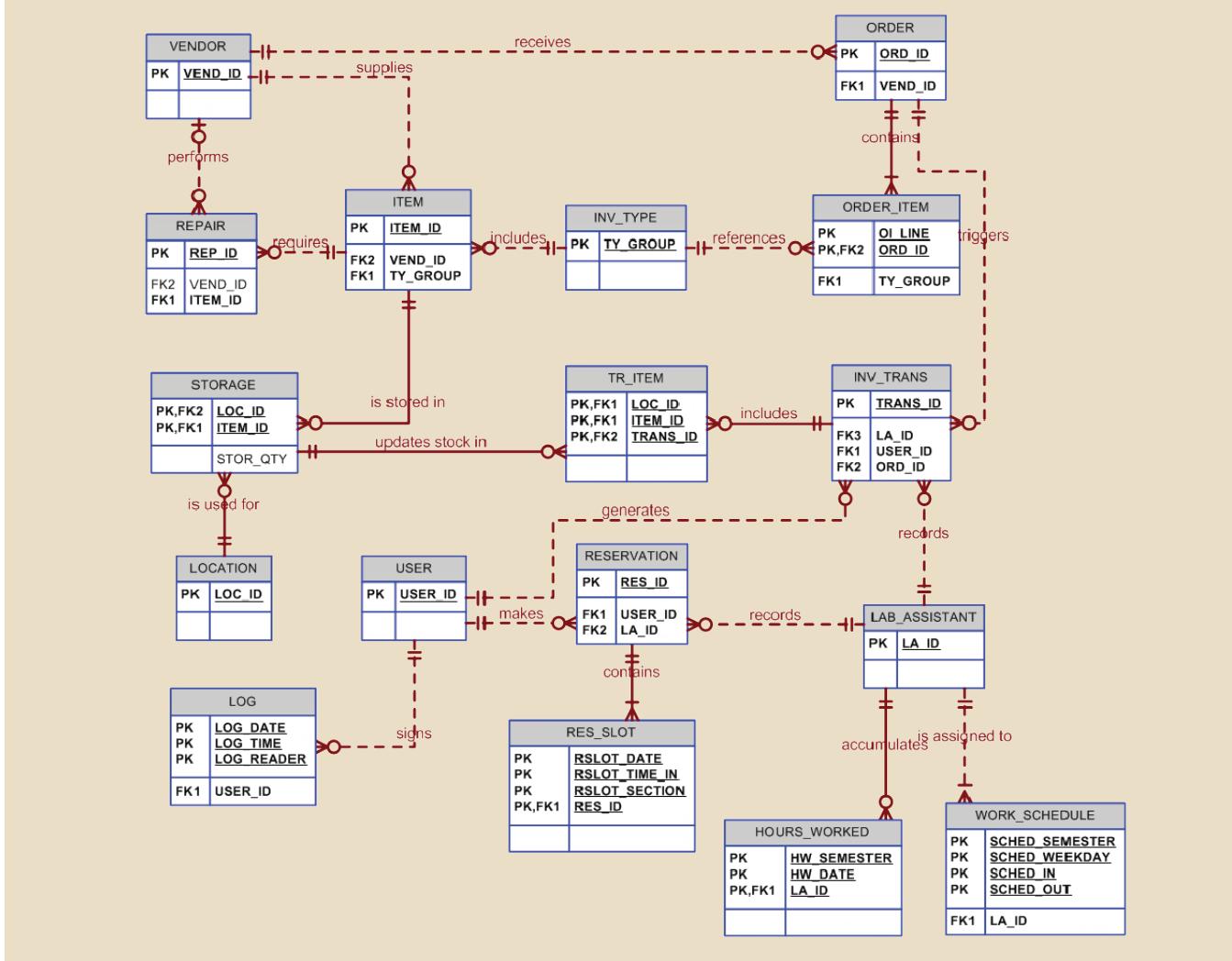
TRANS_ID	ITEM_ID	LOC_ID	TRANS_QTY
325	4238131	KOM245A-1	1
326	3154567	KOM245B-1	5
326	4238131	KOM245A-1	3
326	4238132	KOM245A-1	2
327	4238132	KOM106-1	1
328	3154567	KOM106-1	1
328	4238130	KOM106-1	1
401	4228753	KOM245A-1	1
402	4228753	KOM245A-1	1
403	4358255	KOM245B-1	1
403	4358258	KOM245B-1	1
404	4228753	KOM245A-1	1
405	4358255	KOM245B-1	1
406	4112151	KOM245A-2	1
407	4112151	KOM245A-2	1
408	3154567	KOM245A-2	35
408	4567920	KOM245B-2	1
408	4567921	KOM245B-2	1

For example, note that the first INV_TRANS row's TRANS_ID = 325 entry (see Figure C.26) now points to the TR_ITEM's TRANS_ID = 325 entry shown in Figure C.27, thus allowing you to conclude that that transaction involved the withdrawal of a single unit of item 4238131, a laser printer cartridge. (You can conclude that item 4238131 is a laser printer cartridge by examining the INV_TYPE and ITEM data in Figures C.11 and C.12, respectively, and noting that ITEM_ID = 4238131 corresponds to TY_GROUP = SUCALPXX.) Transaction 326 involved three items, so the TR_ITEM table contains three detail lines for that transaction.

Examine how check-outs and check-ins are handled. In Figure C.26, INV_TRANS transaction 401 records TRANS_PURPOSE = CC and TRANS_TYPE = O, indicating that a check-out was made. That transaction recorded the following: check-out of a laptop, ITEM_ID = 4228753 (see Figure C.21), on February 2, 2018 by an Accounting faculty member, USER_ID = 1-11-1120. The laptop was returned on February 3, 2018, and that transaction was recorded as TRANS_ID = 402, whose TRANS_PURPOSE = CC and TRANS_TYPE = I, indicating that this particular laptop was returned to the available inventory. Incidentally, because the department owns several laptops, faculty members need not wait for a laptop to be returned before checking one out, *as long as there are laptops in inventory*. However, if no additional laptops are available, the system can trace who has them and when they were checked out. If the CLD wants to place restrictions on the length of time an item can be checked out, this design makes it easy to notify users to return the items in question.

The final entity relationship diagram reflects the changes that have been made. Although the original ER diagram is easier to understand from the user's point of view, the revised ER diagram has more meaning from the *procedural* point of view. For example, the changes made are totally *transparent* (invisible) to the user because the user never sees the INV_TRANS entity. The final ER diagram is shown in Figure C.28.

FIGURE C.28 THE REVISED UNIVERSITY COMPUTER LAB ERD



C-4 Logical Design

When the conceptual design phase is completed, the ERD reflects—at the conceptual level—the business rules that, in turn, define the entities, relationships, optionalities, connectivities, cardinalities, and constraints. (Remember that some of the design elements cannot be modeled and are, therefore, enforced at the application level. For example, the constraint, “a checked-out item must be returned within five days” cannot be reflected in the ERD.) In addition, the conceptual model includes the definition of the attributes that describe each of the entities and that are required to meet information requirements.

Keep in mind that the conceptual model’s entities must be normalized before they can be properly implemented. The normalization process may yield additional entities and relationships, thus requiring the modification of the initial ERD. Because the focus was on the verification of the conceptual design to produce an *implementable* design, the model verified in this appendix was certain to meet the requisite normalization requirements. In short, design and normalization processes were used concurrently. In fact, such concurrent use of design and normalization reflects real-world practice. The logical design process is used to translate the conceptual design into the internal model for the selected DBMS. To the extent that normalization helps establish the appropriate attributes, their characteristics, and their domains, normalization moves you to the logical design phase. Nevertheless, because the conceptual modeling process does not preclude the definition of attributes, you can reasonably argue that normalization occasionally straddles the line between conceptual and logical modeling.

It bears repeating that the logical design translates the conceptual model in order to match the format expected of the DBMS that is used to implement the system. Because you will be using a relational database model, the logical design phase sets the stage for creating the relational table structures, indexes, and views.

C-4a Tables

The following few examples illustrate the design of the logical model, using SQL. (Make sure that the tables conform to the ER model’s structure and that they obey the foreign key rules if your DBMS software allows you to specify foreign keys.)

Using SQL, you can create the table structures within the database you have designated. For example, the STORAGE table would be created with:

```
CREATE TABLE STORAGE (
    LOC_ID      CHAR(12) NOT NULL,
    ITEM_ID     CHAR(10) NOT NULL,
    STOR_QTY    NUMBER,
    PRIMARY KEY (LOC_ID, ITEM_ID),
    FOREIGN KEY (LOC_ID) REFERENCES LOCATION
        ON DELETE RESTRICT
        ON UPDATE RESTRICT,
    FOREIGN KEY (ITEM_ID) REFERENCES ITEM
        ON DELETE CASCADE
        ON UPDATE CASCADE);
```

Most DBMSs now use interfaces that allow you to type the attribute names into a template and to select the attribute characteristics you want from pick lists. You can even insert comments that will be reproduced on the screen to prompt the user for input. For example, the preceding STORAGE table structure might be created in a Microsoft Access template, as shown in Figure C.29.

When all of the tables required by the design have been created, the relationships specified in the design are established. A good CASE tool will let you accomplish those

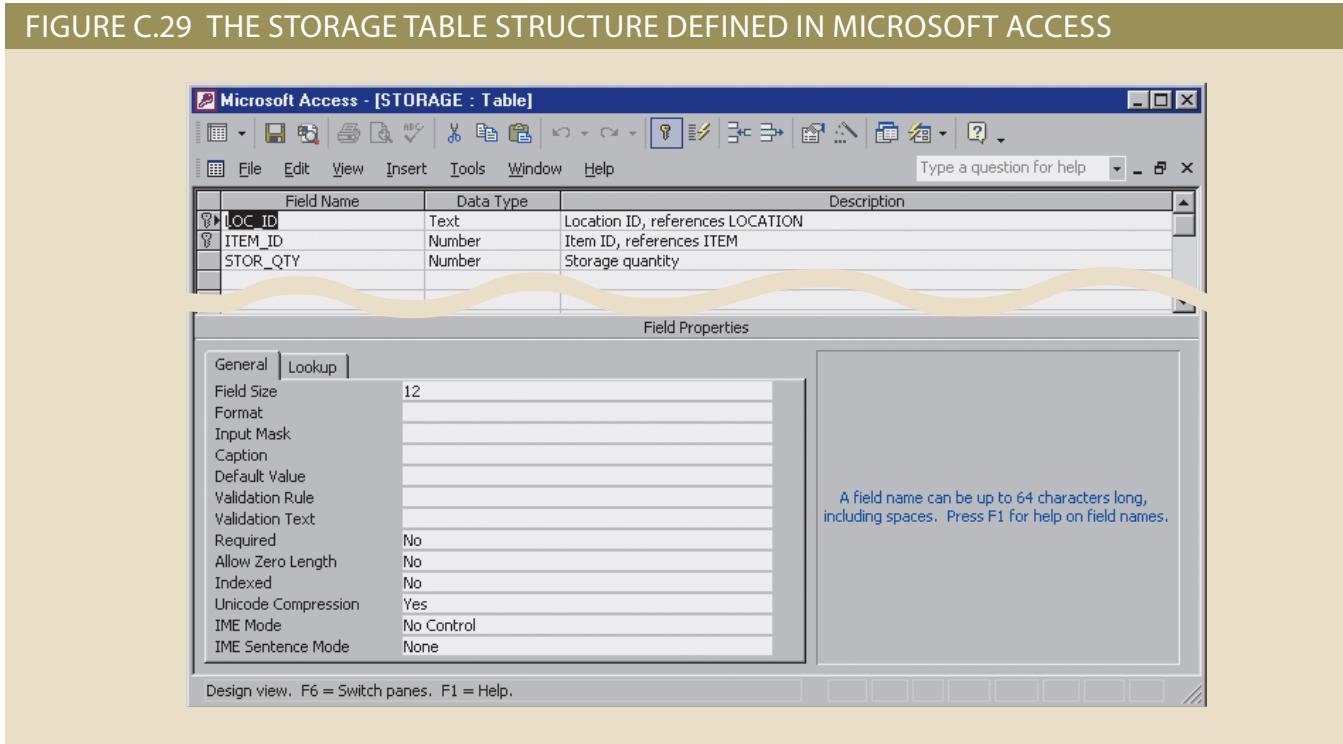
tasks directly from the design. For example, the design shown in Figure C.28 can be written into the specified database structure by the CASE tool. The advantages of letting the CASE tool write all of the table structures and relationships are that:

- The database will match the design precisely.
- All of the relationships have already been tested by the CASE tool to ensure that they are logically correct and that they are implementable as designed.
- All of the FK attribute definitions and characteristics match those of the referenced PKs.

Regardless of how you translate the design shown in Figure C.28 into the matching database structure, the database's relational schema must match the design. For example, Figure C.30 shows the relational schema in MS Access format.

As you examine the relational diagram in Figure C.30, note that all of its tables and relationships match the design specifications in Figure C.29. Note also that the relational diagram shows the addition of attributes that serve the end-user information and data management requirements.

FIGURE C.29 THE STORAGE TABLE STRUCTURE DEFINED IN MICROSOFT ACCESS



C-4b Indexes and Views

In the logical design phase, the designer can specify appropriate indexes to enhance operational speed. Indexes also enable the production of logically ordered output sequences. For example, if you want to generate the LA schedule shown in Table C.6, you need data from two tables, LAB_ASSISTANT and WORK_SCHEDULE. Because the report output is ordered by semester, LA, weekday, and time, indexes must be available for the primary key fields in each table. Using SQL, you would type:

```
CREATE UNIQUE INDEX LA_DEX
    ON LAB_ASSISTANT (LA_ID);
```

and

```
CREATE UNIQUE INDEX WS_DEX
    ON WORK_SCHEDULE (SCHED_SEMESTER, LA_ID, SCHED_WEEK-
DAY, SCHED_IN);
```

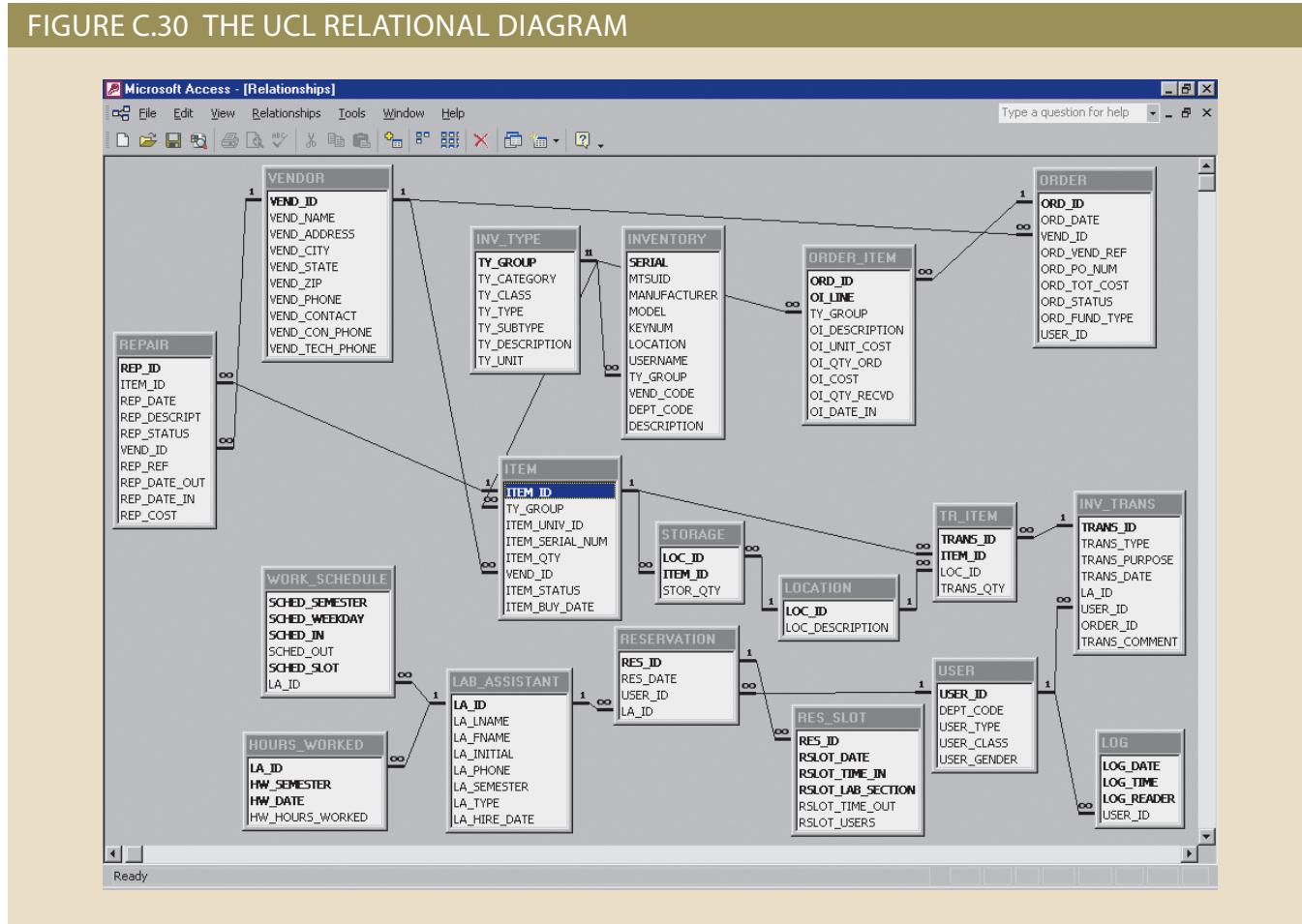
Most modern DBMSs automatically index on the primary key components.

Views (see Chapter 8, Advanced SQL) are often used for security purposes. However, views are also used to streamline the system's processing requirements. For example, output limits may be defined efficiently by specifying appropriate views. To define the view necessary for the LA schedule report for the Spring semester of 2012, the CREATE VIEW command is used:

```
CREATE VIEW LA_SCHED AS
```

```
SELECT LA_ID, LA_NAME, SCHED_WEEKDAY, SCHED_IN,
SCHED_OUT FROM WORK_SCHEDULE
WHERE SCHED_SEMESTER='SPRING12';
```

FIGURE C.30 THE UCL RELATIONAL DIAGRAM



The designer creates the views necessary for each database output operation.



Note

Unlike some other databases, the relational database model does not require the use of views in order to access the database. However, using views yields security benefits and greater output efficiency.

C-5 Physical Design

Physical design requires the definition of specific storage or access methods that will be used by the database. Within the DBMS's confines, the physical design must include an estimate of the space required to store the database. The required space estimate is translated into the space to be reserved within the available storage devices.

Physical storage characteristics are a function of the DBMS and the operating systems being used. Most of the information necessary to define the physical storage characteristics is found in the technical manuals of the software you are using. For example, if you use the newest SQL Server database, an estimate of the physical storage required for database creation (empty database) is provided by a table such as the one shown in Table C.28.

TABLE C.28

FIXED SPACE CLAIMED PER DATABASE

	DISK SPACE IN KB
Fixed space per table created within the database	535
17 tables 2-4 KB per table	68
Total fixed overhead used by database	603

Next, you need to estimate the data storage requirements for each table. Table C.29 shows the calculation for the USERS table only.

TABLE C.29

PHYSICAL STORAGE REQUIREMENTS: THE USER TABLE

ATTRIBUTE NAME	DATA TYPE	STORAGE REQUIREMENT (BYTES)
USER_ID	CHAR(11)	11
DEPT_CODE	CHAR(7)	7
USER_TYPE	CHAR(5)	5
USER_CLASS	CHAR(5)	5
USER_GENDER	CHAR(1)	1
Row length	29	
Number of rows	15,950	
Total space required	462,550	

If the DBMS does not automate the process of determining storage locations and data access paths, physical design requires well-developed technical skills and a precise knowledge of the physical-level details of the database, operating system, and hardware used by the database. Fortunately, the more recent versions of relational DBMS software hide most of the complexities inherent in the physical design phase.

You might store the database within a single volume of permanent storage space, or you can use several volumes, distributing the data in order to decrease data-retrieval time. Some DBMSs also allow you to create cluster tables and indexes. **Cluster tables** store rows of different tables together, in consecutive disk locations. That arrangement speeds up data access; it is mainly used in master/detail relationships such as ORDER and ORDER_ITEM or INV_TRANS and TR_ITEM.

The database designer must make decisions that affect data access time by fine-tuning the buffer pool size, the page frame size, and so on. Those decisions are based on the selected hardware platform and the DBMS. Consult the hardware and DBMS software manuals for the specific storage and access methodologies.

cluster tables

A data storage structure that physically stores related rows from different tables together to improve the speed at which related data can be accessed.

In the UCLMS, several indexes can be created to improve access time:

- Indexes created for all primary keys will increase access speed when you use foreign key references in tables. This is done automatically by the DBMS.
- Indexes can also be created for all alternative search keys. For example, if you want to search the LAB_ASSISTANT table by username, you should create an index for the LA_LNAME attribute; for example:
CREATE INDEX LA001 ON LAB_ASSISTANT (LA_LNAME);
- Indexes can be created for all secondary access keys used in reports or queries. For example, an inventory movement report is likely to be ordered by inventory type and item ID. Therefore, an index is created for the ITEM table:
CREATE INDEX INV002 ON ITEM (TY_GROUP, ITEM_ID);
- Indexes can be created for all columns used in the WHERE, ORDER BY, and GROUP BY clauses of a SELECT statement.

C-6 Implementation

One of the significant advantages of using a database is that it enables users to share data. When data are held in common, rather than being “owned” by various organizational divisions, data management becomes a more specialized task. Thus, the database environment favors the creation of a new organizational structure designed to manage the database resources. Database management functions are controlled by the database administrator (DBA). The DBA must define the standards and procedures required to interact with the database. (See Chapter 16, Database Administration and Security.)

Once the database designer has completed the conceptual, logical, and physical design phases, the DBA adopts an appropriate implementation plan. The plan includes formal definitions of the processes and standards to be followed, the chronology of the required activities (creation, loading, testing, and evaluation), the development of adequate documentation standards, the specific documentation needed, and the precise identification of responsibilities for continued development and maintenance.

Keep in mind that the technical details of the implementation are of little concern to the end user. Once the design has been implemented, the end users must be able to use the database and its contents—according to their work requirements and clearances—with relative ease and great utility. Therefore, the hard work of developing a user-friendly interface remains. Figures C.31 through C.34 show a sample main menu, a selection from that menu, some sample data entries, and the completed record based on a Microsoft Access database. Note that the end user interface shown in those figures uses several techniques to ensure appropriate data entries.

- *Drop-down lists* to limit the input selections. As you examine Figure C.32, note that the customer data have already been entered. In this case, the customer number 10011 was selected from a drop-down list of existing customers. The drop-down list is triggered by clicking on the downward-facing arrow button at the right margin of the customer input field. (Naturally, if the customer is new, a customer record must first be created.) Note that the customer financial data show up after the selection from the customer list, enabling the end user to authorize charges or to require full payment of the charter charges. Similarly, clicking on the downward-facing arrow button located on the right of the aircraft input field produces a drop-down list that shows all of the available aircraft and the relevant data for each aircraft. Those features enable the end user to answer customer questions without having to leave the input screen.
- *Automatic data entry completions* based on the input selections. For example, once an aircraft has been selected from the drop-down list, all appropriate field values for the

selected aircraft—such as the charge per mile and charge per waiting hour—are automatically written into the entry blanks. That feature eliminates end-user input errors and improves efficiency. (The end user does not need to type the values.)

- *System-generated computations* to avoid end-user computational errors. Once the distance flown and the waiting hours have been entered, all charges are calculated by the system, thus avoiding end-user calculation errors. (For example, Hours flown = Hobbs return – Hobbs out. A Hobbs meter is an instrument that records time.) Similarly, once the amount paid is entered, the balance is automatically calculated and entered into the “Balance owed” input field.

Many of the data entries in Figure C.33 are computed automatically. For example, the flight hours are computed after you have entered the Hobbs’ time in and you leave that field. The charges and the unpaid balance, if any, are also computed automatically. When the data entry is complete and you press the Update button, the affected tables

FIGURE C.31 THE RC-CHARTER2 COMPANY MAIN MENU

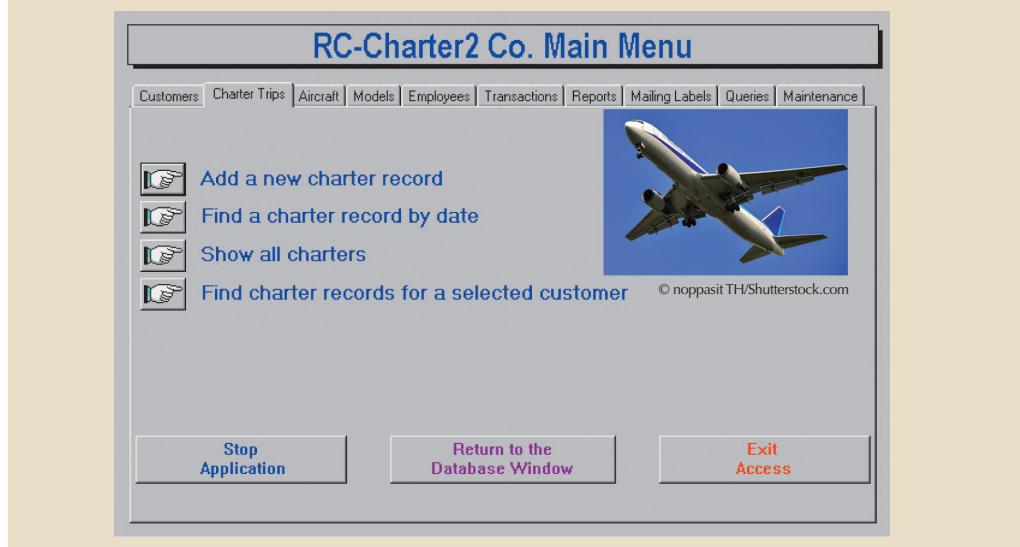


FIGURE C.32 THE RC-CHARTER2 COMPANY NEW CHARTER RECORD SELECTION

are updated, too. Note, for example, that the unpaid balance shown in Figure C.33 has been added to show the new customer balance. (Compare the customer balance value in Figure C.33 with its counterpart in Figure C.34.)

FIGURE C.33 CHARTER RECORD SAMPLE DATA ENTRIES

Customer:		Current balance:		Credit limit:	
Number	Last name	First name	In Balance	Limit	
10010	Ramas	Alfred	A \$2,796.78	\$5,000.00	
10011	Dunne	Leona	K \$4,348.09	\$5,000.00	
10012	Smith	Kathy	W \$1,272.61	\$6,500.00	
10013	Ołowski	Paul	F \$3,058.03	\$4,500.00	
10014	Orlando	Myron	G \$2,452.72	\$6,000.00	
10015	O'Brian	Amy	B \$2,849.52	\$4,500.00	
10016	Brown	James	C \$2,054.40	\$4,500.00	
# Pax:		Cargo (lbs.):		Payload (lbs.):	
Gross T.O wt. (lbs.):		Center of gravity (in.):			

Select the customer.

Customer:		Current balance:		Credit limit:		Aircraft:		Model:			
10015		\$2,849.52		\$4,500.00		1483J	BN2A-III-3	\$4.15	\$95.00	160	54
3766T						3766T	BN2A-III-3	\$4.15	\$95.00	160	54
2289L						2289L	C-90A	\$3.67	\$85.50	195	66
3385Q						3385Q	CV-580	\$37.55	\$548.75	350	350
# Pax:		Cargo (lbs.):		Payload (lbs.):		Record:					
Gross T.O wt. (lbs.):		Center of gravity (in.):		0.0		14		1 / 1 of 1			

Select the aircraft.

Customer:		Current balance:		Credit limit:		Aircraft:		Model:	
10015		\$2,849.52		\$4,500.00		2289L	C-90A		
Assignment:		Trip #	Employee	Last Name	First Name	Crew Assignment			
Crew member:									
# Pax:		Cargo (lbs.):		Payload (lbs.):		Record:			
Gross T.O wt. (lbs.):		Center of gravity (in.):		0.0		14			

Select the first assignment (Pilot in Command) from the crew assignment list.

Customer:		Current balance:		Credit limit:		Aircraft:		Model:	
10015		\$2,849.52		\$4,500.00		2289L	C-90A		
Assignment:		Trip #	Employee	Last Name	First Name	Crew Assignment			
Crew member:		1 Pilot in Command							
# Pax:		Cargo (lbs.):		Payload (lbs.):		Record:			
Gross T.O wt. (lbs.):		Center of gravity (in.):		0.0		14			

Select the crew member for the assignment and then submit the selection.

Customer:		Current balance:		Credit limit:		Aircraft:		Model:	
10015		\$2,849.52		\$4,500.00		2289L	C-90A		
Assignment:		Trip #	Employee	Last Name	First Name	Crew Assignment			
Crew member:		1	Submit Crew Assignment						
# Pax:		Cargo (lbs.):		Payload (lbs.):		Record:			
Gross T.O wt. (lbs.):		Center of gravity (in.):		0.0		14			

Select the next assignment and a crew member and then submit the selection. (Note that the first assignment is now shown to the right of the trip number.)

Customer:		Current balance:		Credit limit:		Aircraft:		Model:	
10015		\$2,849.52		\$4,500.00		2289L	C-90A		
Assignment:		Trip #	Employee	Last Name	First Name	Crew Assignment			
Crew member:		10299	114	Greenbriar	Claire	Pilot in Command			
# Pax:		Cargo (lbs.):		Payload (lbs.):		Record:			
Gross T.O wt. (lbs.):		Center of gravity (in.):		0.0		14			

Complete at least the passenger and loading information and the destination. The remaining information is supplied at the conclusion of the trip. (See the bottom half of the form in Figure C.32.)

FIGURE C.34 SAMPLE COMPLETED CHARTER RECORD

Trip #: **10297**
Trip date:

RC-Charter2 Co. charter data

Customer: **10020** Current balance: **\$1,160.32** Credit limit: **\$7,000.00** Aircraft: **3345K** Model: **PA31-350**

Pax: **3** Cargo (lbs.): **260** Payload (lbs.): **800** Gross T.O. wt. (lbs.): **4,500** Center of Gravity (in.): **37.1**

Trip #	Emp. #	Last Name	First Name	Crew Code	Crew Code Description
10297	101	Lewis	Rhonda	1	Pilot in Command
10297	105	Williams	Robert	2	Copilot

Record: **1** of 2

Destination: TLH	Distance flown: 840	Fuel used (Gallons): 132.8	Oil used (qts.): 0	Payment type: Check
Hobbs out: 2314.5	Hobbs return: 2318.1	Hours flown: 3.6	Waiting hours: 3.7	Check or CC #: 0000003214
Charge/Mile: \$3.35	Mileage charge: \$2,814.00	Charge Subtotal: \$3,135.90		Amount paid: \$2,500.00
Charge/hour: \$87.00	Waiting charge: \$321.90		Tax: \$250.87	Balance owed: \$886.77
Total charge: \$3,386.77				

Record: **1** of 875

Note

Keep in mind that even a beautifully crafted interface cannot overcome a poor database design. Unfortunately, too many organizations try to use applications development to overcome the limitations produced by poor database designs. Such attempts will lead to the inevitable failure of the organization's information requirements. (To use an analogy, you cannot overcome the problems of a poor building design by hiring better bricklayers—you just wind up with a poor building with beautiful brickwork.) That point is worth stressing—over and over and over!

As organizations become increasingly Internet-oriented, most of the database transaction interfaces tend to become Web interfaces. You were introduced to Web development issues in Chapter 15, “Database Connectivity and Web Technologies.” Appendix J, “Web Database Development with ColdFusion,” also discusses that topic.

C-6a Database Creation

All of the tables, indexes, and views that were defined during the logical design phase are created during this phase. The commands (or use utility programs) to create storage space and the access methods that were defined by the physical design are also issued at this time.

C-6b Database Loading and Conversion

The newly created database contains the (still empty) table structures. Those table structures can be filled by entering (typing) the data one table at a time or by copying the data

from existing databases or files. If the new table structures and formats are incompatible with those used by the original DBMS or file system software, the copy procedure requires the use of special loading or conversion utilities.

Because many processes require a precise sequencing of data activities, data are loaded in a specific order. Because of foreign key requirements, the University Computer Lab database must be initially loaded in the following order:

1. User, vendor, and location data.
2. Lab assistant and work schedule data.
3. Inventory type data.
4. Item data.

After those main entities have been loaded, the system is ready for testing and evaluation.

C-6c System Procedures

System procedures describe the steps required to manage, access, and maintain the database system. The development of those procedures constitutes a concurrent and parallel activity that started in the early stages of the system's design.

A well-run database environment requires and enforces strict standards and procedures to ensure that the data repository is managed in an organized manner. Although operational and management activities are logically connected, it is important to define distinct operations and management procedures.

In the case of the University Computer Lab Management System, procedures must be established to:

- Test and evaluate the database.
- Fine-tune the database.
- Ensure database security and integrity.
- Back up and recover the database.
- Access and use the database system.

Several databases may exist within a database environment. Each database must have its own set of system procedures and must be evaluated in terms of how it fits into the organization's information system.

C-7 Testing and Evaluation

The purpose of testing and evaluation is to determine how well the database meets its goals. Although testing and evaluation constitute a distinct DBLC phase, the implementation, testing, and evaluation of the database are concurrent and related. Database testing and evaluation should be ongoing. A database that is acceptable today may not be acceptable a few years from now because of rapidly changing information needs. In fact, an important reason for the relational database's growing dominance is its flexibility. (Because relational database tables are independent, changes can be made relatively quickly and efficiently.)

C-7a Performance Measures

Performance refers to the ability to retrieve information within a reasonable time and at a reasonable cost. A database system's performance can be affected by factors such as communication speeds, number of concurrent users, and resource limits. Performance,

measured primarily in terms of database query response time, is generally evaluated by computing the number of transactions per second. Performance issues are addressed in Chapter 11, “Database Performance Tuning and Query Optimization.”

C-7b Security Measures

Security measures seek to ensure that data are safely stored in the database. Security is especially critical in a multiuser database environment, in which someone might deliberately enter inconsistent data. The DBA must define (with the help of end users) a company information policy that specifies how data are stored, accessed, and managed within a data security and privacy framework.

Access may be limited by using access rights or access authorizations. Such rights are assigned on the basis of the user’s need to know or the user’s system responsibilities. In the case of the UCL database, access rights must be assigned to LAs, the CLD, and the CLS. But those rights are limited. For example, the LAs may read their work schedules, but they are not able to modify the data stored in the LAB_ASSISTANT or HOURS_WORKED tables.

The database administrator may, for example, grant the use of a previously created LA_SCHED view to the lab assistant Anne Smith by using the following (SQL) syntax:

```
GRANT SELECT ON LA_SCHED TO LA_ASMITH;
```

In this case, *only* the LA identified as LA_ASMITH may use the view LA_SCHED to check the LA schedules. A similar procedure is used to enable other LAs to check the Lab schedules.

Physical security deals with controlling access to rooms or buildings where data reside or are processed. Imagine someone unplugging the main computer while several update transactions are being executed! Physical security also includes protection of the database environment against fire, earthquakes, and other calamities.

C-7c Backup and Recovery Procedures

Database backup is crucial to the continued availability of the database system after a database or hardware failure has occurred. Backup must be a formal and frequent procedure, and backup files should be located in predetermined sites. Recovery procedures must delineate the steps to ensure full recovery of the system after a system failure or physical disaster.

The UCL system’s backup and recovery scenario includes the following procedures:

- Each computer in the system has an uninterrupted power supply to protect the computers against electrical interruptions.
- Periodic backups are made: daily for the most active tables and weekly for the less active tables. The backups are created during low system-use periods and are stored in different places to ensure physical safety.
- The database management system uses a transaction log to permit the recovery of the database from a given state when a disaster occurs.

C-8 Operation

Database operation, also called database management, is an ongoing venture, including all of the DBA’s administrative and technical functions, designed to ensure the database’s continuity. Before a database is declared operational, it must pass all operational tests and evaluations. The test and evaluation results must be formally approved by company management.

C-8a Database is Operational

An operational database provides all necessary support for the system's daily operations and maintains all appropriate operational procedures. If the database is properly designed and implemented, its existence not only enhances management's ability to obtain relevant information quickly, but also strengthens the entire organization's operational structure.

C-8b Operational Procedures

Database operational procedures are written documents in which the activities of the daily database operations are described. The operational procedures delineate the steps required to carry out specific tasks, such as data entry, database maintenance, backup and recovery procedures, and security measures.

C-8c Managing the Database: Maintenance and Evolution

After the database has become operational, management and control measures must be established for the database to be effective. The DBA is responsible for coordinating all operational and managerial aspects of the new DBMS environment. The DBA's responsibilities include:

- Monitoring and fine-tuning the database.
- Planning for and allocating resources for changes and enhancements.
- Planning for and allocating resources for periodic system upgrades.
- Providing preventive and corrective maintenance (backup and recovery procedures).
- Providing end-user management services by creating and defining users, passwords, privileges, and so on.
- Performing periodic security audits.
- Performing necessary training.
- Establishing and enforcing database standards.
- Marketing the database to the organization's users.
- Obtaining funding for database operations, upgrades, and enhancements.
- Ensuring completion of database projects within time and budget constraints.

In short, the DBA performs technical and managerial duties that ensure the proper operation of the database to support the organization's mission. Therefore, the DBA's activities are designed to support the end-user community through the creation and enforcement of database-related policies, procedures, standards, security, and integrity that, in turn, foster the generation and proper use of information. A more detailed discussion of the database administration function is provided in Chapter 16.

Key Terms

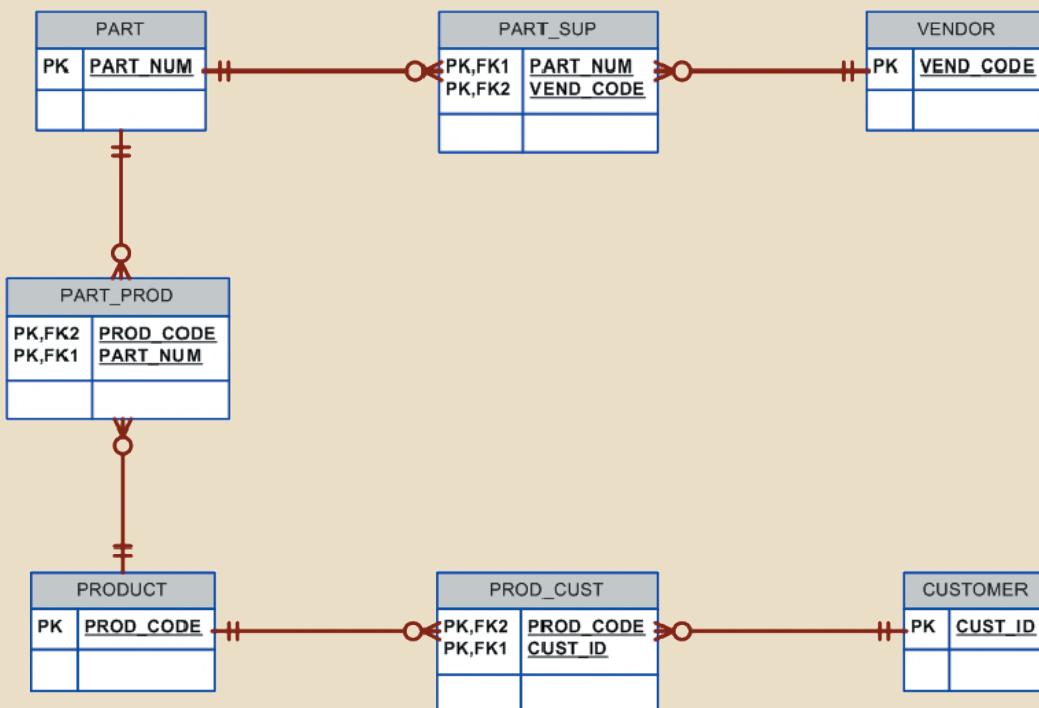
cluster tables, C-44

verification, C-1

Review Questions

1. Why must a conceptual model be verified? What steps are involved in the verification process?
2. What steps must be completed before the database design is fully implemented? (Make sure that you list the steps in the correct sequence and discuss each step briefly.)
3. What major factors should be addressed when database system performance is evaluated? Discuss each factor briefly.
4. How would you verify the ER diagram shown in Figure QC.4? Make specific recommendations.

FIGURE QC.4 THE ERD FOR QUESTION 4

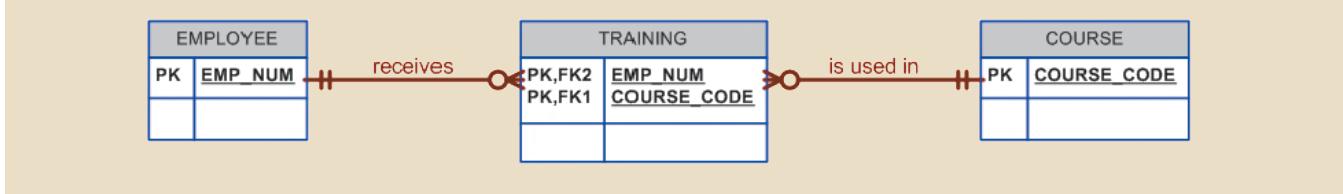


5. Describe and discuss the ER model's treatment of the UCL's inventory/order hierarchy:
 - a. Category
 - b. Class
 - c. Type
 - d. Subtype
6. Modern businesses tend to provide continuous training to keep their employees productive in a fast-changing and competitive world. In addition, government regulations often require certain types of training and periodic retraining. (For example, pilots must take semiannual courses involving weather, air regulations, and so on.) To make sure that an organization can track all training received by each of its employees, trace the development of the ERD segment in Figure QC.6 from the initial business rule that states:

An employee can take many courses, and each course can be taken by many employees.

Once you have traced the development of the ERD segment, verify it and then provide sample data for each of the three tables to illustrate how the design would be implemented.

FIGURE QC.6 THE ERD FOR QUESTION 6



7. You read in this appendix that *an examination of the UCL's Inventory Management module reporting requirements uncovered the following problems:*

- The Inventory module generates three reports, one of which is an inventory movement report. But the inventory movements are spread across two different entities (CHECK_OUT and WITHDRAW). That spread makes it difficult to generate the output and reduces the system's performance.
- An item's quantity on hand is updated with an inventory movement that can represent a purchase, a withdrawal, a check-out, a check-in, or an inventory adjustment. Yet only the withdrawals and check-outs are represented in the system.

What solution was proposed for that set of problems? How would such a solution be useful in other types of inventory environments?

Problems

1. Verify the conceptual model you created in Appendix B, Problem 3. Create a data dictionary for the verified model.
2. Verify the conceptual model you created in Appendix B, Problem 4. Create a data dictionary for the verified model.
3. Verify the conceptual model you created in Appendix B, Problem 5. Create a data dictionary for the verified model.
4. Verify the conceptual model you created in Appendix B, Problem 6. Create a data dictionary for the verified model.
5. Verify the conceptual model you created in Appendix B, Problem 7. Create a data dictionary for the verified model.
6. Design (through the logical phase) a student-advising system that will enable an advisor to access a student's complete performance record at the university. A sample output screen should look like the one shown in Table PC.6.
7. Design and verify a database application for one of your local not-for-profit organizations (for example, the Red Cross, the Salvation Army, your church or synagogue). Create a data dictionary for the verified design.

TABLE PC.6

THE STUDENT TRANSCRIPT FOR PROBLEM 6

Name: XXXXXXXXXXXXXXXX X XXXXXXXXXXXXXXXXXX			Page # of ##
Department: XXXXXXXXXXXXXXXXXXXX	Major: XXXXXXXXXXXXXXXXXXXX		
Social Security Number: ####-##-####	Report Date: ## XXXXXXXXXXXXX ####		
Fall 2012			
Course	Hours	Grade	Grade Points
CIS 200 (Intro to Microcomputers)	3	B	##
XXX #### (XXXXXXXXXXXXXXXXXX)	#	X	##
XXX #### (XXXXXXXXXXXXXXXXXX)	#	X	##
XXX #### (XXXXXXXXXXXXXXXXXX)	#	X	##
XXX #### (XXXXXXXXXXXXXXXXXX)	#	X	##
Total this semester:	##	GPA	#.##
Total to date:	###	Cumulative GPA	#.##
Spring 2013			
Course	Hours	Grade	Grade Points
CIS 300 (Computers in Society)	3	B	##
XXX #### (XXXXXXXXXXXXXXXXXX)	#	X	##
XXX #### (XXXXXXXXXXXXXXXXXX)	#	X	##
XXX #### (XXXXXXXXXXXXXXXXXX)	#	X	##
XXX #### (XXXXXXXXXXXXXXXXXX)	#	X	##
Total this semester:	##	GPA	#.##
Total to date:	###	Cumulative GPA	#.##
Summer 2013			
Course	Hours	Grade	Grade Points
CIS 400 (Systems Analysis)	3	B	##
XXX #### (XXXXXXXXXXXXXXXXXX)	#	X	##
XXX #### (XXXXXXXXXXXXXXXXXX)	#	X	##
XXX #### (XXXXXXXXXXXXXXXXXX)	#	X	##
Total this semester:	##	GPA	#.##
Total to date:	###	Cumulative GPA	#.##

8. Using the information given in the physical design section (C-5), estimate the space requirements for the following entities:
- RESERVATION
 - INV_TRANS
 - TR_ITEM
 - LOG
 - ITEM
 - INV_TYPE

Hint: You may want to check Appendix B, Table B.3, A Sample Volume of Information Log.

Appendix D

Converting an ER Model into a Database Structure

Preview

Converting any ER model to a set of tables in a database requires following specific rules that govern the conversion. The application of those rules requires an understanding of the effects of updates and deletions on the tables in the database. Before discussing these rules in detail, let's briefly review a simple ER model and the SQL commands used to generate the tables.

Data Files and Available Formats

MS Access | **Oracle** | **MS SQL** | **My SQL**

MS Access | **Oracle** | **MS SQL** | **My SQL**

Artist



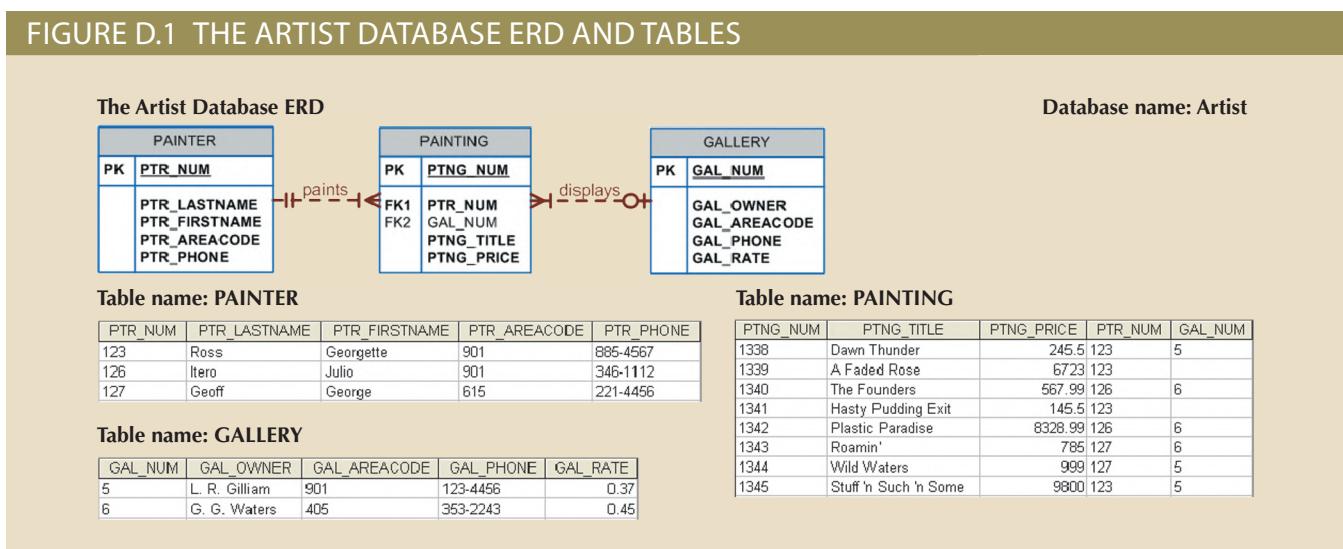
Data Files Available on cengagebrain.com

D-1 The Artist Database

To illustrate the conversion of an ER model into a database structure, let's use the Artist database, located in the Premium Website for this book. The Artist database conforms to the following conditions:

- A painter might paint many paintings. To be considered a painter in the Artist database, the painter must have painted at least one painting. This business rule decrees that the cardinality is (1,N) in the relationship between PAINTER and PAINTING.
- Each painting is painted by one (and only one) painter.
- A painting might (or might not) be exhibited in a gallery; that is, GALLERY is an optional entity to the PAINTING entity.

Given that description, let's use a simple ER model and some matching tables for the Artist database, shown in Figure D.1.



The data dictionary in Table D.1 shows the characteristics of the attributes found in the three tables.

Given the information in Figure D.1 and Table D.1, note that

- PTR_NUM in the PAINTING table is the foreign key that references the PAINTER table. Because the relationship between PAINTER and PAINTING is mandatory, the PTR_NUM foreign key must be classified as NOT NULL.
- GAL_NUM in PAINTING is the foreign key that references the GALLERY table. Because the relationship between PAINTING and GALLERY is optional, the GAL_NUM foreign key may be null.

TABLE D.1 A DATA DICTIONARY FOR THE ARTIST DATABASE

TABLE NAME	ATTRIBUTE NAME	CONTENTS	TYPE	FORMAT	RANGE	REQUIRED	PK OR FK	FK REFERENCED TABLE
PAINTER	PTR_NUM PTR_LASTNAME PTR_FIRSTNAME PTR_AREACODE PTR_PHONE	Painter number Painter last name Painter first name Painter area code Painter phone	CHAR(4) VARCHAR(15) VARCHAR(15) CHAR(3) CHAR(8)	9999 Xxxxxxxxxxx Xxxxxxxxxxx 999 999-9999	1000-9999	Y Y Y	PK	
GALLERY	GAL_NUM GAL_OWNER GAL_AREACODE GAL_PHONE GAL_RATE	Gallery number Gallery owner Gallery area code Gallery phone Gallery commission rate (pct.)	CHAR(4) VARCHAR(35) CHAR(3) CHAR(8) NUMBER(4,2)	9999 Xxxxxxxxxxx 999 999-9999 99.99	1000-9999 0.00-60.00	Y Y Y Y	PK	
PAINTING	PTNG_NUM PTNG_TITLE PTNG_PRICE	Painting number Painting title Painting price	CHAR(4) VARCHAR(35) NUMBER(9,2)	9999 Xxxxxxxxxxx 99,999.99	1000-9999 10.00-99,999.99	Y Y	PK	
	PTR_NUM GAL_NUM	Painter number Gallery number	CHAR(4) CHAR(4)	9999 9999	1000-9999 1000-9999	Y Y	FK FK	PAINTER GALLERY

D-1a The Effect of Foreign Key Constraints on Data Manipulation

Given the Artist database table structures, let's examine the effect of the following data manipulation events of the foreign key constraint actions:

1. *Adding a painter (row) to the PAINTER table.* Adding a painter does not cause any problems because the PAINTER table does not have any dependencies in other tables.
2. *Updating the PAINTER table's primary key.* Changing a PAINTER key causes problems in the database because some paintings in the PAINTING table may make reference to this key. The option is to use the UPDATE CASCADE. This option makes sure that a change in the PAINTER's PTR_NUM automatically triggers the required changes in the PTR_NUM foreign key found in other tables. This is the recommended option. This behavior (UPDATE CASCADE) is not supported by some RDBMS products, such as Oracle.
3. *Deleting a painter (row) from the PAINTER table.* If you delete a row (painter) from the PAINTER table, the PAINTING table may contain references to a painter who no longer exists, thereby creating a deletion anomaly. (A painting does not cease to exist just because the painter does.) Given this situation, it is wise to restrict the ability to delete a row from a table when there is a foreign key in another table that references the row. The restriction means that you can delete a painter from the PAINTER table only when there is no foreign key in another table related to this painter row. This behavior is enforced automatically by the RDBMS when using the FOREIGN KEY clause.
4. *Adding a gallery (row) to the GALLERY table.* Adding a new row does not affect the database because the GALLERY does not have dependencies in other tables.
5. *Updating the GALLERY table's primary key.* Changing a primary key value in a GALLERY row requires that all foreign keys making reference to it be updated as well. Therefore, you must use an UPDATE CASCADE clause. This option makes sure that a change in the GALLERY's GAL_NUM automatically triggers the required changes in the GAL_NUM foreign key found in other tables. This is the recommended option. This behavior (UPDATE CASCADE) is not supported by some RDBMS products, such as Oracle.
6. *Deleting a gallery (row) from the GALLERY table.* Deleting a GALLERY row creates problems in the database when rows in the PAINTING table make reference to the GALLERY row's primary key. Because GALLERY is optional to PAINTING, you may set all of the deleted gallery GAL_NUM values to null (DELETE SET NULL). Or you may want the database user to be alerted to the problem by specifying that the deletion of a GALLERY row is permitted only when there is no foreign key (GAL_NUM) in another table that requires the GALLERY row's existence. This behavior is enforced automatically by the RDBMS when using the FOREIGN KEY clause.

D-1b Transforming the ER Model into a Set of Tables

Armed with the results discussed in Section D-1a, you can now transform the ER model into a set of tables by using the following SQL commands:

1. Create the PAINTER table:

```
CREATE TABLE PAINTER (
    PTR_NUM      CHAR(4)      NOT NULL      UNIQUE,
    PTR_LASTNAME   CHAR(15)     NOT NULL,
    PTR_FIRSTNAME  CHAR(15)     NOT NULL,
    PTR_AREACODE    CHAR(3),
    PTR_PHONE       CHAR(8),
    PRIMARY KEY (PTR_NUM));
```

2. Create the GALLERY table:

```
CREATE TABLE GALLERY (
    GAL_NUM      CHAR(4)      NOT NULL UNIQUE,
    GAL_OWNER     CHAR(35),
    GAL_AREACODE  CHAR(3)      NOT NULL,
    GAL_PHONE      CHAR(8)      NOT NULL,
    GAL_RATE        NUMBER(4,2),
    PRIMARY KEY (GAL_NUM));
```

3. Create the PAINTING table:

```
CREATE TABLE PAINTING (
    PTNG_NUM      CHAR(4)      NOT NULL UNIQUE,
    PTNG_TITLE     CHAR(35),
    PTNG_PRICE      NUMBER(9,2),
    PTR_NUM        CHAR(4)      NOT NULL,
    GAL_NUM        CHAR(4),
    PRIMARY KEY(PTNG_NUM),
    FOREIGN KEY (PTR_NUM) REFERENCES PAINTER
    ON UPDATE CASCADE,
    FOREIGN KEY (GAL_NUM) REFERENCES GALLERY
    ON UPDATE CASCADE);
```

After creating the database tables and entering their contents, you are now ready for data entry, queries, and reports. Note that the decisions made by the designer to govern data integrity are reflected in the foreign key rules. Implementation decisions vary according to the problem being addressed.

D-2 General Rules Governing Relationships Among Tables

Given the experience with the simple Artist database in the previous section, here is a general set of rules to help you create any database table structure that will meet the required integrity constraints.

1. All primary keys must be defined as NOT NULL and UNIQUE. If your applications software does not support the NOT NULL option, you should enforce the condition by using programming techniques. This is another argument for using DBMS software that meets ANSI SQL standards.
2. Define all foreign keys to conform to the following requirements for binary relationships.

1:M Relationships Create the foreign key by putting the primary key of the “one” in the table of the “many.” The “one” side is referred to as the parent table, and the “many” side is referred to as the dependent table. Observe the following foreign key rules:

If both sides are MANDATORY:

Column constraint:	NOT NULL
FK constraint:	Default behavior (on delete restrict)
	ON UPDATE CASCADE

If both sides are OPTIONAL:

Column constraint:	NULL ALLOWED
FK constraint:	ON DELETE SET NULL
	ON UPDATE CASCADE

If one side is OPTIONAL and one side is MANDATORY:

- a. If the “many” and the mandatory components of the relationship are on the same side in the ER diagram, a NULL ALLOWED condition must be defined for the dependent table’s foreign key. The foreign key rules should be:

Column constraint:	NULL ALLOWED
FK constraint:	ON DELETE SET NULL or
	default behavior: ON DELETE RESTRICT
	ON UPDATE CASCADE

- b. If the “many” and the mandatory components of the relationship are not on the same side in the ER diagram, a NOT NULL condition must be defined for the dependent table’s foreign key. Deletion and update in the parent table of the foreign key should be subject to default behavior (on delete restrict) and UPDATE CASCADE restrictions.

Weak Entities

- a. Put the key of the parent table (the strong entity) in the weak entity. The key of the weak entity will be a composite key composed of the parent table key and the weak entity candidate key, if any. (The designer may decide to create a new unique ID for the entity.)
- b. The weak entity relationship conforms to the same rules as the 1:M relationship, except for the following foreign key restrictions:

Column constraint:	NOT NULL
FK constraint:	ON DELETE CASCADE
	ON UPDATE CASCADE

M:N Relationships Convert the M:N relationship to a composite (bridge) entity consisting of (at least) the parent tables’ primary keys. Thus, the composite entity primary key is a composite key that is subject to the NOT NULL restriction.

1:1 Relationships If both entities are in a mandatory participation in the relationship and they do not participate in other relationships, it is most likely that the two entities should be part of the same entity.

Table D.2 summarizes the ramifications of the foreign key actions that could be used to represent multiple cases of 1:1, 1:M, and M:N relationships.

TABLE D.2
SUMMARY OF FOREIGN KEY RULES

RELATIONSHIP	FOREIGN KEY LOCATION	THE ENTITIES PARTICIPATING IN THE RELATIONSHIPS ARE...	FOREIGN KEY ATTRIBUTE CONSTRAINT			FOREIGN KEY ACTIONS	
			DELETE	UPDATE	DELETE	UPDATE	DELETE
M:N	New entity composite primary key	Both mandatory Both optional One mandatory, one optional If FK located on mandatory side If FK located on optional side	NN NN	R C	C C	C C	C C
1:M	Many side	Both mandatory Both optional One mandatory, one optional If FK located on mandatory side If FK located on optional side	NN NA NN	R SN or R R	C C C	C C C	C C C
1:1	Foreign key placement is a matter of informed choice.* Put the FK in the ER's optional side, the strong entity, the most frequently accessed side, or the side dictated by the case semantics. <i>Do not put the FK in both sides.</i>	Both mandatory Both optional One mandatory, one optional If FK located on mandatory side If FK located on optional side	NN NA NN	R SN R	C C C	C C C	C C C
Weak	Weak entity	Create a set of new tables in 1:M relationships. Conform to the weak entity rules.	NN**	C	C	C	C
Multivalued Attributes			NN	C	C	C	C
NN = Not Null SN = Set to Null		NA = Null Allowed C = Cascade	R = Restrict * = See Chapter 5, Advanced Data Modeling, for a detailed discussion of the 1:1 relationship.	** = Inherited from parent entity			

Appendix E

Comparison of ER Modeling Notations

Preview

The ER models used in this text are based on the Chen and Crow's Foot notations. However, you should be aware of other ER notations, including the Rein85 and the IDEF1X. Although those ER notations are based on the same modeling concepts, such alternative notations were developed because they fit computer-based modeling tools more easily than the original Chen notation. It is quite likely that you will convert even the finest Chen model into those (or very similar) models when you are using computer-assisted systems engineering (CASE) tools in the database environment.

Data Files and Available Formats

[MS Access](#) [Oracle](#) [MS SQL](#) [My SQL](#)

[MS Access](#) [Oracle](#) [MS SQL](#) [My SQL](#)

There are no data files for this appendix.

Data Files Available on cengagebrain.com

The following list summarizes the major characteristics of each notation:

- The *Chen model* is based on Peter Chen's landmark work "The Entity-Relationship Model: Toward a Unified View of Data," *ACM Transactions on Database Systems* 1(1), March 1971. The Chen model moved conceptual modeling into the practical database design arena by establishing the basic building blocks: entities and relationships. Chen's work was enhanced by T. J. Teorey, D. Yang, and J. P. Fry when they published "A Logical Design Methodology for Relational Databases Using the Extended Entity Relationship Model," *ACM Computing Surveys*, June 1986, pp. 197–222. The Chen model's basic structure, with the enhancements made by Teorey, Yang, and Fry, became a dominant player in the CASE tool market during the late 1980s and early 1990s. (See especially Mike Ricciuti's "Database Vendors Make Their CASE," *Datamation* 38(5), March 1992.) Excelerator, a CASE tool of choice for many database designers in the early 1990s, is perhaps the best "pure" Chen modeling tool. Although the Chen model is no longer the dominant ERD generator, all current ERD tools find their conceptual origins in the Chen model.
- The *Crow's Foot model*, developed by C. W. Bachman, was made popular by the Knowledgeware modeling tool. You used the Crow's Foot model extensively in Chapter 4, Entity Relationship (ER) Modeling, so you are already familiar with its notation. With that in mind, you will find the following comparisons between the various notations easier to understand if you use the Crow's Foot notation as your reference point. You also saw illustrations of the Chen model in Chapter 3, The Relational Database Model. You should remember that the Chen-style connectivity designations 1 and M and cardinality designations such as (0,1), (0,N), (1,1), and (1,N) are replaced by the Crow's Foot's sticklike symbols illustrated in Figure E.1. (The name "Crow's Foot" is a reflection of the symbol used for the connectivity M, which resembles a three-toed bird's foot.) Note that the Crow's Foot model combines connectivity and cardinality information in a single symbol set. Unlike the Chen methodology, the Crow's Foot model cannot detail cardinalities other than 0, 1, or N. For example, the cardinality (5,25) cannot be shown in a Crow's Foot model. However, the commercial modeling tools that use the Crow's Foot—such as Microsoft Visio Professional—let you add those cardinalities to the diagram, using text, and to define the cardinalities in a data dictionary.
- The *Rein85 model* was developed by D. Reiner in 1985. Although the Rein85 model is based on the same modeling conventions as the Crow's Foot, its symbols are quite different. It also operates at a higher level of abstraction than the Crow's Foot. The Rein85 methodology does not recognize cardinalities explicitly, relying on connectivities to lead to logical cardinality conclusions. The Rein85 symbols are displayed in Figure E.1.
- *IDEF1X* is a derivative of the integrated computer-aided manufacturing (ICAM) studies that were conducted in the late 1970s. ICAM became the source of graphical methods for defining the functions, data structures, and dynamics of manufacturing businesses. The integration of those methods became known as IDEF (ICAM DEFi-nition). The original version of IDEF, developed by Hughes Aircraft, became known as IDEF1. The extended version of IDEF1, known as IDEF1X, became the U.S. Air Force standard and has gained acceptance as a general manufacturing data-modeling tool. As you examine Figure E.1, note that IDEF1X uses fewer symbols than the other modeling methods, thus providing fewer explicit details of the type and extent of the relationships being modeled.



Note

Figure E.1 shows Crow's Foot composite and weak entities to reflect the early implementations of that model. However, modern modeling tools, such as Microsoft Visio, do not depict the composite and weak entities. Instead, the existence of weak and composite entities is inferred from the relationship lines, which are solid when the relationship between parent and child entities is strong or identifying. In addition, the nature of the entity can be established by examining the PK/FK depictions. Therefore, the special weak/composite depictions are redundant in the Crow's Foot model.

FIGURE E.1 A COMPARISON OF ER MODELING SYMBOLS

	Chen	Crow's Foot	Rein85	IDEF1X
Entity				
Relationship line				
Relationship				
Option symbol				
One (1) symbol	1			
Many (M) symbol	M			
Composite entry				
Weak entity				

To illustrate the use of the four methods, let's examine the invoicing example discussed in Chapter 3, Section 3-7. The invoicing example is based on the following business rules:

- A CUSTOMER may generate zero INVOICES, one INVOICE, or many INVOICES. An INVOICE is generated by one CUSTOMER.
- An INVOICE refers to many PRODUCTS—for example, you can sell many hammers over some period of time, and a PRODUCT may or may not be referred to in many INVOICES. (Products that are stocked are not necessarily sold.) You should remember from Chapter 4 that the M:N relationship between INVOICE and PRODUCT is implemented through LINE, in order to decompose the M:N relationship into two 1:M relationships. Therefore, LINE becomes optional to PRODUCT because an unsold product will never appear in an invoice line.

Based on the preceding business rules, the four ERD notations are illustrated in Figures E.2 through E.5.

FIGURE E.2 THE CHEN ERD FOR THE INVOICING PROBLEM

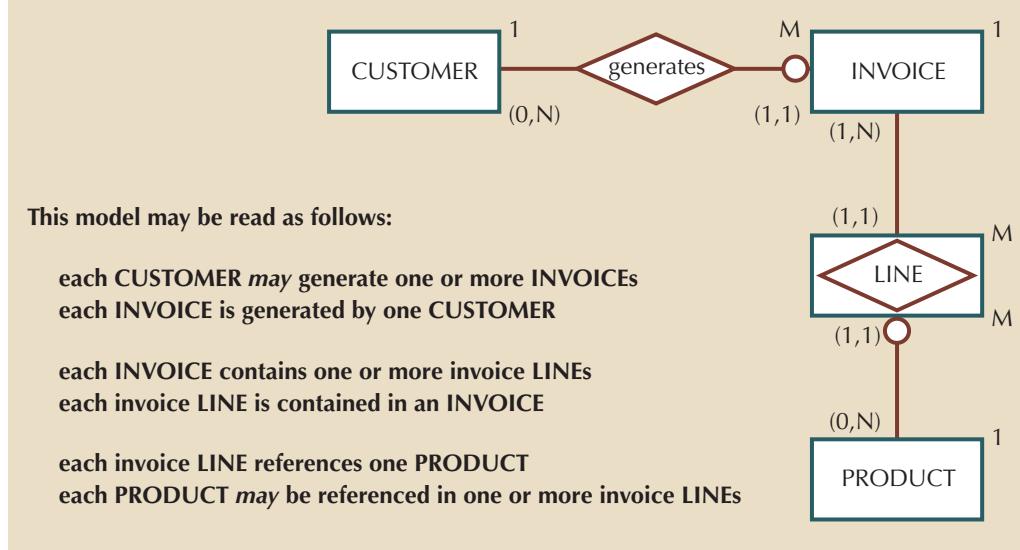


FIGURE E.3 THE CROW'S FOOT ERD FOR THE INVOICING PROBLEM

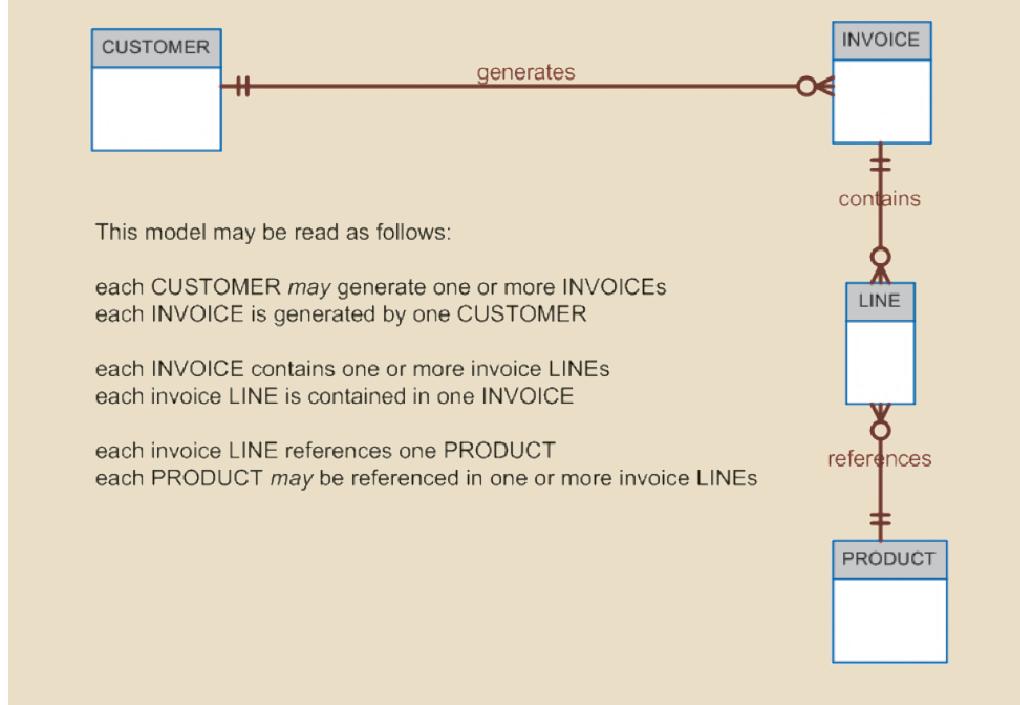


FIGURE E.4 THE REIN85 ERD FOR THE INVOICING PROBLEM

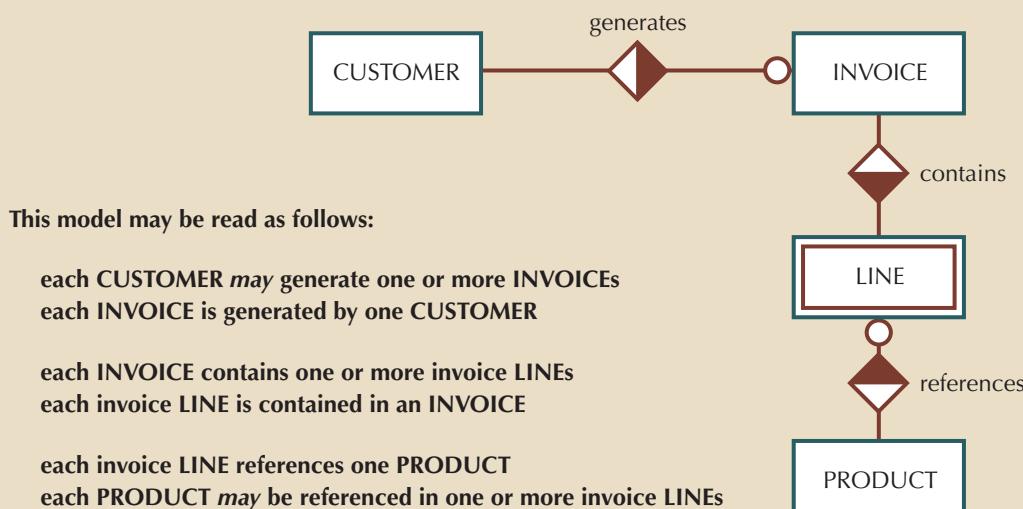
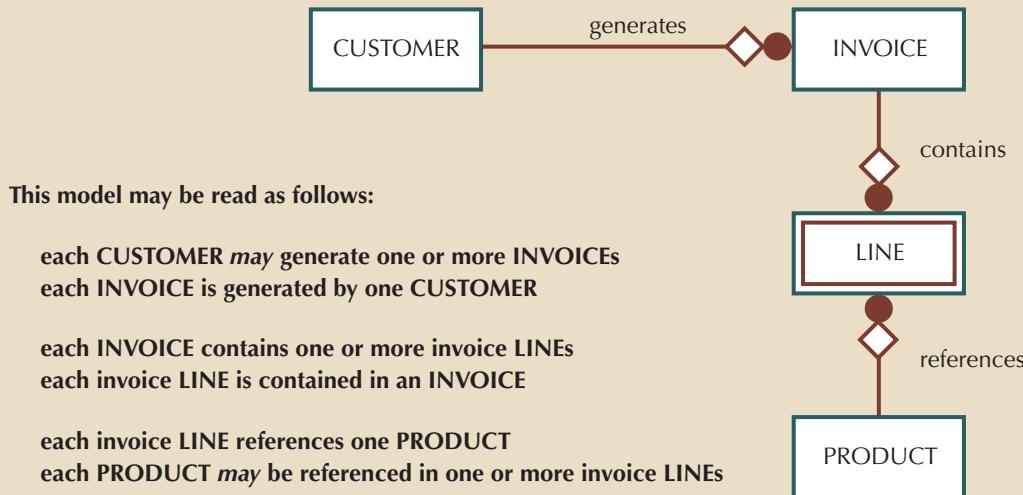


FIGURE E.5 THE IDEF1X ERD FOR THE INVOICING PROBLEM



Appendix F

Client/Server Systems

Preview

Today client/server computing is a fact of life. The Internet—along with its intranet and extranet derivatives—is perhaps the most pervasive example of client/server computing, and it has taken center stage with regard to application development. Because of the Internet’s wide reach and acceptance, you should know what client/server computing is, what its components are, how the components interact, and what effects client/server computing has on database design, implementation, and management.

Data Files and Available Formats

[MS Access](#) [Oracle](#) [MS SQL](#) [My SQL](#)

[MS Access](#) [Oracle](#) [MS SQL](#) [My SQL](#)

There are no data files for this appendix.

Data Files Available on cengagebrain.com

F-1 What Is Client/Server Computing?

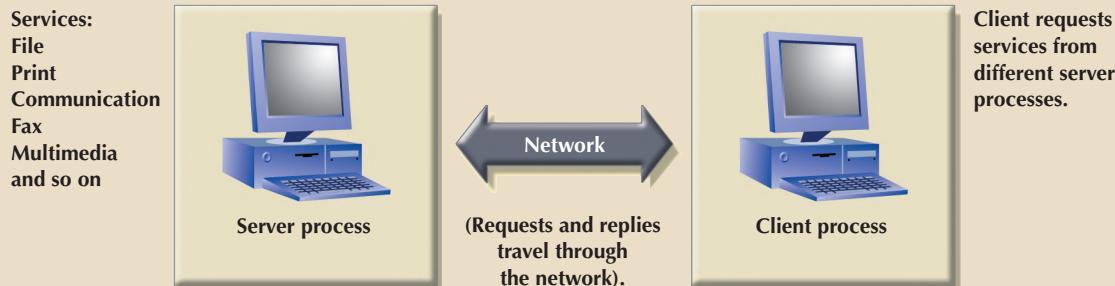
Client/server is a term used to describe a computing model for the development of computerized systems. The model is based on the distribution of functions between two types of independent and autonomous processes: servers and clients. A **client** is any process that requests specific services from server processes. A **server** is a process that provides requested services for clients. Client and server processes can reside in the same computer or in different computers connected by a network.

client
Any process that requests specific services from server processes.

server
Any process that provides requested services to clients. See *client/server architecture*.

When client and server processes reside on two or more independent computers on a network, the server can provide services for more than one client. In addition, a client can request services from several servers on the network without regard to the location or the physical characteristics of the computer in which the server process resides. The network ties the servers and clients together, providing the medium through which clients and servers communicate. (See Figure F.1.) As you examine Figure F.1, note that the services can be provided by a variety of computers on the network. For example, one computer may be dedicated to providing file and print services, another may provide communication and fax services, some may be used as web servers, and others may provide database services.

FIGURE F.1 BASIC CLIENT/SERVER COMPUTING MODEL



The key to client/server power is where the request processing takes place. For example, in a client/server database, the client requests data from the database server. The actual processing of the request (selection of the records) takes place in the database server computer. In other words, the server selects the records that match the selection criteria and sends them over the network to the client. Information processing can be divided among different types of (server) computers ranging from workstations to mainframes.

The extent of the separation of data-processing tasks is the key difference between client/server systems and mainframe systems. In mainframe systems, all processing takes place on the mainframe, and the (usually dumb) terminal is used only to display the data screens. In that environment, there is no autonomy—the terminal is simply an appendage to the mainframe. In contrast, the client/server environment provides a clear separation of server and client processes, and both processes are autonomous. The relationship between servers and clients is many-to-many; one server can provide services to many clients, and one client can request services from many servers.

Depending on the extent to which the processing is shared between the client and the server, a server or a client may be described as fat or thin. A **thin client** conducts a minimum of processing on the client side, while a **fat client** carries a relatively larger proportion of the processing load. A **fat server** carries the bulk of processing burdens,

thin client
A client that carries a relatively smaller proportion of the processing load than compared to the server, thin clients are always paired with fat servers.

fat client
A client that carries a relatively larger proportion of the processing load than compared to the server. Fat clients are always paired with thin servers.

while a **thin server** carries a lesser processing load. Thus, thin clients are associated with fat servers; conversely, fat clients are associated with thin servers.

Finally, client/server systems may also be classified as two-tier or three-tier. In a **two-tier client/server system**, a client requests services directly from the server. In a **three-tier client/server system**, the client's requests are handled by intermediate servers that coordinate the execution of the client requests with subordinate servers.

To understand why client/server computing is such a powerful player in the modern computing arena, you must examine its evolution, architecture, and functions.

F-2 The Evolution of Client/Server Information Systems

In the mid-1970s, corporate data resided safely within big, expensive mainframes that were driven by complex, proprietary operating systems. Dumb terminals, connected to front-end processors, communicated with the mainframe to produce the required information. The mainframe and its accompanying devices were jealously guarded, and access was rigorously restricted to authorized personnel. That computing style, partly dictated by available hardware and software and partly made possible by a relatively static data environment, suited the usually large companies that could afford the high cost of such computing. The centralized computing style imposed rigid control on the applications, strict limits to end-user data access, and a complex MIS department bureaucracy.

With the introduction of microcomputers in the 1980s, users were able to manipulate data locally with the help of relatively easy-to-use software such as spreadsheets and microcomputer-based database systems. However, the data on which the software operated still resided in the mainframe. Users often manually reentered the necessary data to make them accessible to the local application. This “manual download” of information was not very productive and was subject to the data anomalies discussed in Chapter 1, Database Systems. In the early 1980s, many managers’ desks were home to a dumb terminal and a PC. A substantial portion of the information game required the concurrent use of both devices. Few MIS department managers viewed the PC as a first-class citizen in their information delivery infrastructure.

The use of the PC grew steadily over the years and eventually replaced the dumb terminals on end users’ desks. Communications and terminal emulation programs allowed the PC to connect to and integrate with the MIS data center. The PCs connected to the mainframe were usually referred to as **intelligent terminals**. By this time, the electronic download of data from the mainframe to the PC was the standard way to extract required data from the mainframe to be manipulated by the local PC. Given that data access environment, the end users’ data was only a snapshot of the company’s changing mainframe data. Therefore, current mainframe data had to be downloaded frequently to avoid outdated reports or inaccurate query results.

Using their PCs, end users could create their own databases and reports, thus relying less on the MIS department’s centralized control and services. Unfortunately, that new end-user freedom caused the proliferation of snapshot versions of the corporate database. This scenario created so-called islands of information that were independent of the MIS department. Data sharing between the islands was unsophisticated. When users needed to share data, they would simply copy the data to a disk and walk to the coworker’s office, disk in hand. That data-sharing approach was later labeled the **sneakernet**.

It was no surprise that the PC’s introduction caused data security, data replication, and data integrity problems for corporate MIS departments. However, because PCs yielded many end-user information benefits, their growth could not be controlled easily.

fat server

A server that carries a relatively larger proportion of the processing load than compared to the client. Fat servers are always paired with thin clients.

thin server

A server that carries a lesser processing load than the client processes, thin servers are always paired with fat clients.

two-tier client/server system

A system within which a client requests services directly from the server. See also *client/server*.

three-tier client/server system

A system design where the client’s requests are handled by intermediate servers, which coordinate the execution of the client requests with subordinate servers. See also *client/server*.

intelligent terminals

A device that provides enhanced I/O functions to a mainframe system such as a PC connected to a mainframe computer.

sneakernet

One of the original ways to share data. When users needed to share data, they would simply copy the data to a disk and walk to the coworker’s office, disk in hand.

Consequently, to retain some semblance of control, MIS departments encouraged the development of departmental PC users' groups to share information electronically.

The new willingness to share information electronically was made possible, in large part, by a company known as Novell Data Systems, which introduced Netware/86 (originally called ShareNet) in 1983. The Novell software and hardware allowed MIS managers to connect PCs through a local area network (LAN). The LAN made it possible for end-user PCs to share files via a central PC that acted as a network file server. Netware/86 became the first widely accepted **network operating system (NOS)** for IBM personal computers and compatibles.

As PC microprocessor and data storage technology advanced rapidly in the 1990s, new PCs began to rival mainframe processing power. That trend accelerated in the late 1990s and into the first few years of the 21st century. The new PC power made it possible and, based on relative computing cost, even desirable for MIS development teams to shift much of their work to PC-based application development tools. A welcome result of that operational shift was that the end user and the MIS specialist were now working on a common PC platform. As more PCs were integrated into the corporate data centers, MIS department needs grew closer to those of the end user. As operating systems and network technology matured, even some mission-critical applications and business functions were moved from the mainframe to the PC platform.

The evolution from mainframe computing to PC-based client/server information systems generated many changes in key aspects of information management. Some of those differences are highlighted in Table F.1.

TABLE F.1

CONTRASTING MAINFRAME AND CLIENT/SERVER INFORMATION SYSTEMS

ASPECT	MAINFRAME-BASED INFORMATION SYSTEM	PC-BASED CLIENT/SERVER INFORMATION SYSTEM
Management	Centralized	Distributed/decentralized
Vendor	Single-vendor solution	Multiple-vendor solution
Hardware	Proprietary	Multiple vendors
Software	Proprietary	Multiple vendors
Security	Highly centralized	Decentralized
Data manipulation	Very limited	Extensive and very flexible
System management	Integrated	Few tools available
Application development	Overstructured Time-consuming Creation of application backlogs	Flexible Rapid application development Better productivity tools
End-user platform	Dumb terminal Character-based Single task Limited productivity	Intelligent PC Graphical user interface (GUI) Multitasking OS Better productivity tools

network operating system (NOS)

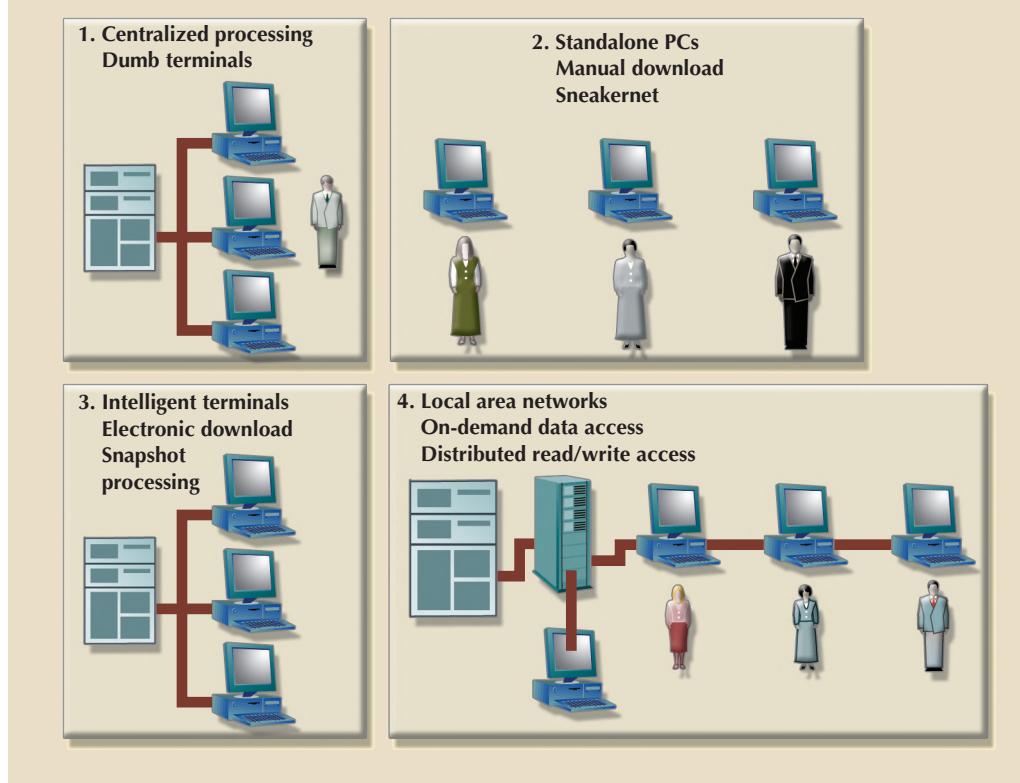
A computer operating system oriented toward providing server side services to clients (such as file and printer sharing and security management).

Figure F.2 summarizes the preceding discussion by depicting the four stages of the information systems' evolution from the mainframe to the PC-based infrastructure required for client/server computing.

The general forces behind the move to PC-based client/server computing are:

- *The changing business environment.* Businesses must meet global competitive pressures by streamlining their operations and by providing an ever-expanding array of customer services. Information management has become critical in this competitive environment, making fast, efficient, and widespread data access key to survival. The

FIGURE F.2 EVOLUTION OF THE COMPUTING ENVIRONMENT



corporate database has become a far more dynamic asset than it used to be, and it must be available at a relatively low cost.

- *The growing need for enterprise data access.* When corporations grow, especially when they grow by merging with other corporations, it is common to find a mixture of disparate data sources in their systems. In such a multiple-source data environment, managers and decision makers need fast, on-demand data access and easy-to-use tools to integrate and aggregate data. Client/server computing makes it possible to mix and match data as well as hardware. In addition, the Internet's inherent client/server structure makes it relatively easy to access both external and internal data sources.
- *The demand for end-user productivity gains, based on the efficient use of data resources.* Client/server computing supports end users' demands for better ad hoc data access and data manipulation, better user interfaces, and better computer integration.
- *Technological advances that have made client/server computing practical.* The change to PC-based information systems was also driven by advances in microprocessor technology and storage capacity, data communications and the Internet, database systems, operating systems and GUI interfaces, and sophisticated application software.
- *Growing cost/performance advantages of PC-based platforms.* PC platforms often offer unbeatable price/performance ratios compared to mainframe and minicomputer platforms. PC application costs, including acquisition, installation, and use, are usually lower than those of similar minicomputer and mainframe applications. (In complex client/server system implementations, PC-based training and support costs might be higher than those in a mainframe environment. Purchasing hardware and software from multiple sources can also become a major management headache, especially when system problems occur. Yet for many organizations, the dollar cost comparison between PC-based client/server systems and mainframe systems favors PC-based systems.)

F-3 Client/Server Architecture

The client/server infrastructure, known as the client/server architecture, is a prerequisite to the proper deployment of client/server systems. The **client/server architecture** is based on hardware and software components that interact to form a system. That system includes three main components: clients, servers, and communications middleware.

- The client is any computer process that requests services from the server. The client is also known as the **front-end application**, reflecting that the end user usually interacts with the client process.
- The server is any computer process providing services to the clients. The server is also known as the **back-end application**, reflecting that the server process provides the background services for the client process.
- The **middleware** is any computer process through which clients and servers communicate. The middleware, also known as communications middleware or the communications layer, is made up of several layers of software that aid the transmission of data and control information between clients and servers. The communications middleware is usually associated with a network. All client requests and server replies travel through the network in the form of messages that contain control information and data.

client/server architecture

The arrangement of hardware and software components to form a system composed of clients, servers, and middleware. The client/server architecture features a user of resources, or a client, and a provider of resources, or a server.

front-end application

Any process that the end user interacts with to request services from a server process.

back-end application

The process that provides service to clients.

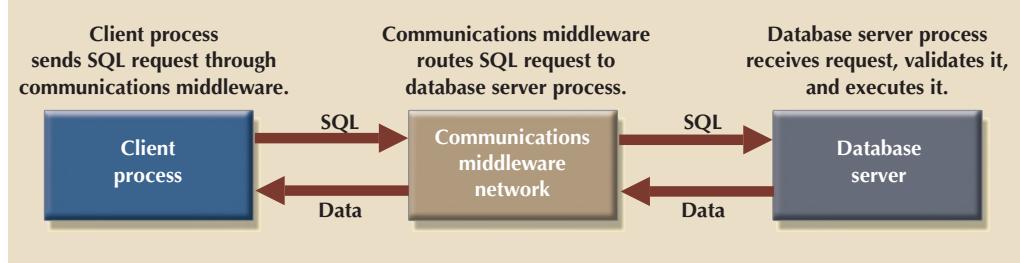
middleware

The computer software that allows clients and servers to communicate within the client/server architecture. It is used to insulate client processes from the network protocols and the details of the server process protocols.

F-3a How Client/Server Components Interact

To illustrate how the components interact, let's examine how a client requests services from a database server. Examine Figure F.3, noting that the application processing has been split into two main, independent processes: a client and a server. The communications middleware makes it possible for the client and server processes to work together. As you examine Figure F.3, also note that the communications middleware becomes the supporting platform on which clients and servers rest. Although the communications middleware is a key component in the system, its presence exacts a price by creating substantial additional overhead; adding system failure points; and, in general, adding complexity to the system's implementation.

FIGURE F.3 INTERACTION BETWEEN CLIENT/SERVER COMPONENTS



In Figure F.3, for example, the client process is in charge of the end-user interface, some portion of the local data validation, some processing logic, and data presentation. The communications middleware ensures that the messages between clients and servers are properly routed and delivered. SQL requests are handled by the database server, which validates the requests, executes them, and sends the results to the clients.

The server and client do not need to be in different computers. They can reside in the same computer and share the same processor, assuming the operating system allows it,

assuming the use of a multitasking operating system. However, most client/server implementations place the client and server processes in separate computers. Figure F.4 illustrates a client/server system with two servers and three clients.

Given the environment shown in Figure F.4, a database server process runs on an HP computer, while an **imaging server** process runs on an IBM computer. The three client processes run under three different operating systems: Windows, Linux, and Apple Mac OS. The client and server processes are connected through a network. The front-end applications in the client computers request data and images from the back-end processes (database and imaging servers). The network and supporting software form the communications middleware through which clients and servers communicate. Note that the communications can take place between clients and servers as well as between servers. Remember from Chapter 12, Distributed Database Management Systems, that this scenario is typical of distributed database environments, in which requested data can be stored in different locations.

FIGURE F.4 AN EXAMPLE OF CLIENT/SERVER ARCHITECTURE

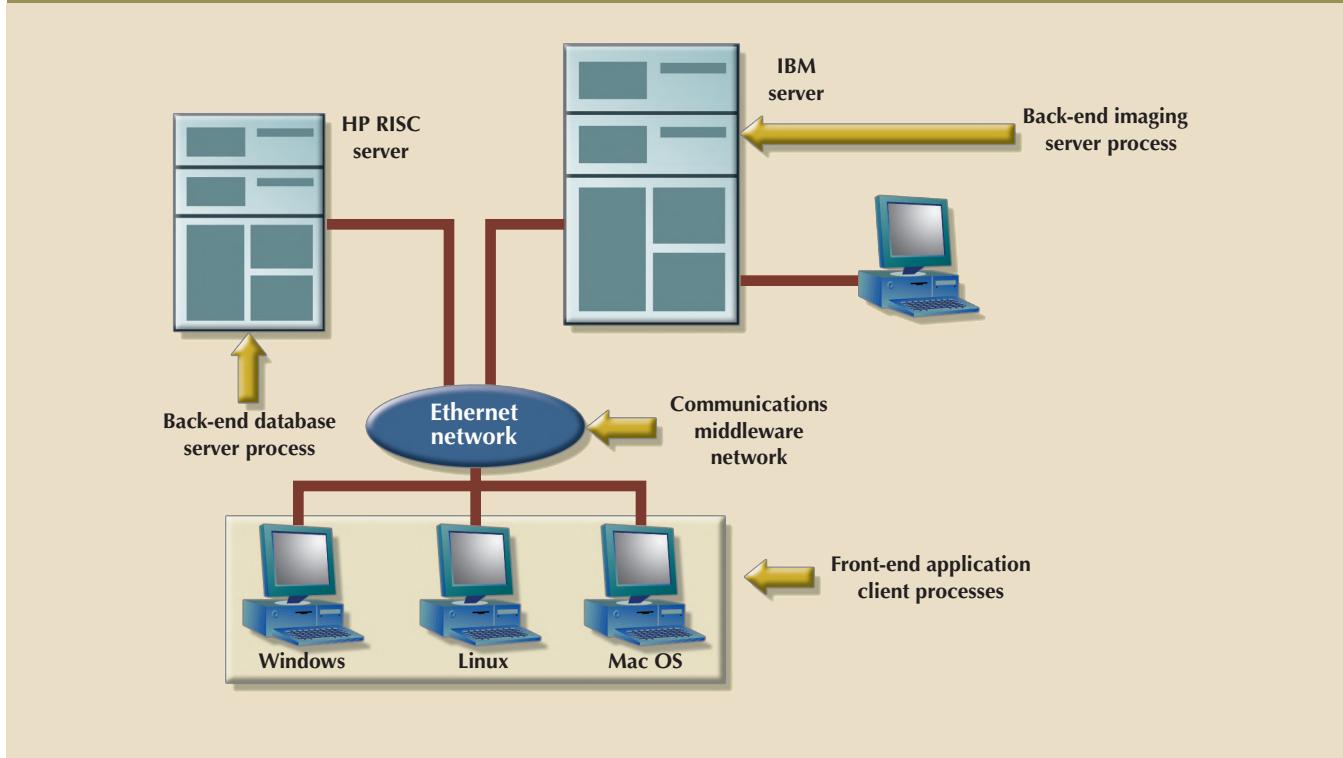


Figure F.4 illustrates a complex, yet common client/server environment in which the server processes are running under two different operating systems, the client processes are running under three different operating systems, and the system contains three different hardware platforms. In that scenario, the communications middleware (network and supporting software) becomes the integrating platform for all components. The following section examines the communications middleware components in greater detail.

F-3b Client Components

As mentioned earlier, the client is any process that requests services from a server process. The client is proactive and will, therefore, always initiate the conversation with the server. The client includes hardware and software components. Desirable client hardware and software features are:

imaging server

A process that runs on a computer and provides image management services to client computers.

- Powerful hardware
- An operating system capable of multitasking
- A graphical user interface (GUI)
- Communications capabilities

Because client processes typically require a lot of hardware resources, they should be stationed on a computer with sufficient processing power, such as a fast 64-bit processor workstation. Such processing power facilitates the creation of systems with multimedia capabilities. Multimedia systems handle multiple data types, such as voice, images, and video. Client processes also require large amounts of hard disk space and physical memory. You should have as much memory and hard disk space available as possible.

The client should have access to an operating system with at least some multitasking capabilities. Various versions of Microsoft Windows are the most common client platforms as of this writing. Windows provides access to memory, preemptive multitasking capabilities, and a graphical user interface. Those capabilities, in addition to the abundance of applications developed for the Windows interface, make Windows the platform of choice in the majority of client/server implementations. However, although the Windows operating system is popular at the client side, other operating systems—such as Microsoft Windows Server and the many “flavors” of UNIX, including Linux—are better suited to handle the client/server processing that is largely done on the server side.

To interact efficiently in a client/server environment, the client computer must be able to connect and communicate with other computers in a network environment. Therefore, the combination of hardware and operating system must also provide adequate connectivity to multiple network operating systems (NOSs). The reason for requiring a client computer to be capable of connecting with and accessing multiple network operating systems is simple: services may be located in different networks.

The client application, or front end, runs on top of the operating system and connects with the communications middleware to access services available in the network. Several third-generation programming languages (3GLs) and fourth-generation languages (4GLs) can be used to create the front-end application. Most front-end applications are GUI-based to hide the complexity of the client/server components from the end user. Figure F.5 depicts the basic client components.

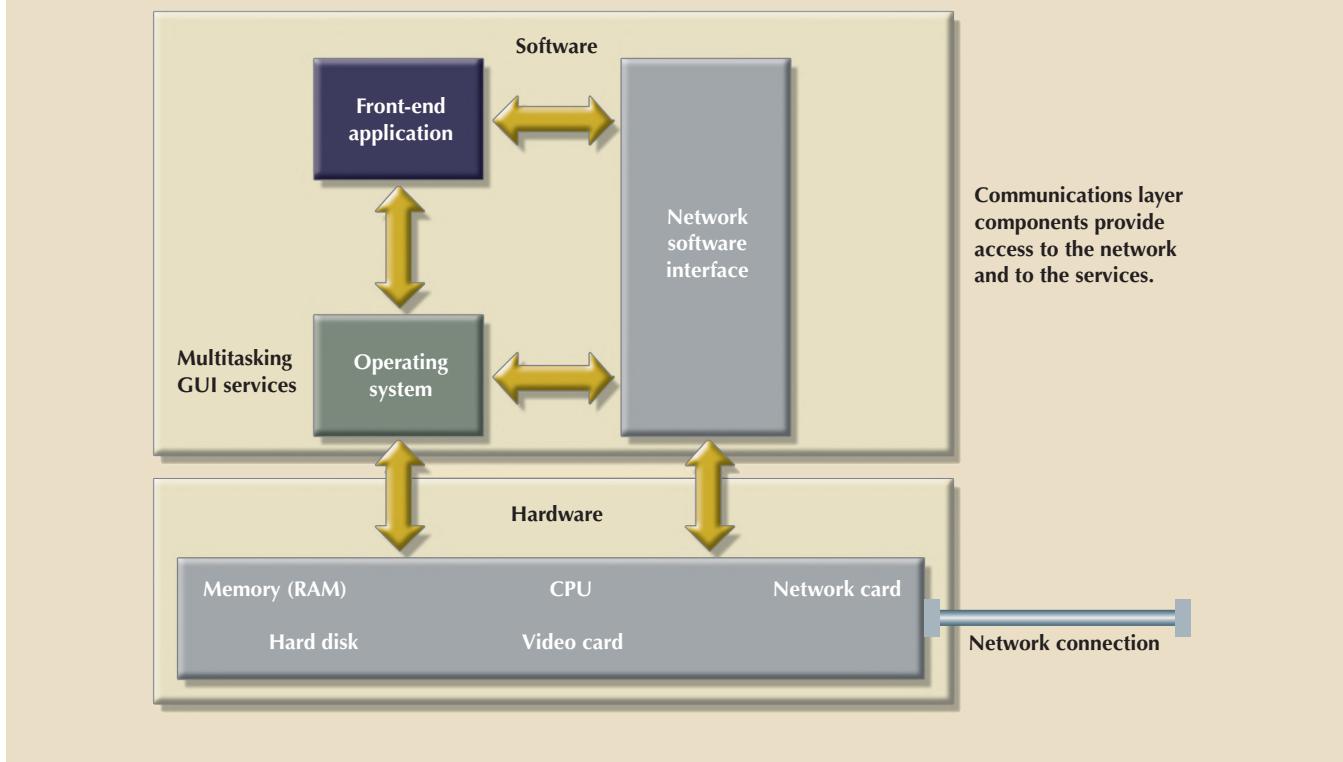
As you examine Figure F.5, note that the front-end application interacts with the operating system to access the multitasking and graphical user interface capabilities provided by the operating system. The front-end application also interacts with the network software component of the communications middleware to access the services located in the network. The hardware components of the communications middleware (network cable and network board) physically transport the requests and replies between clients and servers. While the request is being processed by the server, the client is free to perform other tasks.

F-3c Server Components

As mentioned, the server is any process that provides services to client processes. The server is reactive because it waits for the client's requests. Servers typically provide:

- *File services* for a LAN environment in which a computer with a big, fast hard disk or an array of disks is shared among different users. A client connected to the network can store files on the file server as if it were another local hard disk.
- *Print services* for a LAN environment in which a PC with one or more printers attached is shared among several clients. A client can access any one of the printers as if it were directly connected to its own computer. The data to be printed travel from the client's

FIGURE F.5 CLIENT COMPONENTS



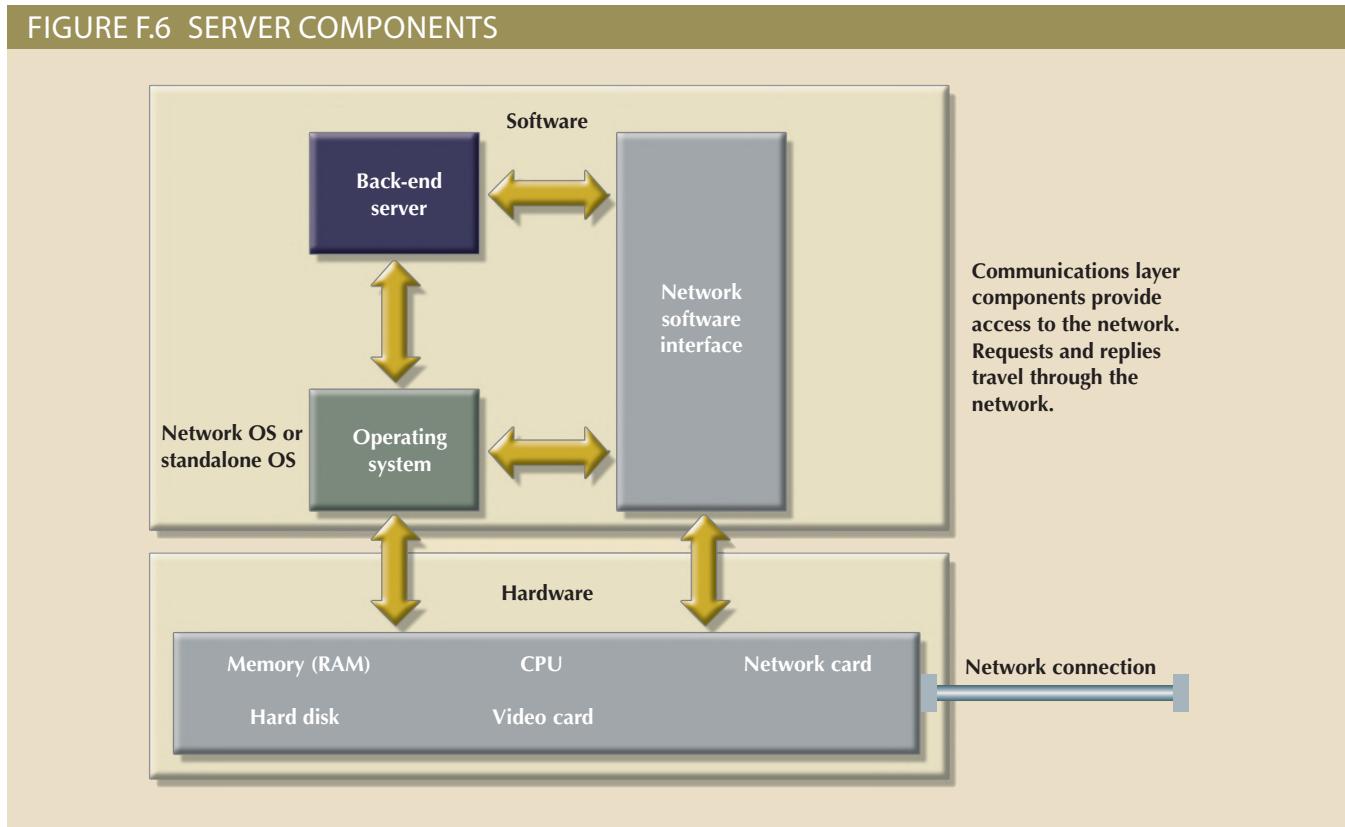
PC to the print server PC, where they are temporarily stored on the hard disk. When the client finishes sending the print job, the data are moved from the hard disk on the print server to the appropriate printer.

- *Fax services* that require at least one server equipped (internally or externally) with a fax device. The client PC need not have a fax machine or even a phone line connection. Instead, the client submits the data to be faxed to the fax server, with the required information, such as the fax number or name of the recipient. The fax server schedules the fax, dials the fax number, and transmits the fax. The fax server should also be able to handle any problems that occur in the process.
- *Communications services* that let client PCs connected to the communications server access other host computers or services to which the client is not directly connected. For example, a communications server allows a client PC to dial out to access a bulletin board or a remote LAN location.
- *Database services*, which constitute the most common and most successful client/server implementation. The client sends SQL requests to a database server. The server receives the SQL code, validates it, executes it, and sends only the results to the client. The data and the database engine are located on the database server computer. The client is required to have only the front-end application to access the database server.
- *Transaction services*, which are provided by transaction servers that are connected to the database server. A transaction server contains the database transaction code or procedures that manipulate the data in the database. A front-end application in a client computer sends a request to the transaction server to execute a specific procedure stored on the database server. No SQL code travels through the network. Transaction servers reduce network traffic and provide better performance than database servers.
- *Miscellaneous services* that include CD-ROM, DVD, video, and backup.

A common misconception is that the server process must run on the computer that contains the network operating system. This is not necessarily so. Unless circumstances dictate otherwise, separation of the server process and the NOS is highly recommended. That separation allows the server process to be located on any of the network's computers and still be available to all client computers. For example, suppose you have a CD-ROM server in a Novell NetWare network. If the server software requires the CD-ROM server process to be run on the same computer with the NetWare operating system, the host computer will be severely taxed. The host must double as a file server and a CD-ROM server. If the product does not require such "doubling," it can be installed on any PC in the network, thereby effectively distributing the workload. Both types of products use the network services (IPX or TCP/IP) provided by Novell NetWare to transport the messages between clients and the server. Each solution is subject to advantages and disadvantages. The best solution always depends on the specific circumstances.

Like the client, the server also has hardware and software components. The hardware components include the computer, CPU, memory, hard disk, video, and network card. The computer that houses the server process should be a more powerful computer than the "average" client computer because the server process must be able to handle concurrent requests from multiple clients. Server components are illustrated in Figure F.6.

FIGURE F.6 SERVER COMPONENTS



The server application, or *back end*, runs on top of the operating system and interacts with the communications middleware components to "listen" for the client's requests for services. Unlike the front-end client process, the server process need not be GUI-based. Keep in mind that the back-end application interacts with the operating system (network or standalone) to access local resources (hard disk, memory, CPU cycles, and so on). The back-end server constantly "listens" for the client's requests. Once a request

is received, the server processes it locally. The server knows how to process the request; the client tells the server only what it needs done, not how to do it. When the request is met, the answer is sent back to the client through the communications middleware.

Server hardware characteristics depend on the extent of the required services. For example, a database server for a network of 50 clients may require a computer with the following minimum characteristics:

- Fast CPU (Pentium Xeon, AMD Opteron 64-bit, or multiprocessor)
- Fault-tolerant capabilities:
 - Dual power supply to prevent power supply problems
 - Standby power supply to protect against power line failures
 - Error checking and correcting (ECC) memory to protect against memory module failures
 - Redundant array of independent disks (RAID) to protect against physical hard disk failures
- Expandability of CPU, memory, disk, and peripherals
- Bus support for multiple add-on boards
- Multiple communications options

In theory, any computer process that can be clearly divided into client and server components can be implemented through the client/server model. When properly implemented, the client/server architectural principles for process distribution are translated into the following server process benefits:

- *Location independence.* The server process can be located anywhere in the network.
- *Resource optimization.* The server process can be shared by several client processes.
- *Scalability.* The server process can be upgraded to run on more powerful platforms.
- *Interoperability and integration.* The server process should be able to work in a plug-and-play environment.

Those benefits, in addition to the hardware and software independence principles of the client/server computing model, facilitate the integration of PCs, midrange computers, and mainframes in a nearly seamless environment.

F-3d Communications Middleware Components

The communications middleware software provides the means through which clients and servers communicate to perform specific actions. In the client process, the communications middleware software also provides the specialized services that insulate the front-end applications programmer from the internal workings of the database server and network protocols. In the past, applications programmers had to write code that would directly interface with the specific database language (generally, a variation of SQL) and the specific network protocol used by the database server. If the same application were to be used with a different database and network, the application's routines had to be rewritten for the new database and network protocols. Clearly, that condition is undesirable, which is where middleware is valuable.

Although middleware can be used in different scenarios, such as email, fax, or network protocol translation, most first-generation middleware used in client/server applications

is oriented toward providing transparent data access to several database servers. The use of database middleware yields:

- *Network independence* by allowing the front-end application to access data without regard to the network protocols.
- *Database server independence* by allowing the front-end application to access data from multiple database servers without having to write code that is specific to each database server.

The use of database middleware makes it possible for the programmer to use generic SQL sentences to access different and multiple database servers. The middleware layer isolates the programmer from the differences among SQL dialects by transforming generic SQL sentences into the database server's expected syntax. For example, a problem in developing front-end systems for multiple database servers occurs because applications programmers must have in-depth knowledge of the network communications and the database access language characteristics of each database in order to access remote data. The problem is aggravated because each DBMS vendor implements its own version of SQL (with differences in syntax, additional functions, and enhancements with respect to the SQL standard). Furthermore, the data might reside in a nonrelational DBMS that doesn't support SQL, thus making it harder for the programmers to access the data. Given such cumbersome requirements, programming in client/server systems can be more difficult than programming in traditional mainframe systems. Database middleware eases the problem of accessing multiple sources of data in multiple networks and releases the programmer from the details of managing the network communications.

To accomplish its functions, the communications middleware software operates at two levels:

- The *physical* level deals with the communications between client and server computers (computer to computer). In other words, it addresses how the computers are physically linked. The physical links include the network hardware and software. The network software includes the network protocols. Recall that network protocols are the rules that govern how computers must interact with other computers in a network. They ensure that computers are able to send and receive signals to and from each other. Physically, the communications middleware is, in most cases, the network. Because the client/server model allows the client and the server to reside on the same computer, it may exist without the benefit of a computer network.
- The *logical* level deals with the communications between client and server processes (process to process), that is, with how the client and server processes communicate. The logical characteristics are governed by **interprocess communication (IPC)** protocols that give the signals meaning or purpose. It is at this level that most client/server conversation takes place.

To illustrate the two levels at which client/server communications take place, let's use an analogy. Suppose you order a pizza by phone. You start by picking up the phone, dialing the number, and waiting for someone to answer. When the phone is answered, you identify yourself, tell the clerk what type of pizza you want, how many pizzas you want, and other details. In turn, the clerk asks for your address, provides price information, and gives you an estimated delivery time. That simple transaction required both physical- and logical-level actions:

- The physical-level actions included the telephone changing your voice to analog signals and the subsequent movement of those signals through phone lines to the phone company's central PBX and from there to the phone installed at the pizza place.
- The logical-level actions were handled by you and the clerk. Because you and the clerk spoke the same language, you requested the service in a format that the clerk understood and the two of you discussed the details of the transaction successfully.

Interprocess communication (IPC)

A capability supported by various operating systems to allow two processes to communicate with each other so that applications can share data without interfering with each other.

Other than requiring that you know how to use the phone, the physical details of the phone connection are hidden. The phone company handled all of the physical details of your conversation, whereas you and the pizza clerk handled all of the logical details.

F-3e The OSI Model

Although the preceding analogy helps you understand the basic client/server interactions, you should know some details of computer communications to better understand the flow of data and control information in a client/server environment. Consider the **Open Systems Interconnection (OSI)** network reference model as an illustration of those details. That model, published in 1984, was developed by the International Organization for Standardization (ISO) in an effort to standardize the diverse network systems. The OSI model is based on seven layers, which are isolated from one another. No layer needs to know the details of another layer in order to operate. The OSI model, shown in Table F.2, was designed to let each layer provide specific services to the layer above it.

TABLE F.2

THE OSI NETWORK REFERENCE MODEL

LAYER	DESCRIPTION
Application	End-user applications program. Client: front-end application such as email or a spreadsheet. Server: back-end application such as a file server, a database server, or email.
Presentation	Provides formatting functions for Application layer protocol conversion, compression, encoding, and so on.
Session	Establishes and controls communication between applications. Ensures security, delivery, and communications recovery.
Transport	Provides error recognition and recovery, ensures that all data are properly delivered, and adds Transport-layer-specific ID.
Network	Provides end-to-end routing of packets. Splits long messages into smaller units.
Data-Link	Creates frames for transmission and controls the shared access to the network physical medium (cable). Includes error checking, correction, and so on.
Physical	Provides standards dealing with the electrical details of the transmission (network cards, cable types, voltages, and so on). Physically transmits frames of data through the cable or other media.

As you examine the OSI network reference model, note how data flow in a network. The objective of the bottom layers is to hide the network complexity from all of the layers above. In short:

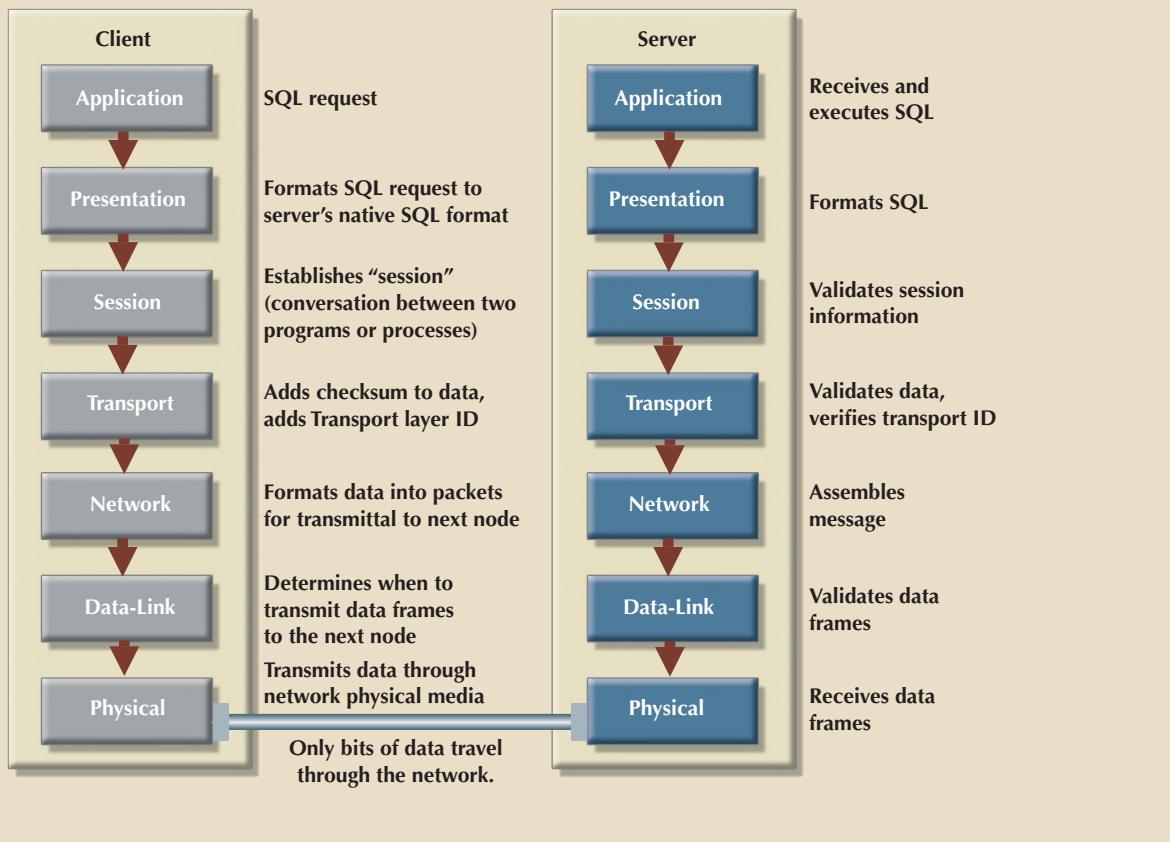
- The Application and Presentation layers provide end-user application-oriented functions.
- The Session layer ensures and controls program-to-program communications.
- The Transport, Network, Data-Link, and Physical layers provide network-oriented functions.

To better illustrate the functions contained within the OSI reference model, let's examine how a client requests services from a database server in a network. Figure F.7 depicts the flow of information through each layer of the OSI model.

Open Systems Interconnection (OSI)

A seven-layer reference model developed by the International Organization for Standardization (ISO) to help standardize diverse network systems.

FIGURE F.7 INFORMATION FLOW THROUGH THE OSI MODEL



Using Figure F.7 as a guide, you can trace the data flow.

1. The client application (*Application layer*) generates a SQL request.
2. The SQL request is sent down to the *Presentation layer*, where it is changed to a format that the SQL server engine can understand. Actions include translating ASCII characters, indicating single- and double-precision numbers, and specifying date formats (e.g., mm/dd/yyyy instead of dd/mm/yyyy).
3. The SQL request is handed down to the *Session layer*. The Session layer establishes the connection of the client process with the server process. If the database server process requires user verification, the Session layer generates the necessary messages to log on and verify the end user. At this point, usually at the beginning of the session, the end user may be required to enter a user ID and a password to access the database server, after which additional messages may be transmitted between the client and the server processes. The Session layer will identify which messages are control messages and which are data messages.
4. After the session is established and validated, the SQL request is sent to the *Transport layer*. The Transport layer generates some error validation checksums and adds some Transport-layer-specific ID information. For example, when several processes run on the client, each process may be executing a different SQL request or each process may access a different database server. The Transport layer ID helps the Transport layer identify which data correspond to which session.

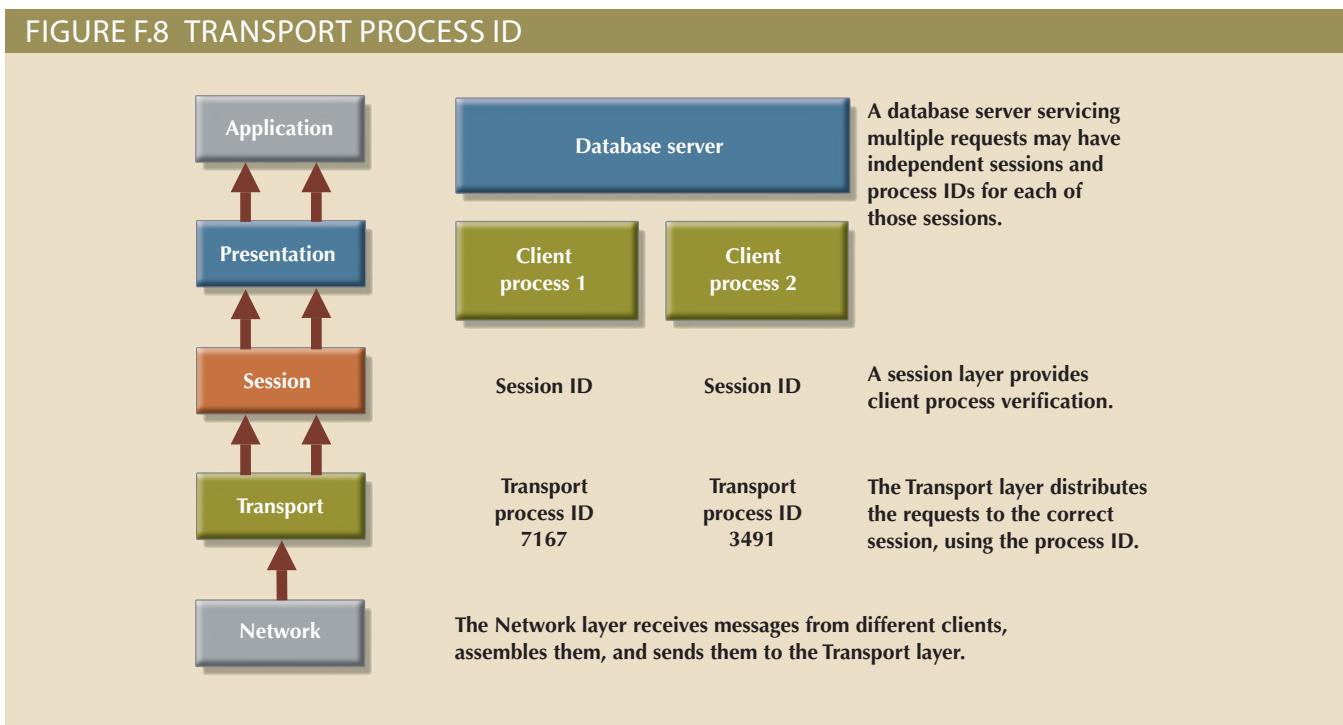
- © 2019 Cengage. May not be copied, scanned, or duplicated, in whole or in part, except for use as permitted in a license distributed with a certain product or service or otherwise on a password-protected website for classroom use.
5. Once the Transport layer has performed its functions, the SQL request is handed down to the *Network layer*. The Network layer takes the SQL request, identifies the address of the receiving node (where the server is located), adds the address of the next node in the path (if any), divides the SQL request into several smaller packets, and adds a sequence number to each packet to ensure that they are assembled in the correct order.
 6. The packet is handed to the *Data-Link layer*. The Data-Link layer adds more control information. That control information depends on the network and on which physical media are used. This information is added at the beginning (header) and at the end (trailer) of the packet, which is why the output of this process is called a **frame**. Then the Data-Link layer sends the frame to the next node. The Data-Link layer is responsible for sharing the network medium (cable) and ensuring that no frames are lost.
 7. When the Data-Link layer determines that it is safe to send a frame, it hands the frame down to the *Physical layer*, which transforms the frame into a collection of ones and zeros (bits) and then transmits the bits through the network cable. The Physical layer does not interpret the data; its only function is to transmit the signals.
 8. The signals transmitted by the Physical layer are received at the server end by its *Physical layer*, which passes the data to the *Data-Link layer*. The Data-Link layer reconstructs the bits into frames and validates them. At this point, the Data-Link layers of the client and the server computer may exchange additional messages to verify that the data were received correctly and that no retransmission is necessary. Then the Data-Link layer strips the header and trailer information from the packet and sends the packet up to the *Network layer*.
 9. The Network layer checks the packet's destination address. If the final destination is some other node in the network, the Network layer identifies it and sends the packet down to the Data-Link layer for transmission to that node. If the destination is the current node, the Network layer assembles the packets and assigns appropriate sequence numbers. Then the Network layer generates the SQL request and sends it to the *Transport layer*.
 10. The Transport layer provides additional validation checks and then routes the message to the proper session by using the transport ID. Figure F.8 illustrates how the transport process ID correctly distributes network requests for a database server process that serves multiple clients.
 11. Most of the client/server “conversation” takes place in the *Session layer*. If the communication between client and server processes is broken, the Session layer tries to reestablish the session. The Session layer identifies and validates the request and then sends it to the *Presentation layer*.
 12. The Presentation layer provides additional validation and formatting.
 13. The SQL request is sent to the database server or *Application layer*, where it is executed.

Keep in mind that although the OSI framework helps you understand network communications, it functions within a system that requires considerable infrastructure. The network protocols constitute the core of the network infrastructure because all data traveling through the network must adhere to some network protocol. In a client/server environment, it is not unusual to work with several different network protocols. Different server processes may support different network protocols to communicate over a network.

frame

Create in the Data-Link layer to add control to the information by specifying the network and physical media being used. The information is added at the beginning (header) and at the end (trailer) of a network packet to enclose (frame) the packet data.

FIGURE F.8 TRANSPORT PROCESS ID



F-3f Database Middleware Components

As depicted in Figure F.9, database middleware is divided into the following three main components:

- Application programming interface (API)
- Database translator
- Network translator

Those components (or their functions) are generally distributed among several software layers that are interchangeable in a plug-and-play fashion.

The **application programming interface (API)** is public to the client application. The programmer interacts with the middleware through the APIs provided by the middleware software. The middleware API allows the programmer to write generic SQL code instead of code specific to each database server. In other words, the middleware API allows the client process to be independent of the database server. That independence means that the server can be changed without requiring the client applications to be completely rewritten.

The **database translator** translates the SQL requests into the specific database server syntax. The database translator layer takes the generic SQL request and maps it to the database server's SQL protocol. If a database server has some nonstandard features, the database translator layer will opt to translate the generic SQL request into the specific format used by the database server. If the SQL request uses data from two different database servers, the database translator layer will take care of communicating with each server, retrieving the data using the common format expected by the client application.

The **network translator** manages the network communications protocols. Remember that database servers can use any of the network protocols discussed earlier. Therefore, if a client application taps into two databases, one that uses TCP/IP and another that uses IPX/SPX, the Network layer handles all the communications details of each database transparently to the client application. Figure F.10 illustrates the interaction between client and middleware database components.

Application programming interface (API)

Software through which programmers interact with middleware. An API allows the use of generic SQL code, thereby allowing client processes to be database server-independent.

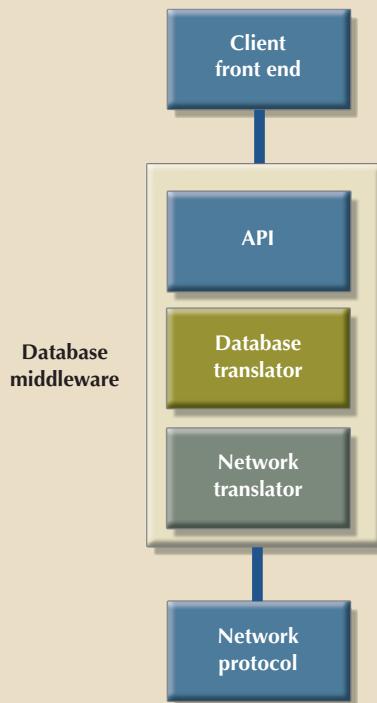
database translator

A middleware component that translates generic SQL calls into specific database server syntax to create database server independence.

network translator

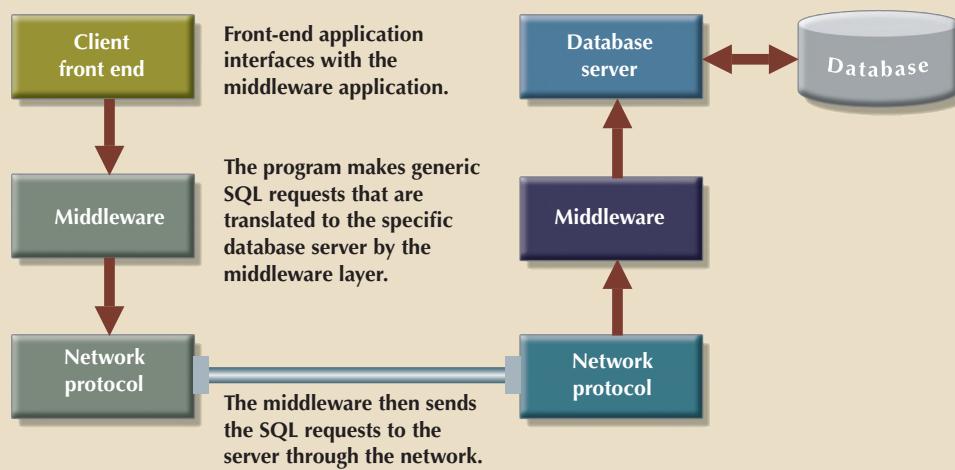
A middleware component that manages the network communications protocols.

FIGURE F.9 DATABASE MIDDLEWARE COMPONENTS



© 2019 Cengage. May not be copied, scanned, or duplicated, in whole or in part, except for use as permitted in a license distributed with a certain product or service or otherwise on a password-protected website for classroom use.

FIGURE F.10 INTERACTION BETWEEN CLIENT/SERVER MIDDLEWARE COMPONENTS



Given the existence of the three middleware components shown in Figure F.9, the three main benefits of using middleware software can be identified. Clients can:

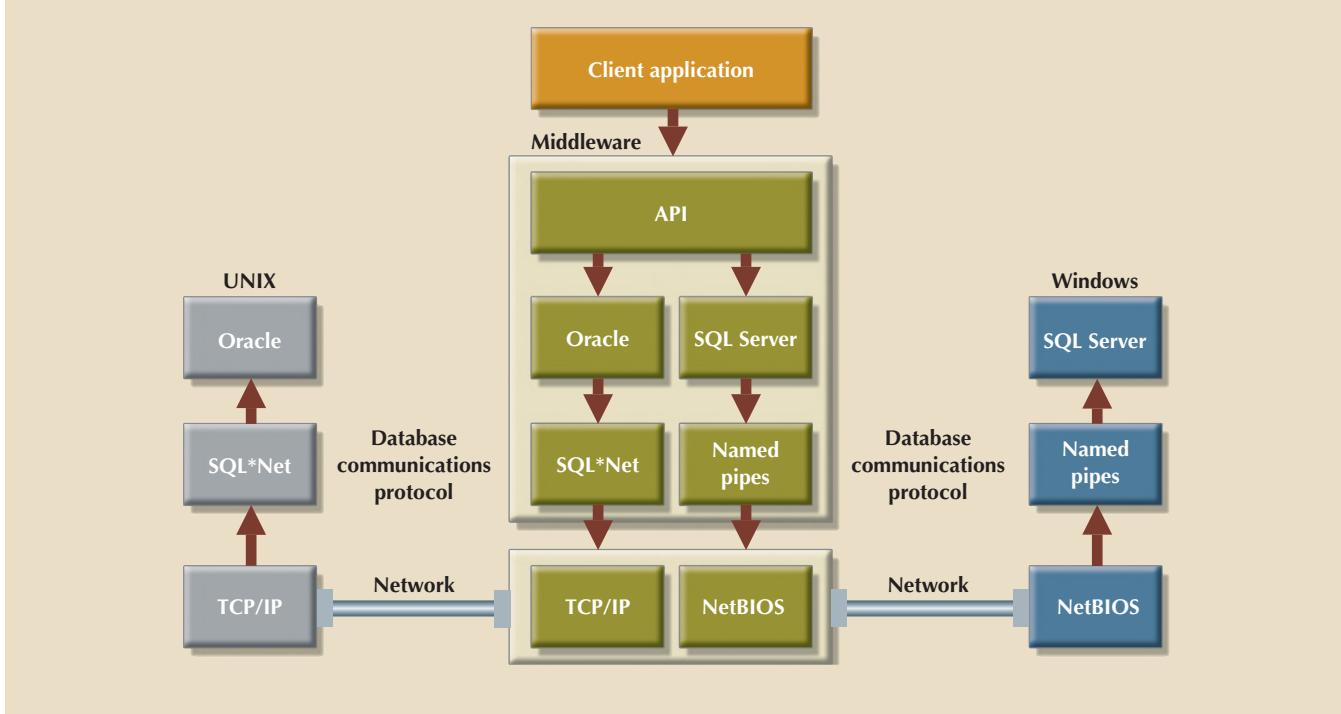
- Access multiple (and quite different) databases
- Be database-server-independent
- Be network-protocol-independent

To illustrate how all of those pieces work together, let's see how a client accesses two different database servers. Figure F.11 shows how a client application requests data from

an Oracle database server (Oracle Corporation) and from a SQL Server database server (Microsoft Corporation). The Oracle database server uses SQL*Net as its communications protocol with the client; the SQL Server database server uses Net-Library routines. SQL*Net, a proprietary solution limited to Oracle databases, is used by Oracle to send SQL requests over a network. Net-Library routines provide an interprocess communications (IPC) protocol used in SQL Server to manage client and server communications across the network.

As you examine Figure F.11, note that the Oracle server runs under the UNIX operating system and uses TCP/IP as its network protocol. The SQL server runs under the Windows Server operating system and uses NetBIOS as its network protocol. In this case, the client application uses a generic SQL query to access data in two tables: an Oracle table and a SQL Server table. The database translator layer of the middleware software contains two modules, one for each database server type to be accessed.

FIGURE F.11 MIDDLEWARE ACCESSING MULTIPLE DATABASE SERVERS



Each module handles the details of each database communications protocol. The network translator layer takes care of using the correct network protocol to access each database. When the data from the query is returned, they are presented in a format common to the client application. The end user or programmer need not be aware of the details of data retrieval from the servers. The end user might not even know where the data reside or from what type of DBMS the data was retrieved.

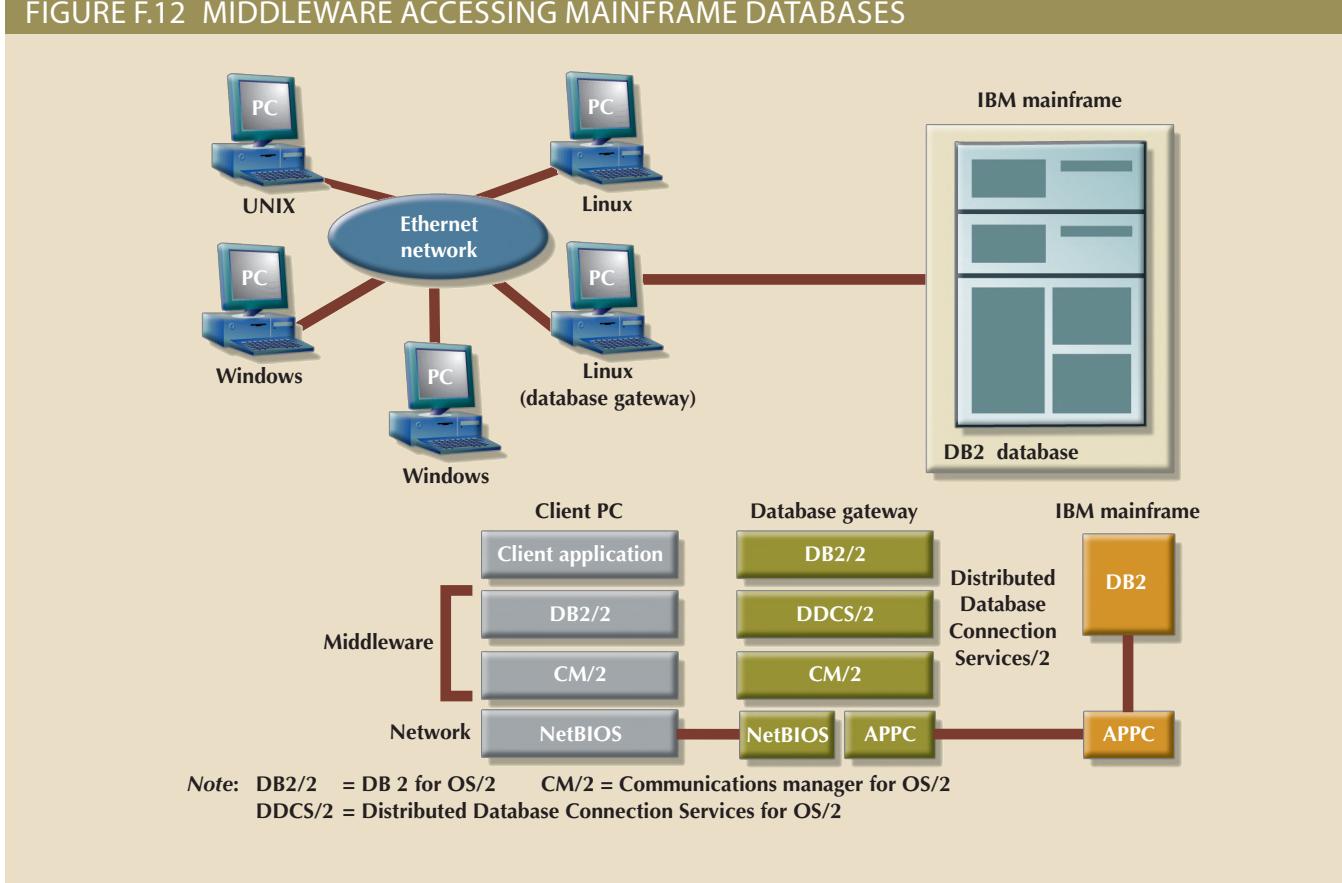
Another example of how middleware can be used to provide transparent access to databases is shown in Figure F.12. In this case, the network serves several clients that draw their data from an IBM mainframe containing a DB2 database. The clients are Windows Vista, Windows XP, and Linux computers that request data through the network.

Using the bottom of Figure F.12 as a guide, note that the mainframe DB2 database uses the Application Program-to-Program Communications (APPC) protocol to communicate with the computers in the network. A computer in the network is used to translate the TCP/IP requests of the clients into the APPC protocol needed to access

the mainframe database. This computer is known as a **gateway**. A gateway computer provides communications translation functions between dissimilar computers and networks. The term *gateway* refers to another type of middleware software; thus, a gateway computer is one that uses gateway middleware. In this case, the middleware software is installed on several computers.

Given the scenario shown in Figure F.12, the client applications request data from the IBM DB2 mainframe database. The DB2 component on the client computer performs some database translator functions. The CM component on the client computer manages the network communications in the network. The gateway computer uses the DB2, DDCS, and CM components to provide database transparency features across the network. The CM on the gateway computer translates the requests from TCP/IP to APPC and sends the requests to the DB2 mainframe database. The middleware components, residing on the client and gateway computers, work together across the network to provide database and network transparency features to all client applications.

FIGURE F.12 MIDDLEWARE ACCESSING MAINFRAME DATABASES



F-3g Middleware Classifications

Database middleware software can be classified according to the way clients and servers communicate across the network. Therefore, middleware software is usually classified as:

- Message-oriented middleware (MOM)
- Remote-procedure-call-based (RPC-based) middleware
- Object-based middleware

gateway

A type of middleware software that is used to translate client requests into the appropriate protocols needed to access specific services.

Choosing the best-suited middleware depends on the application. For example, RPC-based middleware is probably best for highly integrated systems in which data integrity is paramount, as well as for high-throughput networks. Message-oriented middleware is generally more efficient in local area networks with limited bandwidth and in applications in which data integrity is not quite as critical. Object-based middleware is an emerging type of middleware that is based on object-oriented technology. Although not as widely used as the other two, it promises better systems integration and management.

network protocol

A set of rules (at the physical level) that determines how messages between computers are sent, processed, and interpreted.

Transmission Control Protocol/Internet Protocol (TCP/IP)

The official communications protocol of the Internet, a worldwide network of heterogeneous computer systems.

Internet Packet Exchange/Sequenced Packet Exchange (IPX/SPX)

A communications protocol that determines how messages between computers are sent, interpreted and processed.

Network Basic Input/Output System (NetBIOS)

A network protocol originally developed by IBM and SYTEK Corporation in 1984.

Application Program-to-Program Communications (APPC)

A communications protocol used in IBM mainframe systems network architecture (SNA). Allows for communications between personal computers and IBM mainframe applications.

Systems Network Architecture (SNA)

A network environment used by IBM mainframe computers

F-4 Software Infrastructure: Network Protocols

A **network protocol** is a set of rules that determines how messages between computers are sent, interpreted, and processed. Network protocols enable computers to interact in a network and work at different levels of the OSI model. Other terms that are used to label the network protocols are *LAN protocol* and *network transport protocol*. The main network protocols are as follows:

- **Transmission Control Protocol/Internet Protocol (TCP/IP)** is the official communications protocol of the Internet, a worldwide network of heterogeneous computer systems. TCP/IP is the main communications protocol used by UNIX systems, is supported by most operating systems at the midrange and personal computer levels, and has become the de facto standard for heterogeneous network connections. Because UNIX is the preferred operating system for medium- and large-scale database servers, TCP/IP is an important player in the client/server arena.
- **Internetwork Packet Exchange/Sequenced Packet Exchange (IPX/SPX)** is the communications protocol developed by Novell, one of the world's leading LAN operating systems companies. The IPX/SPX protocol does not behave well when integrated into MANs (metropolitan area networks) or WANs (wide area networks), given their high levels of network traffic. That is why the latest versions of Novell operating systems have adopted TCP/IP as their default network protocol.
- **Network Basic Input/Output System (NetBIOS)** is a network protocol originally developed by IBM Corporation and Sytek in 1984 as a standard for PC applications communications. NetBIOS's limitations render it unusable in geographically dispersed internetworks. It is also perceived to be a poorer performer than the IPX/SPX protocol.
- **Application Program-to-Program Communications (APPC)** is a communications protocol used in IBM mainframe **Systems Network Architecture (SNA)** environments. This protocol allows communications between personal computers and IBM mainframe applications, such as DB2, running on the mainframe. APPC is used in IBM shops to create client/server applications that blend PCs, midrange computers such as the IBM AS/400 and RISC/6000, and mainframe systems.

The TCP/IP protocols are the leading networking protocols in use today. Mainframe network protocols are also used in many companies, especially when the company has a mainframe or minicomputer as its main data repository. As a result of the client/server computing boom, many mainframes and midrange computers are now implementing support for more open, nonproprietary network protocols, such as TCP/IP, to allow direct access from client/server PC-based front-end applications.

The network protocol you select directly affects the software products you can use. For example, an older Novell PC-based database server may be limited to supporting IPX as the network protocol. Most database servers based on UNIX and recent versions of Windows and NetWare use the TCP/IP network protocol. In companies with multiple

servers, networks, and clients, the network communications hardware (bridges, routers, and so on) must be able to translate network messages from one protocol to another.

The selection of network topology (covered later in this appendix) and protocols is a critical decision in the development of a client/server system. For commercial software developers, that decision may be market-driven because they want to sell their products to the largest markets as quickly as possible. Therefore, commercial software developers use the network protocols that provide access to the largest number of customers. A commercially developed client/server front-end or back-end application program must support multiple network protocols to communicate with different servers or clients. The network protocol decision may include additional critical variables for MIS systems developers or consultants. For example, does the company already have a network infrastructure in place? Does the company have a mainframe or a wide area network that must be integrated into the system? What type of internal expertise is available? It would not be economically feasible or efficient for commercial software developers or corporate MIS developers to create applications more than once to support multiple network protocols and multiple database systems.

F-5 Hardware Infrastructure: Cabling and Devices

The primary hardware components of network infrastructure are cabling and devices that permit and regulate network communications.

F-5a Network Cabling

Usually, cables are used to physically connect computers and to transmit data between them. There are three main types of network cabling systems: twisted pair, coaxial, and fiber-optic cable. There are also some wireless connection alternatives.

- **Twisted pair cable** is chosen for most installations because it is easy to install and carries a low price tag. Twisted pair cable resembles typical telephone cable and is formed by pairs of wires that are twisted inside a cover. The wires are classified as shielded twisted pair (STP) or unshielded twisted pair (UTP). Quality requirements of twisted pair cable depend on the intended use. Quality is classified by a system that grades the cable's quality and reliability on a scale from category 1 (lowest) to category 6 (highest). The scale reflects cable resistance to electromagnetic interference, electrical resistance, speed, and so on. STP or UTP category 5 or above is recommended for client/server system implementations.
- **Coaxial cable** uses copper cables enclosed in two layers of insulation or shielding. This cable comes in a variety of types and is similar to the cable used for cable TV. The most common varieties used in local area network installations are thicknet and thinnet. Thinnet is cheaper and easier to install than thicknet, but thicknet allows greater distances between computers.
- **Fiber-optic cable** is the most expensive option, but it offers the highest data transmission quality and allows greater distances between computers. This cable is free of electromagnetic interference because it uses laser technology to transmit signals through glass cables. Fiber-optic cable is recommended for the connection of critical network points, such as a connection between two database server computers.
- Wireless communications media, such as satellite and radio, are gaining popularity in connecting remote sites and in providing an alternative to cables in office networks.

twisted pair cable

Network cable formed by pairs of wires that are twisted inside a protective insulating cover; choice cabling for most network installations because it is easy to install and carries a low price tag. It resembles typical telephone cable.

coaxial cable

Copper cables enclosed in two layers of insulation or shielding. Often referred to as "coax." Very similar to cable used for home cable TV.

fiber-optic cable

Data transmission medium for computer networks. It used light pulses to transmit the data from node to node and allows for the highest speed of information transfer available.

These media possess great potential in replacing conventional cables in the long run. Several standards (including 802.11ac, 802.11n, 802.11b, 802.11g, 802.11a, and others) allow wireless networks to achieve high transmission speeds.

Network interface cards (NICs)

Electronic circuit board that allows computers to communicate within a network.

wireless adapter

In the case of wireless networks, this adapter, sometimes called a wireless NIC, allows a computer to communicate using a wireless network.

bridge

A device that connects similar networks. Allows computers in one network to communicate with computers in another network.

repeater

A device used in Ethernet networks to add network segments to the network and to extend its signal reach.

hub

A warehouse of data packets housed in a central location on a local area network. It contains multiple ports that copy the data in the data packets to make it accessible to selected or all segments of the network.

access point

In the case of wireless networks, this allows you to connect wireless devices to a wired or wireless network.

switch

An intelligent device that connects computers. Unlike a hub, a switch allows multiple simultaneous transmissions between two ports (computers). Therefore, switches have greater throughput and speed than regular hubs.

F-5b Network Communications Devices

Network communications devices include network interface cards (NICs), hubs, repeaters, concentrators, bridges, routers, and other devices. Those devices allow you to extend and connect networks, even dissimilar ones. They also allow you to mix different cable media within the same network. Because networks are crucial components of client/server architecture, you should know the following basic device descriptions:

- **Network interface cards (NICs)** are electronic boards that allow computers to communicate within a network. An NIC interfaces with the physical cable to send and receive signals through the cable medium. In the case of wireless networks, the **wireless adapter**, sometimes called a wireless NIC, allows a computer to communicate using a wireless network.
- A **bridge** is a device that connects similar networks. The bridge, which allows computers in one network to communicate with computers in another network, operates at the OSI model's Data-Link layer and allows two or more networks to be managed as a single logical network.
- The **repeater** is a device used in Ethernet networks to add network segments to the network and to extend the reach of the network. This device, which regenerates the signal and retransmits the signal to all segments, operates at the OSI model's Physical layer.
- A **hub** is a special repeater that allows computers to be added to a network that conforms to a star configuration. A hub will retransmit the packet through all ports (computers); therefore, only one transmission takes place at a time. In the case of wireless networks, a network **access point** allows the connection of wireless devices to a wired or wireless network.
- A **switch** is an intelligent device that connects computers. Unlike a hub, a switch allows multiple simultaneous transmissions between two ports (computers). Therefore, switches have greater throughput and speed than regular hubs.
- A **router** is an intelligent device used to connect *dissimilar* networks. Routers operate at the Network layer and allow a network to span different protocols, topologies, and cable types. A router is frequently used to divide a network into smaller subnetworks. A router also can be programmed to support specific network protocols and provide multiple functions, such as packet filtering and address blocking.
- The **concentrator** is a device that resembles a network wiring closet. It provides multiple functions, such as bridge, router, repeater, and network segmentation, in a single box. Concentrators support different network topologies, cabling, and protocols. Some also provide network management capabilities.

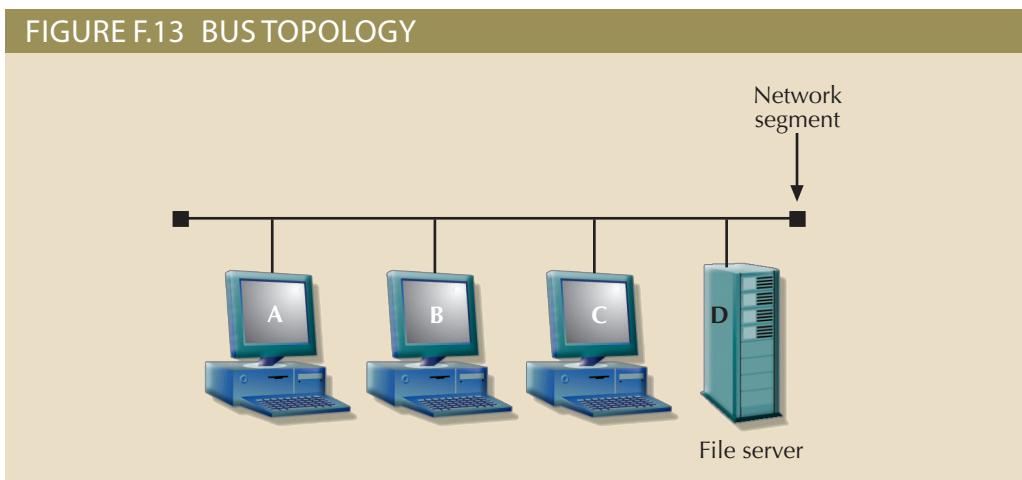
Network devices are used to extend and expand a network's "reach" and to connect existing networks with similar or dissimilar ones. The preceding list is far from exhaustive; it represents only a sampling of the most frequently used network devices. New network communications devices, designed to combine and enhance the capabilities of existing devices, appear at a dizzying rate. Keeping abreast of the new network technology is a full-time, never-ending job in the network world.

F-6 Network Topologies

The term *network topology* refers to the way data travel along the network. Network topology is closely related to the way computers are connected physically. There are three main network topologies, as follows:

- **Bus topology** requires that all computers be connected to a main network cable. In this case, messages traveling through the network are handled by all computers in the bus until they reach their final destination. For example, if A sends a message to D, the message travels through B and C before it reaches D. See Figure F.13.

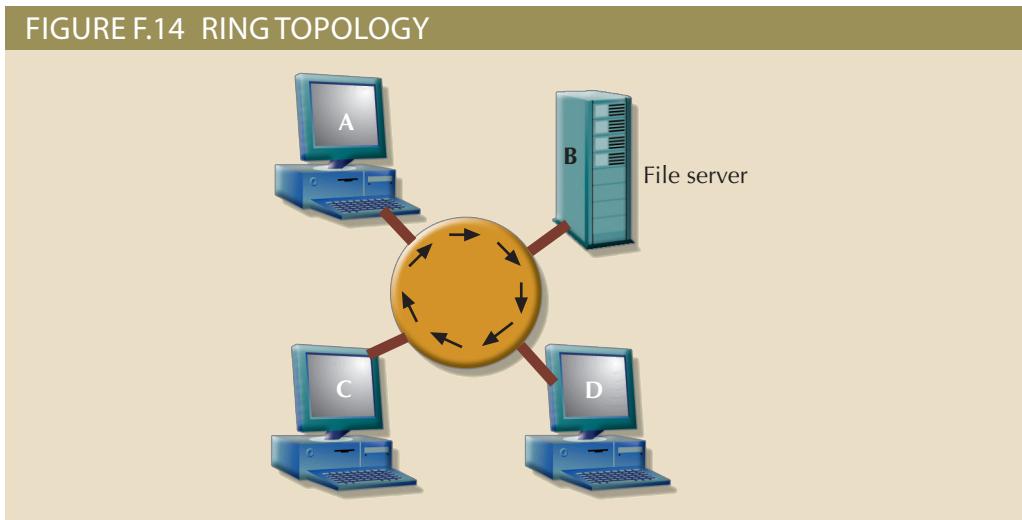
FIGURE F.13 BUS TOPOLOGY



Bus topology, usually implemented through Coaxial cable, is widely used in medium-to small-sized networks. The principal disadvantage of bus topology is that if node B or C breaks down, the entire network segment goes down. (A **network segment** is a single section of cable that connects several computers.)

- **Ring topology** computers are connected to one another through a cabling setup that, as the name implies, resembles a ring. Messages are sent from computer to computer until they reach their final destination. IBM uses the ring topology in its token ring network. The token ring implementation uses a device called a **multiple access unit (MAU)** as a wiring concentrator through which the network's computers are connected physically. Ring topology, shown in Figure F.14, is more flexible than bus topology because computers can be added to or disconnected from the ring without affecting the rest of the computers.

FIGURE F.14 RING TOPOLOGY



router

(1) An intelligent device used to connect dissimilar networks.
 (2) Hardware/software equipment that connects multiple and diverse networks.

concentrator

A device that takes multiple wires and combines them into a single method of transfer to allow multiple users to access the line simultaneously. It resembles a network wiring closet.

bus topology

Network topology requiring that all computers be connected to a main network cable. It bears the disadvantage that a single lost computer can result in network segment breakdown.

network segment

A single section of cable that connects several computers.

ring topology

Network topology in which computers are connected to one another via a cabling setup that, as the name implies, resembles a ring; it is more flexible than a bus topology, because addition or loss of a computer does not have a negative impact on other network activities.

Multiple access unit (MAU)

A wiring concentrator through which a token ring's computers are connected physically.

token

In a ring topology network, the marker that passes from computer to computer, similar to a baton in a relay race. Only the computer with the token can transmit at a given time.

star topology

Network topology with all computers connected to one another in a star configuration through a central computer or network hub. Like a ring topology, allows for computers to be added to or removed from the network without having an impact on other computers.

local area network (LAN)

A network of computers that spans a small area, such as a single building.

Ethernet

The dominant LAN standard used to interconnect computer systems. Ethernet is based on a bus or star topology that can use coaxial, twisted-pair, or fiber-optic cabling.

token ring networks

Networks that use a ring topology and token passing access control.

wireless LANs (WLANs)

Local area networks that are connected by wireless technology rather than wires.

campus-wide network (CWN)

A typical college or university network in which buildings containing LANs are connected through a network backbone.

network backbone

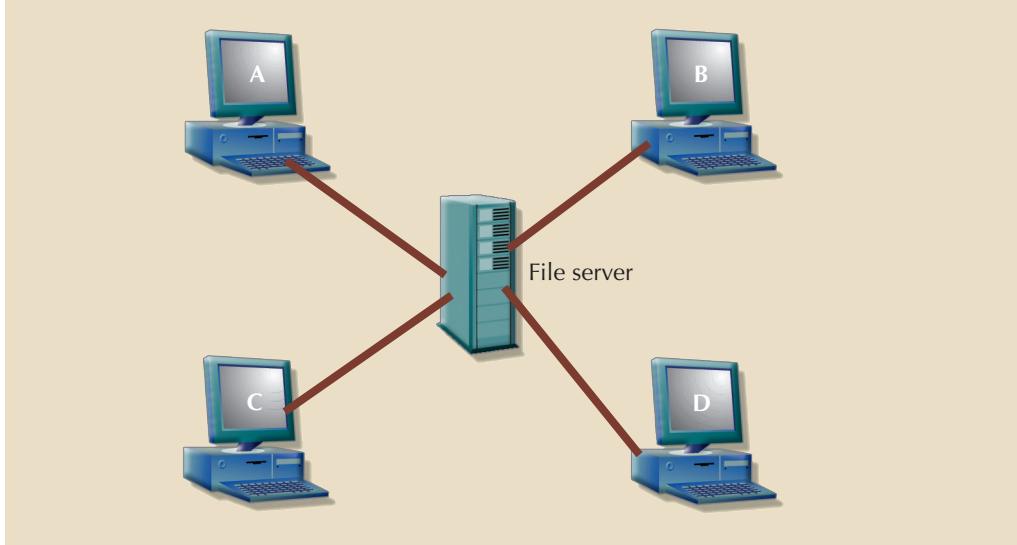
The main network cabling system for one or more local area networks.

The computers logically exchange messages by passing them along the ring. For a message to be sent from a computer in a token ring network, it must use a token. The **token**, which resembles a baton in a relay race, travels through the ring from computer to computer. Only the computer with the token can transmit at a given time.

- **Star topology** allows all computers to be connected to a central computer, as shown in Figure F.15. Like ring topology, star topology allows workstations to be added to or removed from the network without affecting the operation of the rest of the computers. Unlike ring topology, however, star topology computers are not connected to one another. Instead, they are connected to a central computer. Therefore, all network messages travel through the central computer.

The network topology is independent of the cabling system used. Any network can use coaxial, twisted pair, or fiber-optic cable.

FIGURE F.15 STAR TOPOLOGY



F-7 Network Types

Networks are usually classified by the extent of their geographical area coverage: local area, campuswide, metropolitan area, and wide area networks.

- A **local area network (LAN)** typically connects PCs in an office, a department, a floor, or a building. The LAN is the most frequently encountered network type and is preferred when workgroups are connected. There are two main LAN types: Ethernet and token ring. **Ethernet** is based on a bus or star topology that can use coaxial, twisted pair, or fiber-optic cabling. Most Ethernet LANs transfer data at a speed of 100 Mbps (one hundred million bits per second). **Token ring networks** are based on a ring topology that can use shielded twisted pair (STP), unshielded twisted pair (UTP), or fiber-optic cabling. Token ring networks can transfer data at speeds of 4 Mbps to 16 Mbps. Today, token ring networks have fallen out of favor, and are being replaced by most organizations with Ethernet networks. In addition, there are **wireless LANs (WLANs)** that can be configured according to several different standards.
- A **campuswide network (CWN)** is the typical college or university network in which buildings containing LANs and often WLANs are (usually) connected through a main network cabling system known as a **network backbone**.

- The **metropolitan area network (MAN)** is used to connect computers across a city or metropolitan area. The MAN is designed to cover much more territory than the CWN. It can even be used to connect CWNs located within a city or metropolitan area.
- A **wide area network (WAN)** is used to connect computer users across and between countries. The MAN and WAN generally make use of telephone and specialized communications companies to connect networks in sites separated by great distances.

F-8 Network Standards

Because client/server computing is focused on the *sharing* of resources, adherence to network standards is crucial. Fortunately, the Institute of Electrical and Electronics Engineers (IEEE) developed standards to provide uniformity among networks. Those IEEE standards specify the technical details that define network topology and data transmission across shared media. In addition, the IEEE standards yield the rules that govern network cabling, cable distances between computers, devices used in networks, and so on. Three important IEEE network standards are:

- IEEE 802.3:** Ethernet networks.
- IEEE 802.5:** Token ring networks.
- IEEE 802.11 and 802.16:** Wireless networks.

F-9 The Quest for Standards

Standards ensure that dissimilar computers, networks, and applications can interact to form a system. But what constitutes a standard? A *standard* is a publicly defined method to accomplish specific tasks or purposes within a given discipline or technology. Given the use of standards, it is possible to use a TV set to receive video from different broadcasters, to use a DVD player manufactured by different companies located in different countries, and so on. Standards make networks practical.

There are several organizations whose members work to establish the standards that govern specific activities. For example, the **Institute of Electrical and Electronics Engineers (IEEE)** is dedicated to defining standards for network hardware. Similarly, the **American National Standards Institute (ANSI)** has created standards for programming languages such as COBOL and SQL. The **International Organization for Standardization (ISO)** produced the Open Systems Interconnection (OSI) reference model to achieve network systems communications compatibility.

Truly universal standards for all client/server components do not yet exist. There are many different standards from which to choose. There are standards for the user interface, data access, network protocols, interprocess communications, and so on.

For example, a system might use ODBC, OLE DB, or ADO.Net database middleware. **Open Database Connectivity (ODBC)**, developed by Microsoft Corporation and the de facto standard for database middleware, is designed to provide Windows applications with an API that is independent of the data source. ODBC also provides applications programmers with a generic format for data access. A specific ODBC driver (for example, an Oracle ODBC driver or a SQL Server ODBC driver) must be used for each database being accessed. ODBC requires the database communications protocol to be present—for example, TCP/IP or SQL*Net—for communication to take place with the database server. ODBC also provides the capability to access database-specific options if

metropolitan area network (MAN)

Network type used to connect computers across a city or metropolitan area.

wide area network (WAN)

Network type used to connect computer users across distant geographical areas; generally makes use of telephone or specialized communications companies.

Institute of Electrical and Electronics Engineers (IEEE)

An organization that develops standards to provide uniformity among the technical details that define network topology and data transmission across shared media for networks.

American National Standards Institute (ANSI)

The group that accepted the DBTG recommendations and augmented database standards in 1975 through its SPARC committee.

International Organization for Standardization (ISO)

An organization formed to develop standards for diverse network systems.

Open Database Connectivity (ODBC)

Database middleware developed by Microsoft to provide a database access API to Windows applications.

they are required by the client application. Microsoft also offers OLE DB and ADO.Net as other alternatives for database connectivity.

An application that does not use a single standard can still be a client/server application. The point is to ensure that all components (server, clients, and communications middleware) are able to interact as long as they use the same standards. What really defines client/server computing is that application processing is split into client and server components.

Figure F.16 shows some of the options available to client/server systems developers. Ultimately, the objective is to have options that allow systems to interact regardless of the selection made from this list, thus producing a plug-and-play environment.

FIGURE F.16 CLIENT/SERVER OPTIONS

Client operating system and GUI	Windows, UNIX, Linux, Mac OS, and so on
Middleware	Database middleware: ODBC, OLE DB, ADO.N
Mail, database, and so on	
Network	Network protocols: TCP/IP, NetBIOS, and so on
Server services	Server OS: Windows, UNIX, Linux, and so on
Database, file, print, mail, and so on	Databases: Oracle, DB2, SQL Server, and so on
Hardware platforms	Intel Xeon, AMD Opteron, and so on

Ultimately, standards must be developed that provide systems interoperability at all levels. Recent technological advances have removed some major systems integration barriers, thus setting the stage for realizing a client/server environment that was just a dream only a few years ago: standards-based systems that function seamlessly across operating systems, graphical user interfaces, networks, and hardware platforms.

F-10 Client/Server DBMSs

A database management system (DBMS) lies at the center of most client/server systems in use today. To function properly, the client/server DBMS must be able to:

- Provide transparent data access to multiple and heterogeneous clients, regardless of the hardware, software, and network platform used by the client application.
- Allow client requests to the database server (using SQL requests) over the network.
- Process client data requests at the local server.
- Send only the SQL results to the clients over the network.

A client/server DBMS reduces network traffic because only the rows that match the query are returned. Therefore, the client computer resources are available to perform other system chores such as managing the graphical user interface. Client/server DBMSs

differ from other DBMSs in terms of where the processing takes place and what data are sent over the network to the client computer. However, client/server DBMSs do not necessarily require distributed data.

Client/server systems change the way in which data processing is approached. Data may be stored in one site or in multiple sites. When the data is stored in multiple sites, client/server databases are closely related to distributed databases. (See Chapter 12.) Distributed client/server database Management systems (DDBMSs) must have the following characteristics:

- The *location* of data is transparent to the user. The user does not need to know what the data location is, how to get there, or what protocols are used to get there.
- Data can be *accessed* and *manipulated* by the end user at any time and in many ways. Powerful applications stored on the end user's side allow access and manipulation of data in ways that had never before been available. The data request is processed on the server side; the data formatting and presentation are done on the client side.
- The *processing* of data (retrieval, storage, validation, formatting, presentation, and so on) is distributed among multiple computers.

The distinctions between client/server systems and DDBMSs are sometimes blurred. The client/server system distributes data processing among several sites. The DDBMS distributes data at different locations. In other words, client/server systems and DDBMSs involve mainly complementary functions, as well as some overlapping functions. In fact, DDBMSs use distributed processing to access data at multiple sites. Therefore, the DDBMS resembles a client/server implementation.

F-11 Client/Server Application-Processing Logic

Because the division of the application-processing logic components is a prime client/server characteristic, the two key questions that every client/server systems designer must answer are these:

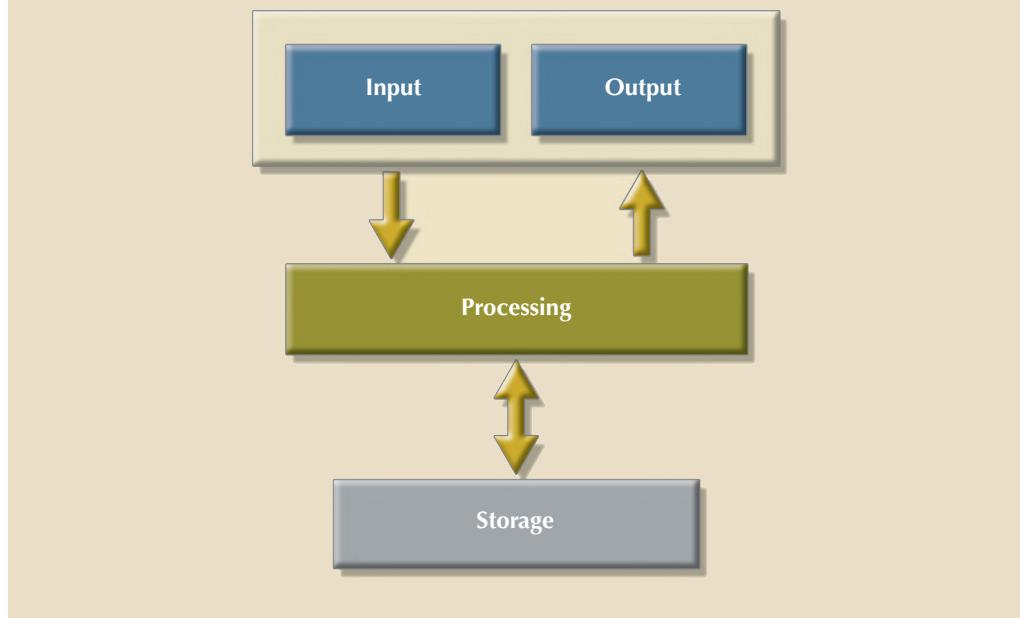
- How is the division to be made?
- Where in the system should the results of that division be placed?

To answer those questions, you must first look at application-processing logic components. (See Figure F.17.)

Figure F.17 illustrates that an application's logic can be divided into three main components: input/output, processing, and storage.

- The *input/output (I/O)* component works to format and present data in output devices such as the screen, and manages the end-user input through devices such as the keyboard. For example, the input logic shows a menu screen, waits for the end user to enter data, and then responds to the data entry. (Within this I/O component, the application uses *presentation logic* to manage the graphical user interface and data formatting.)
- The *processing* component refers to the application code that performs data validation, error checking, and so on. The processing component's logic represents the business rules and the data management logic for data retrieval and storage. For example, the processing logic "knows" that a sales transaction generates an invoice record entry, an inventory update, and a customer's accounts receivable entry. The processing logic performs several functions, including managing input and output, enforcing business rules, managing information flows within a business, and mapping the real-world

FIGURE F.17 APPLICATION LOGIC COMPONENTS



business transactions to the actual computer database. Therefore, the processing component can be further divided into three functional subcomponents, as follows:

1. *I/O processing logic* manages data entry validation and basic error checking.
 2. *Business logic* is applied through the code that represents the business rules.
 3. *Data management logic* determines which data is needed for each business transaction. For example, a sales transaction might require vendor, customer, and product data.
- The *storage component* uses *data manipulation logic* to deal with the actual data storage to and retrieval from the physical storage devices. For example, data manipulation logic is used to access the files and to check for data integrity.

In short, the three main client/server application logic components can be subdivided into the following five main functional logic components:

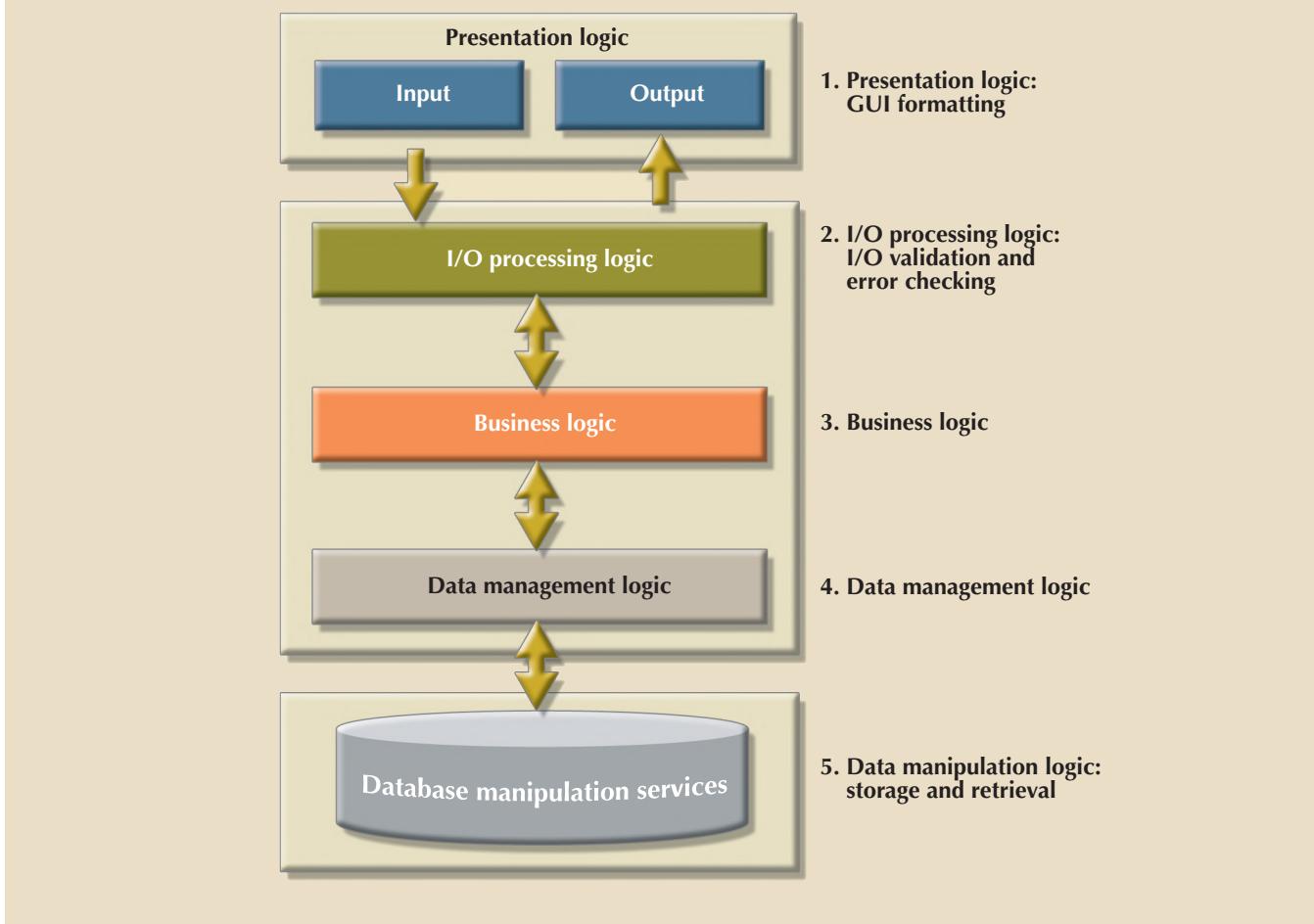
- Presentation logic
- I/O processing logic
- Business logic
- Data management logic
- Data manipulation logic

Figure F.18 shows the five functional components that form the basis for splitting the application logic processing in the client/server model.

Although there is no methodology to dictate the precise distribution of the logic components shown in Figure F.18 among clients and servers, the client/server architectural principles of process distribution (autonomy, resource maximization, scalability, and interoperability) and hardware and software independence may be used as a guide.

So where should each component be placed? With the probable exception of the presentation logic, which normally goes on the client side, each of the remaining components may be placed on the server, thus becoming a service for all of the clients. Given

FIGURE F.18 APPLICATION FUNCTIONAL LOGIC COMPONENTS



that arrangement, even the purest mainframe environment can be classified as client/server. It may even be argued that a mainframe resembles a primitive client/server incarnation in which the mainframe provides services to dumb, rather than intelligent, terminals. However, if the objective is to create a distributed environment, the pure mainframe yields few advantages compared to the naturally distributed client/server architecture, in which clients are not necessarily entirely dependent on a single server. What's more, the pure mainframe's architecture is not designed to allow the distribution of the various functional areas because the other services cannot be split out of the mainframe to be placed on other computers. (That is not, of course, referring to mainframes that are part of a client/server setup!) In fact, given a pure mainframe environment, none of the processing logic components is split: only the presentation logic can be kept on the client. The price of such architectural rigidity is high because, as it has been illustrated several times in this appendix, the greatest distributed processing benefits are obtained when the processing logic is split between server(s) and client(s).

The location combinations reflect different computing styles. For example, all components for a typical home computer are located on a single PC. The pure mainframe style reflects a condition in which only the data presentation takes place on the client side, whereas all other processing takes place on the mainframe side. It is not practical to put each component on a unique server—saving only the presentation logic for the client side—though it can be done.

Although it is possible to select any combination of logic component locations, practical considerations require that specific services such as file, print, communications, and fax be logically identified and separated, and that a decision then be made on the placement of each component. The following placement is typical:

- The *presentation logic* is always placed on the client side because it is required for end-user interaction. The GUI usually provides the services to the front-end application services.
- The *I/O processing logic* may be placed on the client side or on the server side. Although it is most commonly located on the client side in the client/server model, it may be placed on the server side when a fat server/thin client implementation exists. (Naturally, the latter scenario is the norm when the mainframe model is considered.) If a three-tier client/server system is used, the intermediate servers usually contain all of the I/O processing logic, thus making it available to all clients.
- The *business logic* can also go to either the client or the server. However, it is usually located on the client side. This logic component can also be split into client and server subcomponents. If a three-tier client/server system is used, the intermediate servers usually contain all of the business logic. Given this three-tier arrangement, changes in business logic are available to all clients.
- The *data management logic* can also be placed on either the client or the server side. However, it is normally placed on the client side or on an intermediate business logic server. The data management logic can also be split into client and server subcomponents, as is done in database middleware. Or in the case of distributed databases, the subcomponents can be placed within multiple server computers.
- The *data manipulation logic* is most commonly located on the server side. However, the data manipulation logic can also be divided among several computers in the distributed database environment.
- The split and distribution of the application-processing components are also a function of the architectural style. Figure F.19 shows the likely distribution of application-processing components within the four basic client/server architectural styles: the file server model, the database server model, the transaction server model, and the application server model.

As you examine Figure F.19, keep in mind that the server side provides services for many clients. Further, the server column represents one or more server computers. Examine Figure F.19 with the following details in mind:

- The *file server* architectural style reflects a setup in which the client does most of the processing, whereas the server side manages only the data storage and retrieval. If a client application wants to select some database table rows, the actual selection of the records takes place in the client rather than in the server.
- The data management logic is split between the client and the server computers in the *database server* architectural style. For example, the client sends a SQL request to the server and the server executes it locally, returning only the requested rows to the client. Keep in mind that there may be many servers and that a client may access many servers concurrently. If the client application executes a transaction that requires access to multiple servers, the client computer must address all of the transaction management details. Therefore, each SQL transaction must travel from the client to the server, thus increasing network traffic.

FIGURE F.19 FUNCTIONAL LOGIC SPLITTING IN FOUR CLIENT/SERVER ARCHITECTURAL STYLES

Component	File server		Database server		Transaction server		Application server	
	Client	Server	Client	Server	Client	Server	Client	Server
Presentation logic	■	■		■		■	■	
I/O processing logic	■							■
Application business logic	■			■		■		
Data management logic	■				■			
Data manipulation logic			■		■		■	■

- The *transaction server* architectural style permits the sharing of transaction details between the client and the server. For example, if the server side has some knowledge about the transaction details, some of the business logic must reside on the server. This architectural style is favored when the application transaction details are known beforehand (i.e., they are not ad hoc) and do not change very often. In this scenario, some business logic is stored on the server in the form of SQL code or some other DBMS-specific procedural language. Such stored code, usually known as stored procedures (see Chapter 8, Advanced SQL) is verified, compiled, and stored in the DBMS. The client application merely calls the stored procedure, passing it the necessary parameters for its execution. No code travels through the network, and the transaction server can be connected to many database servers.
- The *application server* architecture makes it possible to enjoy the benefits of client/server computing even when the client computers are not powerful enough to run some of the client/server applications. This architectural style allows any application to reside on a powerful computer known as the application server and then be executed and shared by many less powerful clients. In this case, all of the processing is done on the application server side, and the client computers deal just with the application output presentation. The application server architectural style is favored when it is necessary to use remote control computers over a network or when office workers are likely to require access to their office desktop PCs through their home phones.

The use of one architectural style does not preclude the use of another. In fact, it is possible to create several server “layers” by daisy chaining the server processes. For example, an application server may access a transaction server that, in turn, may access multiple database servers. It is possible to have several client/server computing styles supported concurrently within the same network. Such flexibility makes it imperative that the client/server network infrastructure be carefully planned to enable it to support diverse client/server information requirements. The client/server model’s ability to work with multiple servers is also the reason it works so well as an integrating platform on which personal computers, minicomputers, and mainframes can be brought together in a seamless fashion.



Note

The web application server represents a new computing style that integrates all of the architectural styles presented in this appendix. A web server acts like a file server; the web server transmits files to clients for execution. At the same time, the web server acts like a transaction server to coordinate database access by multiple clients. The web server is also able to provide session status control for each client as it accesses the server, effectively behaving like an application server. (The web application server model is discussed in detail in Chapter 15, Database Connectivity and Web Technologies.)

F-12 Client/Server Implementation issues

Implementing client/server systems is best described as a challenge. The development of client/server systems differs greatly in process and style from the traditional information systems development methods. For example, the systems development approach, oriented toward the centralized mainframe environment and based on traditional programming language, can hardly be expected to function well in a client/server environment that is based on hardware and software diversity. In addition, modern end users are more demanding and are likely to know more about computer technology than users did before the PC made its inroads. Therefore, MIS department managers are constantly racing the knowledge clock to assimilate new technologies that are based on multiple platforms, multiple GUIs, multiple network protocols, and so on. In addition, MIS managers must cope with rapid application development as well as the issues that arise from greater end-user autonomy in information management.

This section explores some of the managerial and technical issues involved in the development and implementation of client/server systems. Discussion begins by examining how the client/server and traditional data-processing models differ. Next, you will examine the management issues that arise from the adoption of the client/server model. Then some basic technical issues will be presented. Finally, a basic framework will be developed within which you can approach the development and implementation of client/server systems.

F-12a Client/Server versus Traditional Data Processing

You already know that the new client/server computing model environment is more complex technically than the traditional data-processing model because the former may be based on multiple platforms, operating systems, and networks. Yet the basic technical characteristics of the client/server model cannot explain why it has set the stage for another information shake-up. Instead, it is more important to note that the client/server model changes the way you look at the most fundamental data-processing issues. The client/server model's impact is far greater than the measure of its technological prowess alone.

Client/server computing expands the reach of information systems, thus changing how things are done and creating information-aware end users who will not settle for less than information autonomy. End users traditionally relied on the MIS department for information; they now create their own information by tapping into a common data pool and producing their own queries and reports to support their decision making.

This new view of the information world creates a paradox. On the one hand, end users have declared their independence from the MIS department. But on the other hand, end users have become very dependent on the client/server infrastructure (servers, networks,

middleware, and client front ends) that is managed by—you guessed it—the MIS department. Clearly, client/server computing has introduced major changes from traditional data processing.

- *From proprietary to open systems.* Traditional data-processing architecture is typically based on single-vendor solutions. Integrating multiple-vendor products within this architecture was a difficult and often impossible task. The new client/server environment demands systems that are easily integrated—systems that are open to other systems.
- *From maintenance-oriented coding to analysis, design, and service.* Given the traditionally centralized mainframe environment, most of the MIS department's focus was on application maintenance. Although client/server systems do require substantial infrastructure maintenance, the MIS department that manages such systems spends the greater portion of its time on end-user support functions. Traditional systems development life cycles dedicated most of their time to limited-use application coding and maintenance. The client/server environment changes the role of programmers by letting them use sophisticated 4GL, CASE, and other development tools to free them from coding. The price tag for using the new tools is that programmers must spend more time on systems analysis and design because errors tend to be very costly. (For one thing, end-user autonomy means that errors will be more widespread.) In short, the focus changes from coding to design.
- *From data collection to data deployment.* Instead of focusing on centralized data storage and data management, the client/server-based MIS department must concentrate on making data more easily and efficiently available to end users.
- *From a centralized to a more distributed style of data management.* Typical data management in the traditional mainframe environment was tightly structured and required rigid procedural enforcement. The new client/server environment requires a more flexible data management style. Characterized by a more decentralized management and decision-making approach, client/server computing forces a shift in focus toward the solution of end-user information problems and customer needs.
- *From a vertical, inflexible organizational style to a more horizontal, flexible organizational style.* Traditional MIS department structures will be flattened. Direct data access empowers individual users to be more information-independent from the MIS department. Consequently, the MIS department must modify or restructure its activities to accommodate people with diverse PC, GUI, and network skills.

The change in the data-processing environment brought about by client/server computing may also be evaluated by examining information systems components.

- *Hardware.* Information systems are no longer single-vendor-dependent; instead, they are likely to integrate many different hardware platforms.
- *Software.* Traditional systems consisted of procedural language routines written in a 3GL such as COBOL or FORTRAN and supported character-based applications only. All of the processing was done by the mainframe. New client/server systems are the result of the integration of many routines created by and supported by graphical user interfaces, databases, networks, and communications. The new systems split application processing into many subcomponents that integrate seamlessly. Usually, these new systems are created through the use of languages such as Visual Basic, C++, and Java.
- *Data.* Traditionally, data was centralized within a single repository. New systems tend to distribute data among multiple computers, thus putting data closer to the end user. In addition, multiple data formats (sound, images, video, text, and so on) are available.

- *Procedures.* Traditional systems were based on centralized procedures that were very rigid and complex. New distributed systems have made the procedures more flexible and decentralized.
- *People.* Client/server computing changes people's roles and functions. Updated skills are required to support and use the new technology, thus demanding intensive training and retraining to stay up to date. Such changes are not limited to the MIS department, but are spread throughout the organization.

F-12b Managerial Considerations

You have seen how client/server systems change the data-processing style and how those changes affect the organization. You will now look at some of the managerial issues that result from the introduction of client/server systems. Those issues are based on managing multiple platforms, hardware, networks, operating systems, and development environments and on dealing with multiple vendors.

- *Management and support of communications infrastructure.* One of the most complex issues in client/server environments is management of the communications infrastructure (network hardware and software). Managers must deal with several layers of network equipment to make sure that equipment from multiple vendors works together properly. The situation is especially complex because there are no comprehensively integrated client/server network management tools. Mainframe systems administrators are often afraid of the changes induced by the client/server environment because they are used to the integrated management tools that mainframe systems provide. Managers cannot count on equivalent comprehensive monitoring and management tools to support the client/server environment.
- *Management and support of applications.* Client/server applications are characterized by the distribution of processing among multiple computers. Each of those computers may be running a different operating system. An enterprise client/server system may support multiple GUIs (Windows and Apple Macintosh) on the client side, several operating systems (Novell NetWare, UNIX, or Windows Server on the server side, and Windows 10, Windows 8, Windows 7, and Apple Macintosh on the client side), and appropriate versions of middleware components. Managers must ensure that all of the components maintain current version levels at all stages: client application modules, middleware components, network components, and the back-end server side. Fortunately, there are software tools that use the network to distribute and update software automatically at the client computers. End-user support may be enhanced by creating a Help Desk to give end users a central point of support for all of their computer needs. Help Desk personnel staffing and training must be the priority of MIS management.
- *Control of escalating and hidden costs.* Client/server systems generally are expected to reduce MIS costs. Part of such cost reduction is based on the economy of scale enjoyed by the personal computer industry. (You can afford to sell complex database software for \$299 when you expect to sell 2 million copies.) Although cost reductions are expected when client/server solutions are compared to an all-mainframe alternative, there are significant startup costs. In fact, costs associated with the adaptation of resources (personnel and computer systems) to the client/server environment might be higher than expected because:
 - It may be difficult and expensive to find personnel who have the right mix of wide-ranging skills.

- Training (or retraining) of data-processing staff, managerial personnel, and end users can be time-consuming and expensive.
- The acquisition of sophisticated new hardware and software technologies is expensive.
- Establishing new procedures or adapting existing procedures to the new system can be cumbersome.

The initial costs associated with client/server computing must be treated as an investment that is likely to yield good returns through subsequent savings in new systems development, increased flexibility, and improved customer service benefits. Nevertheless, the hidden costs of maintaining and supporting the client/server environment must be carefully determined in the planning stage. (One hidden cost that managers often overlook is the “people cost” of retraining—retraining not only the data-processing personnel, but also the managers and end users. Such educational investments help minimize the culture shock associated with the freedom of information management fostered by client/server computing.) The cost of implementing client/server computing must include the following:

- *Managing people and cultural changes.* Dealing with the psychological impact of the employees’ changing roles is a never-ending task. Managers must involve the end user in the implementation of the client/server infrastructure. In the long run, the effectiveness of the system depends on whether end users put it to good use. Although the garbage-in-garbage-out (GIGO) phenomenon is encountered in any system, the wide reach of client/server computing and the power of its applications make it especially easy for users to make bigger mistakes—and make them faster. Managers also must understand that not all end users are equal. Some will be eager to learn how to use SQL or a graphical query tool to get data or to produce a report, whereas others may still be very dependent on the MIS department for the required information. The MIS department must develop and implement a gradual and progressive educational plan.
- *Managing multiple vendor relationships.* In the past, mainframe MIS managers could dial a single phone number to find solutions to hardware and software problems. In contrast, the client/server environment forces the MIS manager to deal with multiple vendors. Therefore, managers often must develop partnership-like relationships with vendors to ensure that the multiple-vendor environment works satisfactorily.

F-12c Client/Server Development Tools

In today’s rapidly changing environment, choosing the right tools to develop client/server applications is a critical decision. As a rule of thumb, managers tend to choose a tool that has long-term survival potential. However, the selection of a design or application development tool must also be driven by the system requirements. Once such requirements have been delineated, it is appropriate to determine the characteristics of the tool you want to have. Client/server tools include:

- GUI-based development
- A GUI builder that supports multiple interfaces
- Object-oriented development with support for code reusability
- A data dictionary with a central repository for data and applications
- Support for multiple databases (relational, network, hierarchical, and flat file)
- Data access regardless of data model (using SQL or native navigational access)

- Seamless access to multiple databases
- Complete systems development life cycle support from planning to implementation and maintenance
- Team development support
- Support for third-party development tools (CASE, libraries, and so on)
- Prototyping and rapid application development (RAD) capabilities
- Support for multiple platforms (operating systems, hardware, and GUIs)
- Support for middleware protocols (ODBC, IDAPI, APPC, and so on)
- Support for multiple network protocols (TCP/IP, IPX/SPX, NetBIOS, and so on)

There is no single best choice for any application development tool. For one thing, not all tools will support all GUIs, operating systems, middleware, and databases. Managers must choose a tool that fits the application development requirements and that matches the available human resources, as well as the hardware infrastructure. Chances are the system will require multiple tools to make sure that all or most of the requirements are met. Selecting the development tools is just one step. Making sure the system meets its objectives at the client, server, and network levels is another issue.

F-12d An Integrated Approach

The development of client/server systems is based on the premise that those systems are effective in helping management reach the organization's goals. Client/server-based systems should never be developed because the available tools are so technically advanced or because management wants to ride a new technology wave. Remember that client/server technology is one possible road to an objective; it is not the objective.

If a thorough study of the client/server system's technical and human dimensions indicates that its use can help achieve desired ends, a marketing plan should be developed before the client/server design and development effort is started. The objective of that plan is to build and obtain end-user and managerial support for the future client/server environment. Although there is no single recipe for the process, the overall idea is to conceptualize client/server systems in terms of their scope, optimization of resources, and managerial benefits. In short, the plan requires an integrated effort across all departments within the organization. If the client/server decision is being made for the first time, such an effort includes the following six phases:

1. *Information systems infrastructure self-study.* The objective is to determine the actual state of the available computer resources. The self-study will generate at least the following:
 - A software and hardware inventory.
 - A detailed and descriptive list of critical applications.
 - A detailed human resources (personnel and skills) inventory.
 - A detailed list of problems and opportunities.
2. *Client/server infrastructure definition.* The output of phase 1, combined with the company's computer infrastructure goals, is the input for the design of the basic client/server infrastructure blueprint. This blueprint will address the main hardware and software issues for the client, server, and networking platforms.
3. *Selection of a window of opportunity.* The next phase is to find the right system on which to base the client/server pilot project. After identifying the pilot project,

you must define it very carefully by concentrating on the problem(s), the available resources, and a set of clear and realistic goals. Describe the project in business terms rather than in technological jargon. When defining the system, make sure to plan carefully for costs. Try to balance the costs with the effective benefits of the system. Also make sure to select a pilot implementation that provides immediate and tangible benefits. A system that takes two years to develop and another three to generate tangible benefits is not acceptable.

4. *Management commitment.* Top-to-bottom commitment is essential when you are introducing new technologies that affect the entire organization. You also need managerial commitment to ensure that the necessary resources (people, hardware, software, money, and infrastructure) will be available and dedicated to the system. A common practice is to designate a person to work as a guide, or an agent of change, within the organization's departments. The main role of this person is to ease the process that changes people's roles within the organization.
5. *Implementation.* Guidelines to implementation should include at least:
 - Using “open” tools or standards-based tools.
 - Fostering continuing education in hardware, software, tools, and development principles.
 - Looking for vendors and consultants to provide vendor-specific training and implementation of designs, hardware, and application software.
6. *Review and evaluation.* Make sure that the systems conform to the criteria defined in phase 3. Continuously measure system performance as the system load increases, because typical client/server solutions tend to increase the network traffic and slow down the network. Careful network performance modeling ensures that the system performs well under heavy end-user demand conditions. Such performance modeling should be done at the server end, the client end, and the Network layer.

Key Terms

access point, F-22	concentrator, F-22	International Organization for Standardization (ISO), F-25
American National Standards Institute (ANSI), F-25	database translator, F-16	
application programming interface (API), F-16	Ethernet, F-24	
Application Program-to-Program Communications (APPc), F-20	fat client, F-2	
back-end application, F-6	fat server, F-2	
bridge, F-22	fiber-optic cable, F-21	
bus topology, F-23	frame, F-15	interprocess communication (IPC), F-12
campuswide network (CWN), F-24	front-end application, F-6	local area network (LAN), F-24
client, F-2	gateway, F-19	metropolitan area network (MAN), F-25
client/server architecture, F-6	hub, F-22	middleware, F-6
coaxial cable, F-21	imaging server, F-7	multiple access unit (MAU), F-23
	Institute of Electrical and Electronics Engineers (IEEE), F-25	network backbone, F-24
	intelligent terminals, F-3	

Network Basic Input/Output System (NetBIOS), F-20	repeater, F-22	token, F-24
network interface cards (NICs), F-22	ring topology, F-23	token ring networks, F-24
network operating system (NOS), F-4	router, F-22	Transmission Control Protocol/Internet Protocol (TCP/IP), F-20
network protocol, F-20	server, F-2	twisted pair cable, F-21
network segment, F-23	sneakernet, F-3	two-tier client/server system, F-3
network translator, F-16	star topology, F-24	wide area network (WAN), F-25
Open Database Connectivity (ODBC), F-25	switch, F-22	wireless adapter, F-22
Open Systems Interconnection (OSI), F-13	Systems Network Architecture (SNA), F-20	wireless LANs (WLANS), F-24
	thin client, F-2	
	thin server, F-3	
	three-tier client/server system, F-3	

Review Questions

1. Mainframe computing used to be the only way to manage enterprise data. Then personal computers changed the data management scene. How do those two computing styles differ, and how did the shift to PC-based computing evolve?
2. What is client/server computing, and what benefits can be expected from client/server systems?
3. Explain how client/server system components interact.
4. Describe and explain the client/server architectural principles.
5. Describe the client and the server components of the client/server computing model. Give examples of server services.
6. Using the OSI network reference model, explain the function of the communications middleware component.
7. What major network communications protocols are currently in use?
8. Explain what middleware is and what it does. Why would MIS managers be particularly interested in such software?
9. Suppose you are currently considering the purchase of a client/server DBMS. What characteristics should you look for? Why?
10. Describe and contrast the client/server computing architectural styles that were introduced in this appendix.
11. Contrast client/server data processing and traditional data processing.
12. Discuss and evaluate the following statement: There are no unusual managerial issues related to the introduction of client/server systems.

Problems

1. ROBCOR, a medium-sized company, has decided to update its computing environment. ROBCOR has been a minicomputer-based shop for several years, and all of its managerial and clerical personnel have personal computers on their desks. ROBCOR has offered you a contract to help the company move to a client/server system. Write a proposal that shows how you would implement such an environment.
2. Identify the main computing style of your university computing infrastructure. Then recommend improvements based on a client/server strategy. (You might want to talk with your department's secretary or your advisor to find out how well the current system meets their information needs.)

Appendix G

Object-Oriented Databases

Preview

Object-oriented (OO) technology draws its strength from powerful programming and modeling techniques and advanced data-handling capabilities. Because OO technology has become an important contributor to the evolution of database systems, this appendix explores the characteristics of OO systems and how those characteristics affect data modeling and design. This appendix also investigates how, through the creation of what are known as extended relational or object/relational databases, relational database vendors have responded to the demand for databases capable of handling increasingly complex data types. You will see how some of those features are implemented in Oracle.

Data Files and Available Formats

MS Access **Oracle** **MS SQL** **My SQL**

MS Access **Oracle** **MS SQL** **My SQL**

Avion_Sales	✓
RC_Stores	✓
RC_Systems	✓
RRE_Trucking	✓

Data Files Available on cengagebrain.com

G-1 Object Orientation and Its Benefits

object orientation

A set of modeling and development principles focused on an autonomous entity with embedded intelligence to interact with other objects and itself.

Object orientation is a modeling and development methodology based on object-oriented (OO) concepts. More precisely, **object orientation** is defined as a set of design and development principles based on conceptually autonomous computer structures known as objects. Each object represents a real-world entity with the ability to act upon itself and to interact with other objects.

Considering that definition, it does not require much imagination to see that using objects makes modularity almost inevitable. Object orientation concepts have been widely applied to many computer-related disciplines, especially those involving complex programming and design problems. Table G.1 summarizes some object orientation contributions to computer-related disciplines.

TABLE G.1

OBJECT ORIENTATION CONTRIBUTIONS

COMPUTER-RELATED AREA	OO CONTRIBUTIONS
Programming languages	Reduces the number of lines of code Decreases development time Enhances code reusability Makes code maintenance easier Enhances programmer productivity
Graphical user interfaces (GUIs)	Enhances ability to create easy-to-use interfaces Improves system end-user friendliness Makes it easy to define standards
Databases	Supports abstract data types Supports complex objects Supports multimedia data types
Design	Captures more of the data model's semantics Represents the real world more accurately Supports complex data manipulations in specialized applications that target graphics, imaging, mapping, financial modeling, telecommunications, geospatial applications, medical applications, and so on
Operating systems	Enhances system portability by creating layers of abstractions to handle hardware-specific issues Facilitates system extensibility through the use of inheritance and other object-oriented constructs

G-2 The Evolution of Object-Oriented Concepts

object-oriented programming (OOP)

An alternative to conventional programming methods based on object oriented concepts. It reduces programming time and lines of code, and increases programmers' productivity.

Object-oriented concepts stem from **object-oriented programming (OOP)**, which was developed as an alternative to traditional programming methods. In an OOP environment, the programmer creates or uses objects (self-contained, reusable modules that contain data as well as the procedures used to operate on the data).

OO concepts first appeared in programming languages such as Ada, ALGOL, LISP, and SIMULA. Those programming languages set the stage for the introduction of more refined OO concepts. Smalltalk, C++, and Java are popular **object-oriented programming languages (OOPLs)**. Java is used to create web applications that run on the Internet and are independent of operating systems.

OOPLs were developed to:

- Provide an easy-to-use software development environment.
- Provide a powerful software modeling tool for application development.
- Decrease development time by reducing the amount of code.
- Improve programmer productivity by making the code reusable.

OOP changes not only the way in which programs are written but also how those programs behave. In the object-oriented view of the world, each object can manipulate the data that are part of the object. In addition, each object can send messages to change the data of other objects. Consequently, the OO environment has several important attributes:

- The data set is no longer passive.
- Data and procedures are bound together, creating an object.
- The object has an innate ability to act on itself.

An object can interact with other objects to create a system. Because such objects carry their own data and code, it becomes easier to produce reusable modular systems. It is precisely that characteristic that makes OO systems seem natural to those with little programming experience, but confusing to many whose traditional programming expertise has trained them to split data and procedures. It is not surprising that OO notions became more viable with the advent of personal computers because personal computers are typically operated by end users rather than by programmers and systems design specialists.

OO programming concepts have also had an effect on most computer-based activities, including those based on databases. Because a database is designed to capture data about a business system, it can be viewed as a set of interacting objects. Each object has certain characteristics (attributes) and has relationships with other objects (methods). Given that structure, OO systems have an intuitive appeal for those doing database design and development. As database designers rather than programmers, you cannot afford to ignore the OO revolution.

G-3 Object-Oriented Concepts

Although OO concepts have their roots in programming languages, and the programmers among you will recognize basic programming elements, *you do not need to know anything about programming to understand these concepts*.

G-3a Objects: Components and Characteristics

In OO systems, everything you deal with is an object, whether it is a student, an invoice, an airplane, an employee, a service, a menu panel, or a report. Some objects are tangible, and some are not. An **object** can be defined as an abstract representation of a real-world entity that has a unique identity, embedded properties, and the ability to interact with other objects and act upon itself.

Note the difference between *object* and *entity*. An entity has data components and relationships, but lacks manipulative ability. Other differences will be identified later.

A defining characteristic of an object is its *unique identity*. To emphasize this point, let's examine the real-world objects displayed in Figure G.1. As you examine those simple objects, note that the student named J. D. Wilson has a unique (biological) identity and therefore constitutes a different object from M. R. Gonzalez or V. K. Spelling. Note also that although they share common *general* characteristics such as name, Social Security number, address, and date of birth, each object exists independently in time and space.

**object-oriented
programming
languages (OOPLs)**

A programming language based on object oriented concepts.

object

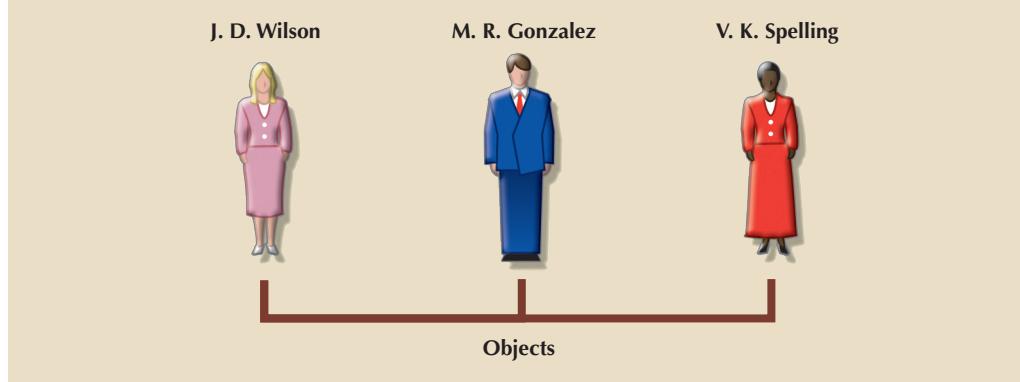
An abstract representation of a real-world entity that has a unique identity, embedded properties, and the ability to interact with other objects and itself.



Note

Current data privacy and security standards do not recommend the use of Social Security numbers as database identifiers. However, Social Security information is still stored by organizations using strict security measures such as encryption, secure access rules, and active data monitoring and protection.

FIGURE G.1 REAL-WORLD STUDENT OBJECTS



G-3b Object Identity

object ID (OID)

A system-generated object identifier that is independent of the object state and any physical address in memory.

instance variables

In the object-oriented model, another term for an attribute. See *attribute*.

base data types

A term used to describe the data types frequently used in traditional programming languages. Base data types include *real*, *integer*, and *string*.

conventional data types

See *base data types*.

domain

In data modeling, the construct used to organize and describe an attribute's set of possible values.

The object's identity is represented by an **object ID (OID)**, which is unique to the object. The OID is assigned by the system at the moment of the object's creation *and cannot be changed under any circumstances*.

Do not confuse the relational model's primary key with an OID. In contrast to the OID, a primary key is based on *user-given* values of selected attributes and can be changed at any time. The OID is assigned by the system, does not depend on the object's attribute values, and cannot be changed. *The OID can be deleted only when the object is deleted, and that OID cannot be reused*.

G-3c Attributes (Instance Variables)

Objects are described by their attributes, known as **instance variables** in an OO environment. For example, the student John D. Smith may have the attributes (instance variables) shown in Table G.2. Each attribute has a unique name and a data type associated with it. In Table G.2, the attribute names are SOCIAL_SECURITY_NUMBER, FIRST_NAME, MIDDLE_INITIAL, LAST_NAME, and so on. Traditional data types, also known as **base data types** or **conventional data types**, are used in most programming languages and include *real*, *integer*, *string*, and so on.

Attributes also have a domain. The **domain** logically groups and describes the set of all possible values that an attribute can have. For example, the possible values of SEMESTER_GPA (see Table G.2) can be represented by the real number base data type. But that does not mean that any real number is a valid GPA. Keep in mind that base data types define base domains; that is, *real* represents all real numbers, *integer* represents all integers, *date* represents all possible dates, *string* represents any combination of characters, and so on. However, base data type domains are the building blocks used to construct more restrictive *named* domains at a higher logical level. For example, to define

TABLE G.2

OBJECT ATTRIBUTES

ATTRIBUTE NAME	ATTRIBUTE VALUE
SOCIAL_SECURITY_NUMBER	414-48-0944
FIRST_NAME	John
MIDDLE_INITIAL	D
LAST_NAME	Smith
DATE_OF_BIRTH	11/23/1966
MAJOR *	Accounting
SEMESTER_GPA	2.89
OVERALL_GPA	3.01
COURSES_TAKEN *	ENG201;MATH243;HIST201;ACCT211;ECON210;ECON212; ACCT212;CIS220;ENG202;MATH301;HIST202;CIS310; ACCT343;ACCT345; ENG242;MKTG301;FIN331;ACCT355
ADVISOR*	Dr. W. R. Learned

* Represents an attribute that references one or more other objects

the domain for the GPA attribute precisely, a domain named GPA must be created. Every domain has a name and a description, including the base data type, size, format, and constraints for the domain's values. Therefore, the GPA domain can be defined as “any positive number between 0.00 and 4.00 with only two decimal places.” In this case, there is a domain name “GPA,” a base data type “real,” a constraint rule “any positive number between 0.00 and 4.00,” and a format “with only two decimal places.” The GPA domain will provide the values for the SEMESTER_GPA and OVERALL_GPA attributes. Domains can also be defined as lists of possible values separated by commas. For example the GENDER domain can be defined as “Male, Female” or as “M, F.”

It is important to note that the relational database model also supports domains. In fact, C. J. Date, one of the relational database model’s “parents,” presents domains as the way in which relational systems are able to support abstract data types, thus providing the same functionality as object-oriented databases.¹

Just as in the ER model, an object’s attribute can be *single-valued* or *multivalued*. Therefore, the object’s attribute can draw a single value or multiple values from its domain. For example, the SOCIAL_SECURITY_NUMBER attribute takes only one value from its domain because the student can have only one Social Security number. In contrast, an attribute such as LANGUAGE or HOBBY can have many values because a student might speak many languages or have many hobbies.

Object attributes may reference one or more other objects. For example, the attribute MAJOR refers to a Department object, the attribute COURSES_TAKEN refers to a list (or collection) of Course objects, and the attribute ADVISOR refers to a Professor object. At the implementation level, the OID of the referenced object is used to link both objects, thus allowing the implementation of relationships between two or more objects. Using the example in Table G.2, the MAJOR attribute contains the OID of a Department object (Accounting) and the ADVISOR attribute contains the OID of a Professor object (Dr. W. R. Learned). The COURSES_TAKEN attribute contains the OID of an object that contains a list of Course objects; such an object is known as a **collection object**.

¹ See C. J. Date and Hugh Darwen, *Foundation for Object/Relational Databases: The Third Manifesto*, Addison Wesley, 1998.

collection object

An object that contains one or more objects.



Note

Observe the difference between the relational and OO models at this point. In the relational model, a table's attribute may contain only a value used to join rows in different tables. The OO model does not need such joins to relate objects to one another.

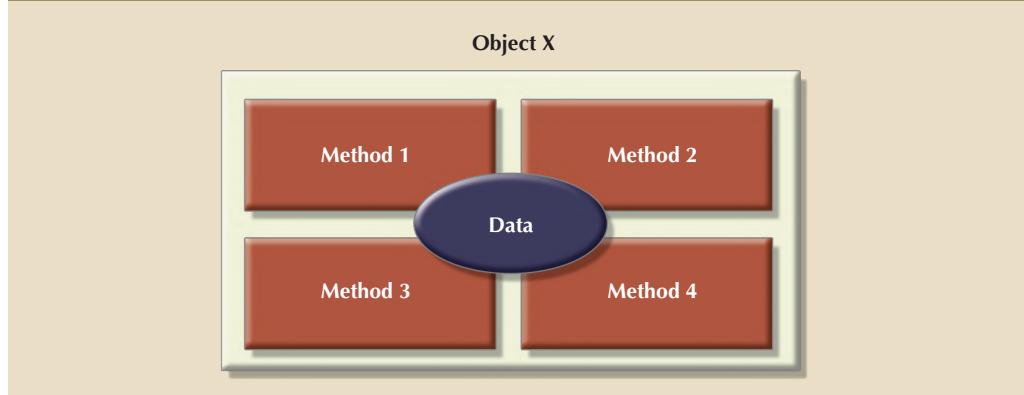
G-3d Object State

The **object state** is the set of values that the object's attributes have at any given time. Although the object's state can vary, its OID remains the same. If you want to change the object's state, you must change the values of the object's attributes. To change the object's attribute values, you must send a message to the object. This *message* will invoke a *method*.

G-3e Messages and Methods

A **method** is the code that performs a specific operation on the object's data. Methods protect data from direct and unauthorized access by other objects. To help you understand messages and methods, imagine that the object is a nutshell. The nutshell's nucleus (the nut) represents the object's data structure, and the shell represents its methods. (See Figure G.2.)

FIGURE G.2 DEPICTION OF AN OBJECT



object state

The set of values that the object's attributes have at a given time.

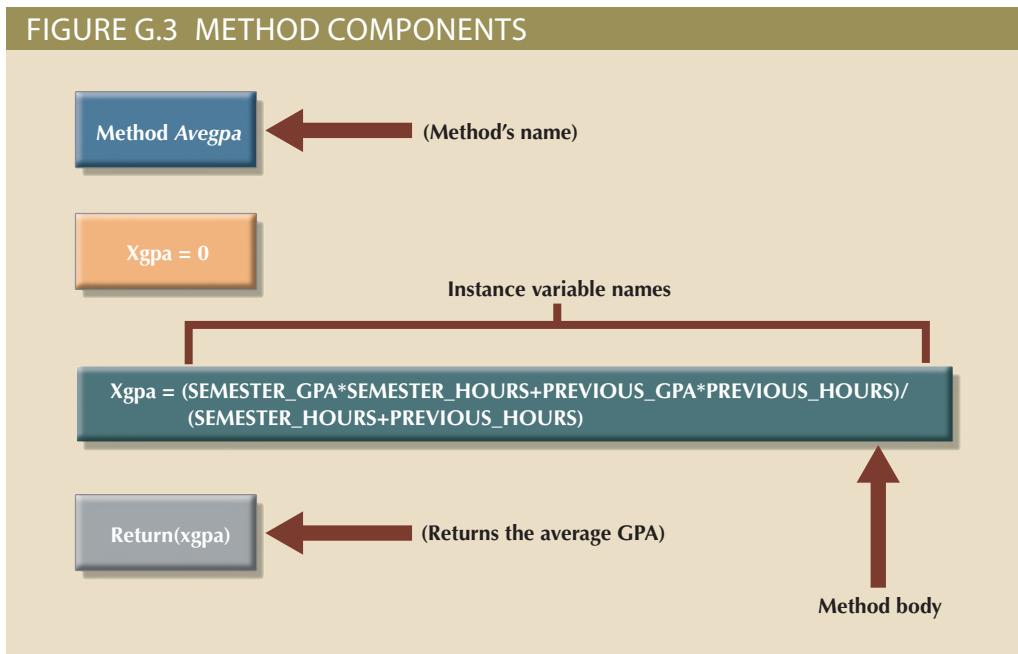
method

In the object-oriented data model, a named set of instructions to perform an action. Methods represent real-world actions, and are invoked through messages.

Every operation performed on an object must be implemented by a method. Methods are used to change the object's attribute values or to return the value of selected object attributes. Methods represent real-world actions, such as changing a student's major, adding a student to a course, or printing a student's name and address. In effect, *methods are the equivalent of procedures in traditional programming languages*. In OO terms, methods represent the object's behavior.

Every method is identified by a *name* and has a *body*. The body is composed of computer instructions written in some programming language to represent a real-world action. For example, using the object attributes described in Table G.2, you can define a method *Avegpa* that will return the average GPA of a student by using the object's attributes SEMESTER_GPA and OVERALL_GPA. Thus, the method named *Avegpa* may be represented by the transformation shown in Figure G.3.

FIGURE G.3 METHOD COMPONENTS



As you examine Figure G.3, note that the Return(xgpa) would yield $(3.2 * 15) + (3.0 * 60)/(15 + 60) = (48 + 180)/75 = 3.04$ for a student with the following characteristics:

- Current semester GPA is 3.2.
- Current class load is 15 semester hours.
- Previous GPA was 3.0 earned for a total of 60 hours.

As you examine that example, note that a method can access the instance variables (attributes) of the object for which the method is defined.

To invoke a method, you send a message to the object. A **message** is sent by specifying a receiver object, the name of the method, and any required parameters. The internal structure of the object cannot be accessed directly by the message sender, which is another object. Denial of access to the structure ensures the integrity of the object's state and hides the object's internal details. The ability to hide the object's internal details (attributes and methods) is known as **encapsulation**.

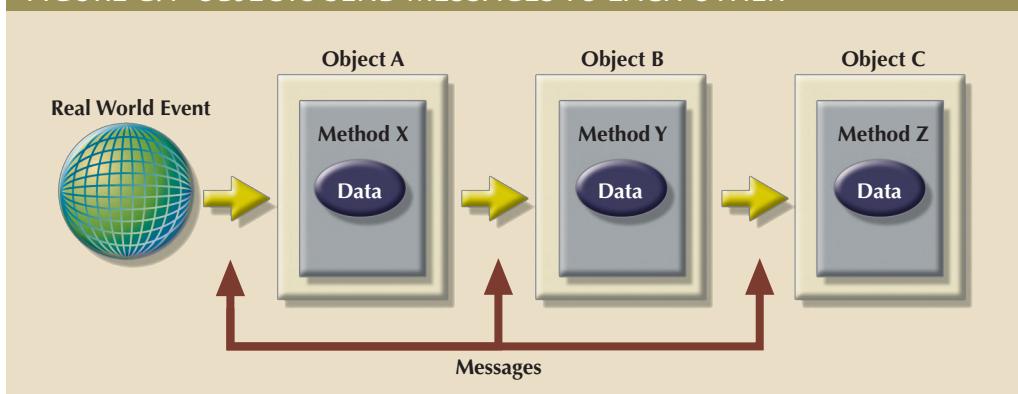
An object may also send messages to change or interrogate another object's state. (To **interrogate** means to ask for the interrogated object's instance variable value or values.) To perform such object-change and interrogation tasks, the method's body can contain references to other objects' methods (send messages to other objects), as depicted in Figure G.4.

message
In the OO data model, the name of a method sent to an object in order to perform an action. A message triggers the object's behavior. See *method*.

encapsulation
A feature by which the object can hide the internal data representation and method's implementation from external objects. Characteristic of an object oriented data model.

interrogate
To ask for the interrogated object's instance variable value or values. An object may send messages to interrogate another object's state.

FIGURE G.4 OBJECTS SEND MESSAGES TO EACH OTHER

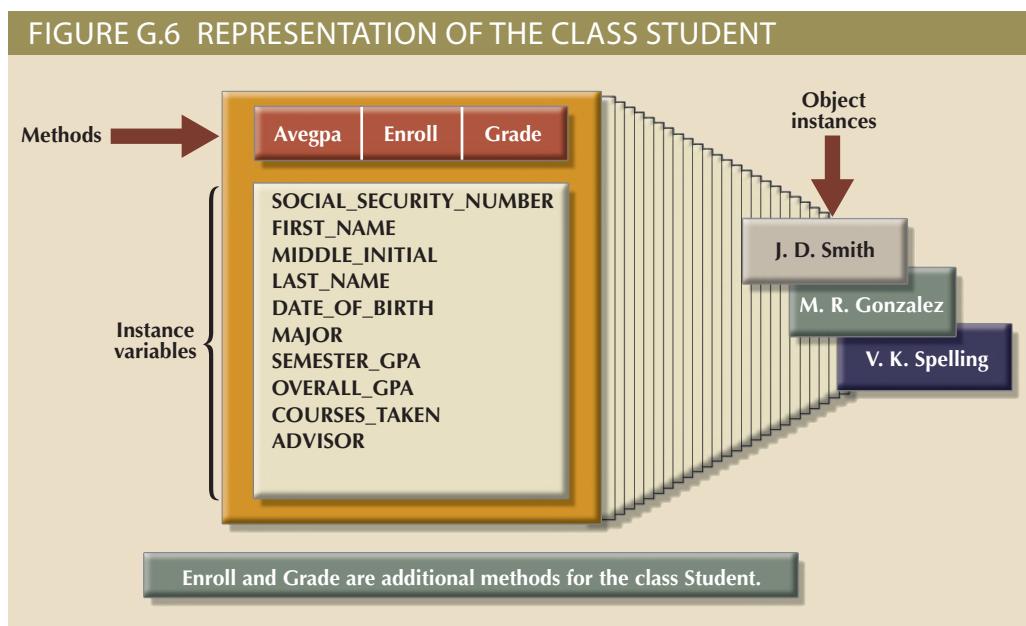
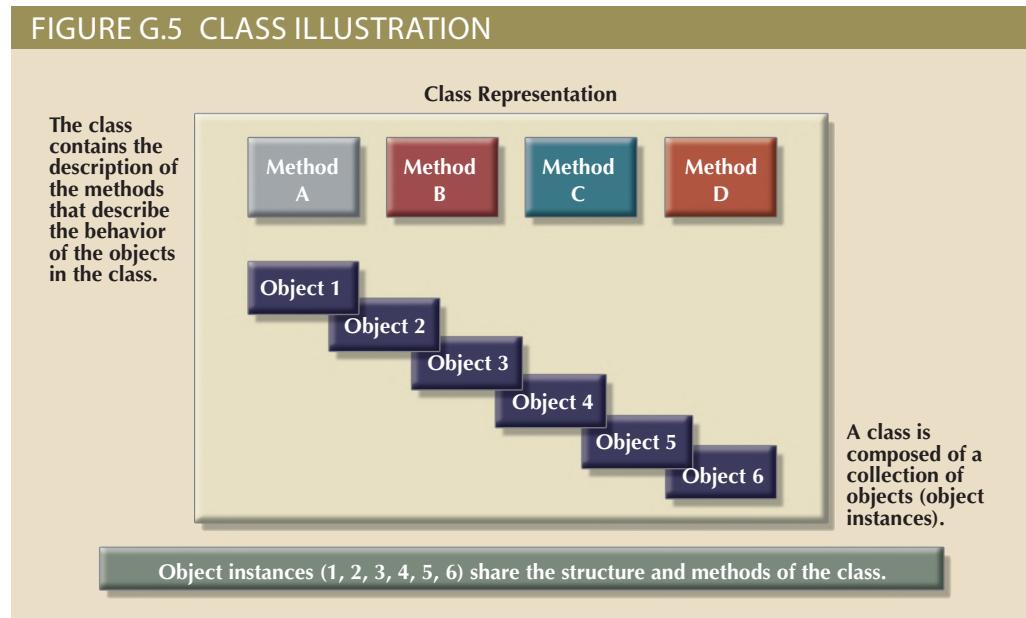


G-3f Classes

OO systems classify objects according to their similarities and differences. Objects that share common characteristics are grouped into classes. In other words, a **class** is a collection of similar objects with shared structure (attributes) and behavior (methods).

A class contains the description of the data structure and the method implementation details for the objects in the class. Therefore, all objects in a class share the same structure and respond to the same messages. In addition, a class acts as a “storage bin” for similar objects. Each object in a class is known as a **class instance** or an **object instance**. (See Figure G.5.)

Using the example shown earlier in Table G.2, a class named Student can be defined to store student objects. All objects of the class Student shown in Figure G.6 share the same structure (attributes) and respond to the same messages (implemented by methods). Note that the *Avegpa* method was defined earlier; the *Enroll* and *Grade* methods shown in Figure G.6 have been added. Each instance of a class is an object with a unique OID, and each object knows to which class it belongs.



class

A collection of similar objects with shared structure (attributes) and behavior (methods).

A class encapsulates an object's data representation and a method's implementation. Classes are organized in a class hierarchy.

class instance

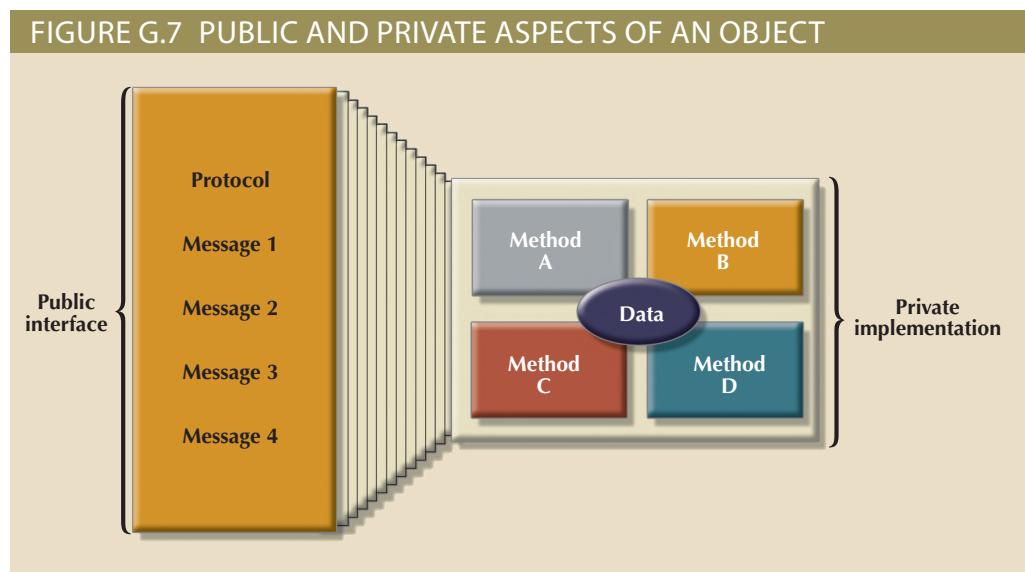
Each individual object stored in a class. Each class instance must share the same structure and respond to the same messages if they are located in the same class. Also known as *object instance*.

object instance

Each particular object belonging to a class.

G-3g Protocol

The class's collection of messages, each identified by a message name, constitutes the object or class *protocol*. The **protocol** represents an object's public aspect, that is, how it is known by other objects as well as end users. In contrast, the implementation of the object's structure and methods constitutes the object's *private aspect*. Both are illustrated in Figure G.7. For example, Figure G.6 shows three methods (*Avegpa*, *Enroll*, *Grade*) that represent the public aspect of the Student object. Because those methods are public, other objects communicate with the Student object, using any of the methods. The internal representation of the methods (see Figure G.3) yields the private aspect of the object. The private aspect of an object is not available for use by other objects.



Usually, a message is sent to an object instance. However, it is also possible to send a message to the class rather than to the object. When the receiver object is a class, the message will invoke a *class method*. One example of a class method is *new*. The *new* class method creates a new object instance (with a unique OID) in the receiver class. Because the object does not exist yet, the message *new* is sent to the class and not to the object.

The preceding discussions have laid the foundation for your understanding of object-oriented concepts. Figure G.8 is designed to put together all of the pieces of this part of the OO puzzle, so examine it carefully before you continue.

G-3h Superclasses, Subclasses, and Inheritance

Classes are organized into a class *hierarchy*. A **class hierarchy** resembles an upside-down tree in which each class has only one parent class. The class hierarchy is known as a **class lattice** when its classes can have multiple parent classes. Class is used to categorize objects into groups of objects that share common characteristics. For example, the class *automobile* includes large luxury sedans as well as compact cars, and the class *government* includes federal, state, and local governments. Figure G.9 illustrates that the generalization *musical instruments* includes stringed instruments as well as wind instruments.

As you examine Figure G.9, note that Piano, Violin, and Guitar are **subclasses** of Stringed instruments, which is, in turn, a subclass of Musical instruments. Musical instruments defines the **superclass** of Stringed instruments, which is, in turn, the superclass of the Piano, Violin, and Guitar classes. As you can see, the superclass is a more general

protocol

A specific set of rules to accomplish a specific function. In the object oriented data model, protocol refers to a collection of messages to which an object responds.

class hierarchy

The organization of classes in a hierarchical tree in which each parent class is a *superclass* and each child class is a *subclass*. See also *inheritance*.

class lattice

The class hierarchy is known as a class lattice if its classes can have multiple parent classes.

subclasses

See *class hierarchy*.

superclass

In a class hierarchy, the superclass is the more general classification from which the subclasses inherit data structures and behaviors.

classification of its subclasses, which, in turn, are more specific components of the general classification.

The class hierarchy introduces a powerful OO concept known as *inheritance*. **Inheritance** is the ability of an object within the hierarchy to inherit the data structure and behavior (methods) of the classes above it. For example, the Piano class in Figure G.9 inherits its data structure and behavior from the superclasses Stringed instruments and Musical instruments. Thus, Piano inherits the strings and its sounding board characteristic from the Stringed instruments superclass and the musical scale from its Musical instruments superclass. *It is through inheritance that OO systems can deliver code reusability.*

In OO systems, all objects are derived from the superclass Object, or the Root class. Therefore, all classes share the characteristics and methods of the superclass Object. The inheritance of data and methods goes from top to bottom in the class hierarchy. There are two types of inheritance: single inheritance and multiple inheritance.

FIGURE G.8 SUMMARY OF OBJECT CHARACTERISTICS

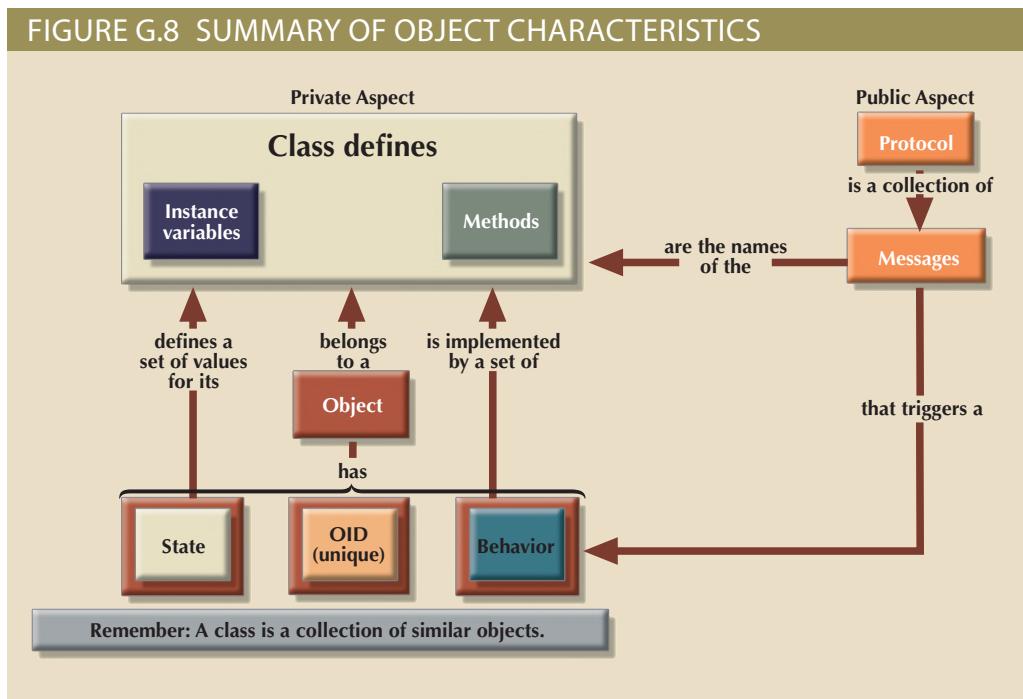
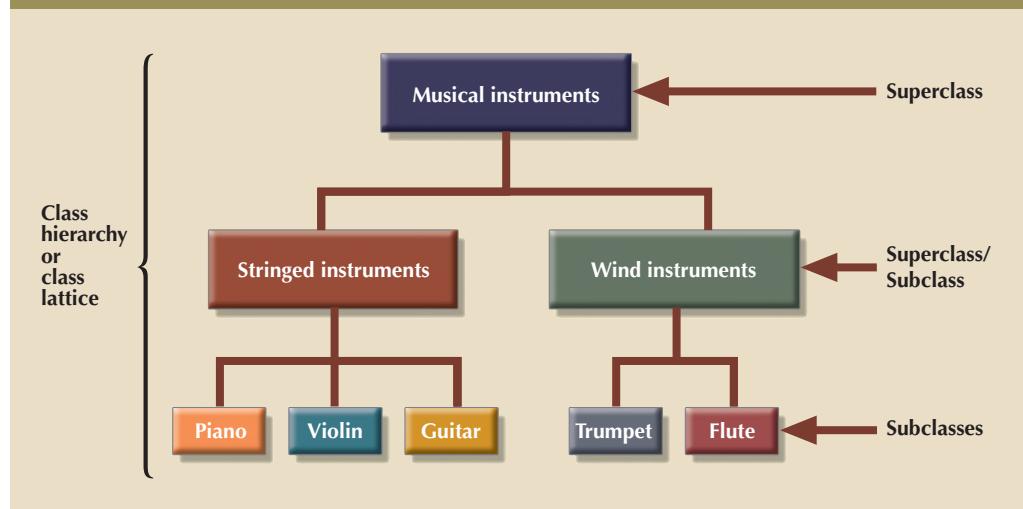


FIGURE G.9 MUSICAL INSTRUMENTS CLASS HIERARCHY



inheritance

In the object-oriented data model, the ability of an object to inherit the data structure and methods of the classes above it in the class hierarchy. See also *class hierarchy*.

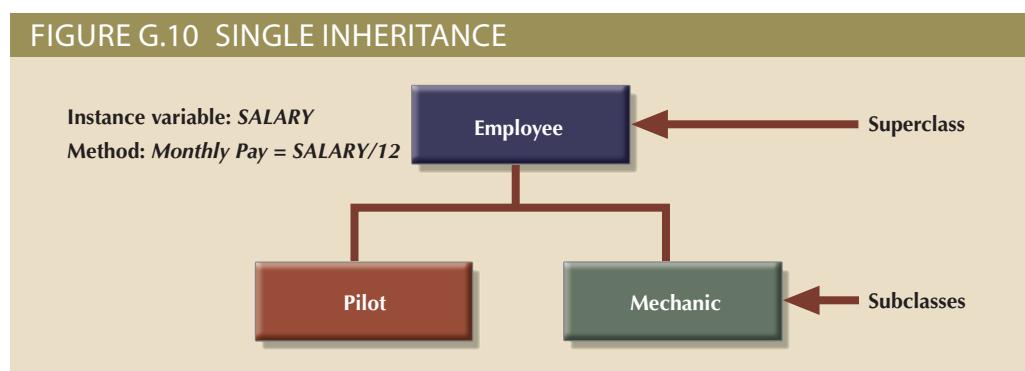
Single Inheritance **Single inheritance** exists when a class has only one immediate (parent) superclass above it. Such a condition is illustrated by the Stringed instruments and Wind instruments classes in Figure G.9. Most of the current OO systems support single inheritance. When the system sends a message to an object instance, the entire hierarchy is searched for the matching method, using the following sequence:

1. Scan the class to which the object belongs.
2. If the method is not found, scan the superclass.

The scanning process is repeated until either one of the following occurs:

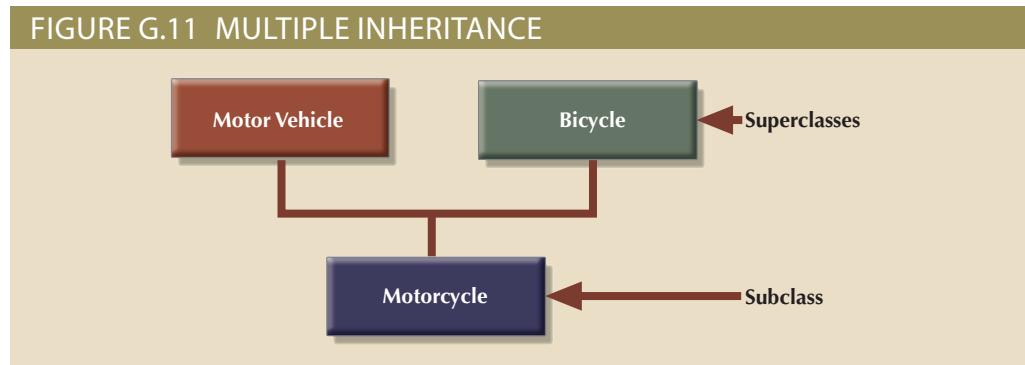
- The method is found.
- The top of the class hierarchy is reached without finding the method. The system then generates a message to indicate that the method was not found.

For an illustration of the scanning process, let's examine the Employee class hierarchy shown in Figure G.10. If the *monthPay* message is sent to a pilot's object instance, the object will execute the *monthPay* method defined in its Employee superclass. Note the code reusability benefits obtained through object-oriented systems: the *monthPay* method's code is available to both the Pilot and Mechanic subclasses.



Multiple Inheritance **Multiple inheritance** exists when a class can have more than one immediate (parent) superclass above it. Figure G.11 provides an example of multiple inheritance, illustrating that the Motorcycle subclass inherits characteristics from both the Motor Vehicle and Bicycle superclasses. From the Motor Vehicle superclass, the Motorcycle subclass inherits:

- Characteristics such as fuel requirements, engine pistons, and horsepower.
- Behavior such as start motor, fill gas, and depress clutch.



single inheritance
In the object oriented data model, the property of an object that allows it to have only one parent superclass from which it inherits its data structure and methods. See also *inheritance, multiple inheritance*.

multiple inheritance
Exists when a class can have more than one immediate (parent) superclass above it.

From the Bicycle superclass, the Motorcycle subclass inherits:

- Characteristics such as two wheels and handlebars.
- Behavior such as straddle the seat and move the handlebar to turn.

The assignment of instance variable or method names must be treated with some caution in a multiple inheritance class hierarchy. For example, if you use the same name for an instance variable or method in each of the superclasses, the OO system must be given some way to decide which method or attribute to use. To illustrate that point, let's suppose that both the Motor Vehicle and Bicycle superclasses shown in Figure G.12 use a MAXSPEED instance variable.

FIGURE G.12 MOTOR VEHICLE AND BICYCLE INSTANCE VARIABLES

	Instance variables	
	Name	Value
Superclass of Motorcycle	Motor Vehicle	MAXSPEED Miles/hour
	Bicycle	MAXSPEED Miles/hour

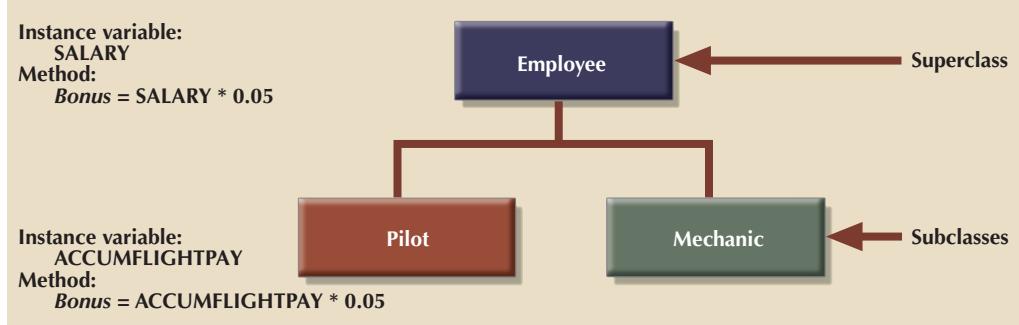
Which version of the MAXSPEED instance variable will be inherited by the Motorcycle's method in this case? A human being would use judgment to correctly assign the 100 miles/hour value to the motorcycle. The OO system, however, cannot make such value judgments and might:

- Produce an error message in a pop-up window, explaining the problem.
- Ask the end user to supply the correct value or to define the appropriate action.
- Yield an inconsistent or unpredictable result.
- Use user-defined inheritance rules for the subclasses in the class lattice. These inheritance rules govern a subclass's inheritance of methods and instance variables.

G-3i Method Overriding and Polymorphism

You may override a superclass method definition by redefining the method at the subclass level. For an illustration of the method override, look at the Employee class hierarchy depicted in Figure G.13.

FIGURE G.13 EMPLOYEE CLASS HIERARCHY METHOD OVERRIDE

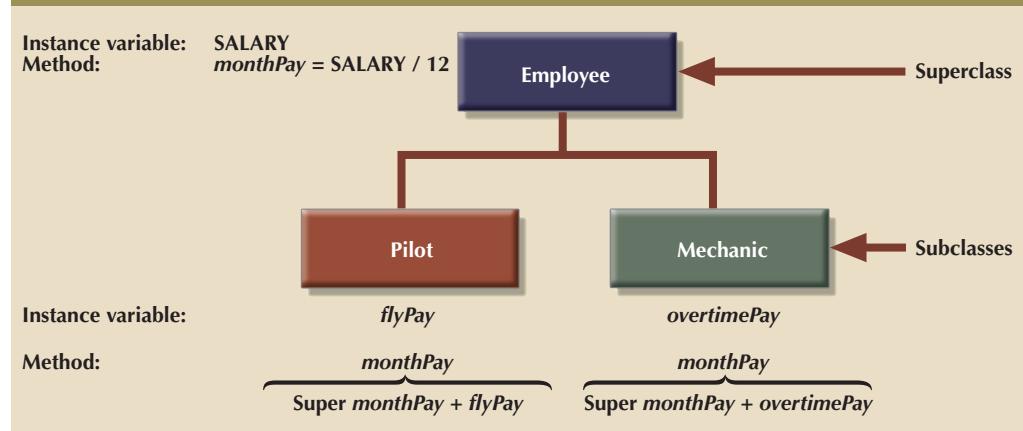


As you examine the summary presented in Figure G.13, note that a *Bonus* method has been defined to compute a Christmas bonus for all employees. The specific bonus computation depends on the type of employee. In this case, *with the exception of pilots*, an employee receives a Christmas bonus equal to 5 percent of his/her salary. Pilots receive a Christmas bonus based on accumulated flight pay rather than annual salary. By defining the *Bonus* method in the Pilot subclass, you are overriding the Employee Bonus method for all objects that belong to the Pilot subclass. However, the Pilot subclass bonus redefinition does not affect the bonus computation for the Mechanic subclass. In contrast to method overriding, polymorphism allows different objects to respond to the same message in different ways. Polymorphism is a very important feature of OO systems because it allows objects to behave according to their specific characteristics. In OO terms, **polymorphism** means that:

- You may use the same name for a method defined in different classes in the class hierarchy.
- The user may send the same message to different objects that belong to different classes and yet will always generate the correct response.

To illustrate the effect of polymorphism, let's examine the expanded Employee class hierarchy shown in Figure G.14. Using the class hierarchy in Figure G.14, the system computes a pilot's or mechanic's monthly pay by sending the same message, *monthPay*, to the pilot or mechanic object. The object will return the correct monthly pay amount even though the *monthPay* includes *flyPay* for the pilot object and *overtimePay* for the mechanic object. The computation of the regular monthly salary payment for both subclasses (Pilot and Mechanic) is the same: the annual salary divided by 12 months.

FIGURE G.14 EMPLOYEE CLASS HIERARCHY POLYMORPHISM



Note

Figure G.14 used Smalltalk syntax: **Super** *monthPay* in the pilot's *monthPay* method indicates that the object inherits its superclass *monthPay* method. Other OOPs, such as C++, use the *Employee.monthPay* syntax.

As you examine the polymorphism example in Figure G.14, note that:

- The Pilot *monthPay* method definition overrides and expands the Employee *monthPay* method defined in the Employee superclass.
- The *monthPay* method defined in the Employee superclass is reused by the Pilot and Mechanic subclasses.

polymorphism

An object oriented data model characteristic by which different objects can respond to the same message in different ways.

Thus, polymorphism augments method override to enhance the code reusability so prized in modular programming and design.

G-3j Abstract Data Types

A data type describes a set of objects with similar characteristics. All conventional programming languages use a set of predefined base data types (real, integer, and string or character). Base data types are subject to a predefined set of operations. For example, the integer base data type allows operations such as addition, subtraction, multiplication, and division.

Conventional programming languages also include type constructors, the most common of which is the record type constructor. For example, a programmer can define a CUSTOMER record type by describing its data fields. The CUSTOMER record represents a new data type that will store CUSTOMER data, and the programmer can directly access that data structure by referencing the record's field names. A record data type allows operations such as WRITE, READ, or DELETE. However, new operations cannot be defined for base data types.

Like conventional data types, an **abstract data type (ADT)** describes a set of similar objects. However, an abstract data type differs from a conventional data type in that:

- The ADT's operations are user-defined.
- The ADT does not allow direct access to its internal data representation or method implementation. In other words, the ADT encapsulates its definition, thereby hiding its characteristics.

Some OO systems, such as Smalltalk, implement base data types as ADTs.

To create an abstract data type, you must define:

- Its name.
- The data representation or instance variables of the objects belonging to the abstract data type; each instance variable has a data type that may be a base data type or another ADT.
- The abstract data type operations and constraints, both of which are implemented through methods.

You might have noted that the abstract data type definition resembles a class definition. Some OO systems differentiate between class and type, using *type* to refer to the *class data structure and methods* and *class* to refer to the *collection of object instances*. A type is a more static concept, while a class is a more run-time concept. In other words, when you define a new class, you are actually defining a new type. The type definition is used as a pattern or template to create new objects belonging to a class at run time.

A simple example will help you understand the subtle distinction between OO type and class. Suppose you bought a cross-stitch pattern with which to create pillow covers. The pattern you bought includes the *description* of its structure as well as *instructions* about its use. That pattern will be the type definition. The collection of all actual pillow covers, each with a unique serial number or OID, that you create with the help of that pattern constitutes the class.

Together with inheritance, abstract data types provide support for complex objects. A **complex object** is formed by combining other objects in a set of complex relations. An example of such a complex object might be found in a security system that uses different data types, such as:

- Conventional (tabular) employee data; for example name, phone, or date of birth.
- Bitmapped data to store the employee's picture.
- Voice data to store the employee's voice pattern.

abstract data type (ADT)

Data type that describes a set of similar objects with shared and encapsulated data representation and methods. An abstract data type is generally used to describe complex objects.
See also *class*.

complex object

An object formed by several different objects in complex relationships.
See also *abstract data types*.

The ability to deal in a relatively easy manner with such a complex data environment gives OO credibility in today's database marketplace.

G-3k Object Classification

An object can be classified according to the characteristics (simple, composite, compound, hybrid, and associative) of its attributes. A **simple object** is an object that contains only single-valued attributes and has no attributes that refer to another object. For example, an object that describes the current semester can be defined as having the following single-valued attributes: SEM_ID, SEM_NAME, SEM_BEGIN_DATE, and SEM_END_DATE.

A **composite object** is an object that contains at least one multivalued attribute and has no attributes that refer to another object. An example of a composite object would be a MOVIE object in a movie rental system. For example, MOVIE might be defined as having the following attributes: MOVIE_ID, MOVIE_NAME, MOVIE_PRICE, MOVIE_TYPE, and MOVIE_ACTORS. In that case, MOVIE_ACTORS is a multivalued attribute that tracks the many performers in the movie.

A **compound object** is an object that contains at least one attribute that references another object. An example is the STUDENT object in Table G.2. In that example, the ADVISOR attribute refers to the PROFESSOR object.

A **hybrid object** is an object that contains a repeating group of attributes, at least one of which refers to another object. A typical example of a hybrid object is the invoice example introduced in Chapter 3, The Relational Database Model, in Figure 3.30. In that case, an invoice contains many products, and each product has a quantity sold and a unit price. The object representation of the invoice contains a repeating group of attributes that represent the product, quantity sold, and unit price (PROD_CODE, LINE_UNITS, and LINE_PRICE) for each product sold. Therefore, the object representation of the invoice does not require a new INV_LINE object as in the ER model representation.

Finally, an **associative object** is an object used to represent a relationship between two or more objects. The associative object can contain its own attributes to represent specific characteristics of the relationship. A good example of an associative object is the Enroll example in Chapter 3, Figure 3.26. In that case, the ENROLL object relates to a STUDENT and a CLASS object and includes an ENROLL_GRADE attribute that represents the grade earned by the student in the class.

In real-world data models, you find fewer simple and composite objects and more compound, hybrid, and associative objects. Those types of objects will be discussed in greater detail in Section G-4c.

G-4 Characteristics of an Object-Oriented Data Model

The object-oriented concepts described in previous sections represent the core characteristics of an **object-oriented data model (OODM)**, also known as an object data model, or ODM. At the very least, an object-oriented data model must:

- Support the representation of complex objects.
- Be **extensible**; that is, it must be capable of defining new data types as well as the operations to be performed on them.
- Support encapsulation; that is, the data representation and the method's implementation must be hidden from external entities.

simple object

An object that contains only single-valued attributes and has no attributes that refer to another object.

composite object

An object that contains at least one multivalued attribute and has no attributes that refer to another object.

compound object

An object that contains at least one attribute that references another object.

hybrid object

The type of object classification that contains and represents a repeating group of attributes. One or more of the attributes reference another object that usually summarizes the contents of the hybrid object.

associative object

An object used to represent a relationship between two or more objects.

object-oriented data model (OODM)

A data model whose basic modeling structure is an object.

extensible

Capable of being extended by adding new data types and the operations to be performed on them.

- Exhibit inheritance; an object must be able to inherit the properties (data and methods) of other objects.
- Support the notion of object identity (OID) described earlier in this appendix.

For instructional purposes and to the extent possible, OODM component descriptions and definitions will be used to correspond to the entity relationship model components described in Chapter 3. Although most of the basic OODM components were defined earlier in this appendix chapter, a quick summary may help you read the subsequent material more easily:

- The OODM models real-world entities as *objects*.
- Each object is composed of *attributes* and a set of *methods*.
- Each attribute can *reference* another object or a set of objects.
- The attributes and the methods' implementation are hidden, *encapsulated*, from other objects.
- Each object is identified by a unique *object ID (OID)*, which is independent of the value of its attributes.
- Similar objects are described and grouped in a *class* that contains the description of the data (attributes or instance variables) and the methods' implementation.
- The class describes a *type* of object.
- Classes are organized in a *class hierarchy*.
- Each object of a class *inherits* all properties of its superclasses in the class hierarchy.

Armed with that summarized OO component description, note the comparison between the OO and ER model components presented in Table G.3.

TABLE G.3

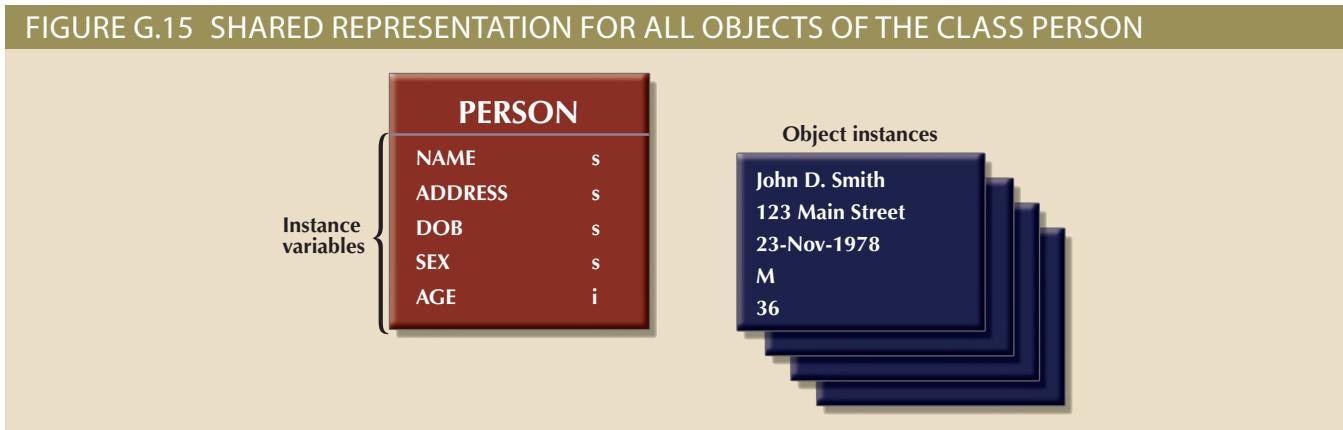
COMPARING THE OO AND ER MODEL COMPONENTS

OO DATA MODEL	ER MODEL
Type	Entity definition
Object	Entity
Class	Entity set
Instance variable	Attribute
N/A	Primary key
OID	N/A
Method	N/A
Class hierarchy	ER diagram

G-4a Object Schemas: The Graphical Representation of Objects

A graphical representation of an object resembles a box, with the instance variable names inside the box. Generally speaking, the object representation is shared by all objects in the class. Therefore, you will discover that the terms *object* and *class* are often used interchangeably in the illustrations. With that caveat in mind, let's begin by examining the illustration based on the Person class, shown in Figure G.15. In that case, the instance variables NAME, ADDRESS, DOB, and SEX use a string base data type, and the AGE instance variable uses an integer base data type.

FIGURE G.15 SHARED REPRESENTATION FOR ALL OBJECTS OF THE CLASS PERSON

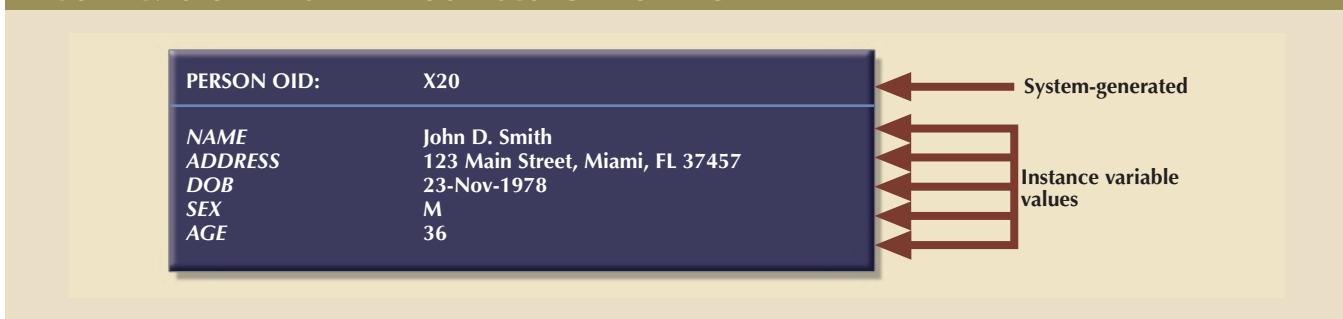


Next, let's examine the *state* of a Person object instance. (See Figure G.16.) As you examine Figure G.16, note that the AGE instance variable can also be viewed as a *derived* attribute. Derived attributes may be implemented through *methods*. For instance, a method named *Age* can be created for the Person class. That method will return the difference in years between the current date and the date of birth (DOB) for a given object instance. Aside from the fact that methods can generate derived attribute values, methods have the added advantages of encapsulation and inheritance.

object space

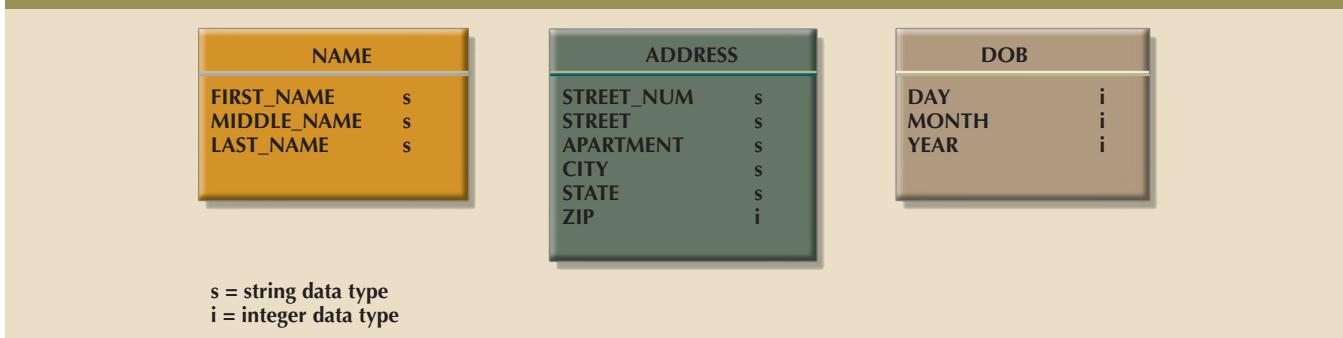
The equivalent of the database schema, as seen by the designer in an object oriented database.

FIGURE G.16 STATE OF A PERSON OBJECT INSTANCE



Keep in mind that the OO environment allows you to create abstract data types from base data types. For example, NAME, ADDRESS, and DOB are composite attributes that can be implemented through classes or ADTs. To illustrate that point, Name, Address, and DOB have been defined to be abstract data types in Figure G.17.

FIGURE G.17 DEFINING THREE ABSTRACT DATA TYPES



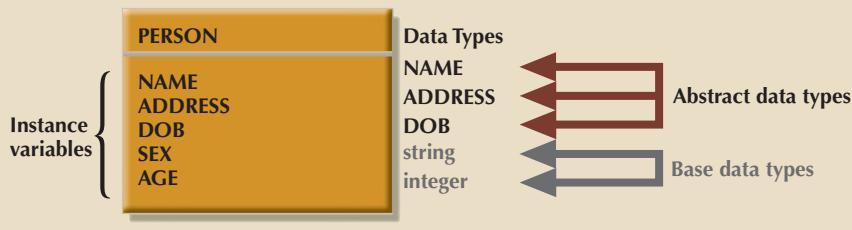
As you examine Figure G.17, note that the Person class now contains attributes that point to objects of other classes or abstract data types. The new data types for each instance variable of the class Person are shown in Figure G.18.

The **object space**, or **object schema**, is used to represent the composition of the state of an object at a given time.

object schema

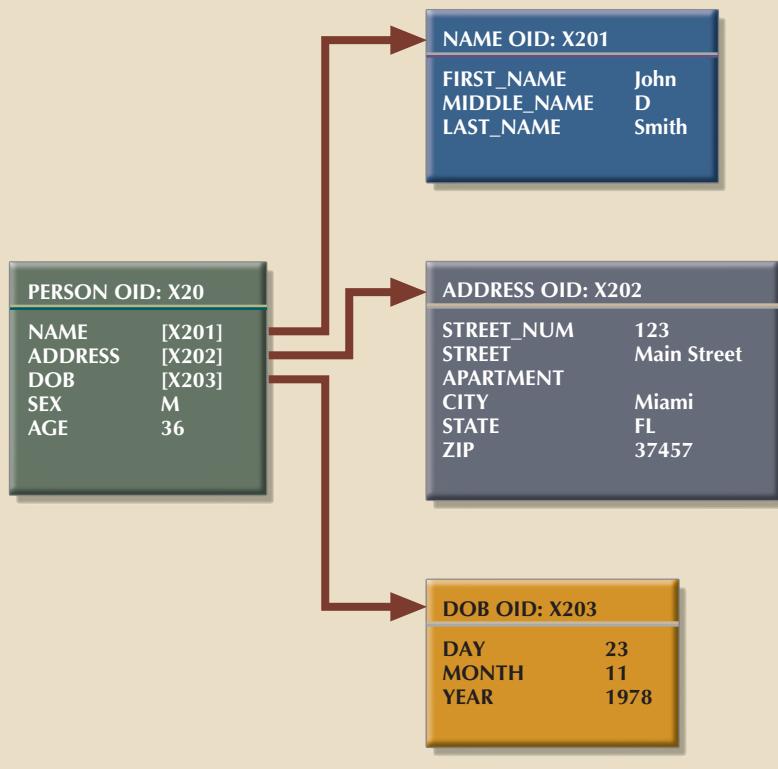
See *object space*.

FIGURE G.18 OBJECT REPRESENTATION FOR INSTANCES OF THE CLASS PERSON WITH ADTs



The object's state for an instance of class Person is illustrated in Figure G.19. As you examine Figure G.19, note the use of OIDs to reference other objects. For example, the attributes NAME, ADDRESS, and DOB now contain an OID of an instance of their respective class or ADT instead of the base value. The use of OIDs for object references avoids the data inconsistency problem that would appear in a relational system if the end user were to change the primary key value when changing the object's state. That is because the OID is independent of the object's state.

FIGURE G.19 OBJECT STATE FOR AN INSTANCE OF THE CLASS PERSON, USING ADTs



referential object sharing

When an object instance is referenced by other objects. That is, two or more different objects point to the same object instance, a change in the referenced object instance values is automatically reflected in all other referring objects.

To illustrate this point further, a rental property application will be used by which many rental properties and the persons living in them are tracked. In that case, two people living at the same address are likely to reference the same Address object instance. (See Figure G.20.) This condition is sometimes labeled as **referential object sharing**. A change in the Address object instance will be reflected in both Person instances.

Figure G.20 illustrates the state of two different object instances of the class Person; both object instances reference the same Address object instance. Note that Figure G.20 depicts four different classes or ADTs: Person (two instances), Name (two instances), Address, and DOB (two instances).

G-4b Class-Subclass Relationships

Do you remember that classes inherit the properties of their superclasses in the class hierarchy? That property leads to the use of the label “is a” to describe the relationship between the classes within the hierarchy. That is, an employee *is a* person, and a student *is a* person. This basic idea is sufficiently important to warrant a more detailed illustration based on the class hierarchy. (See Figure G.21.)

FIGURE G.20 REFERENTIAL OBJECT SHARING

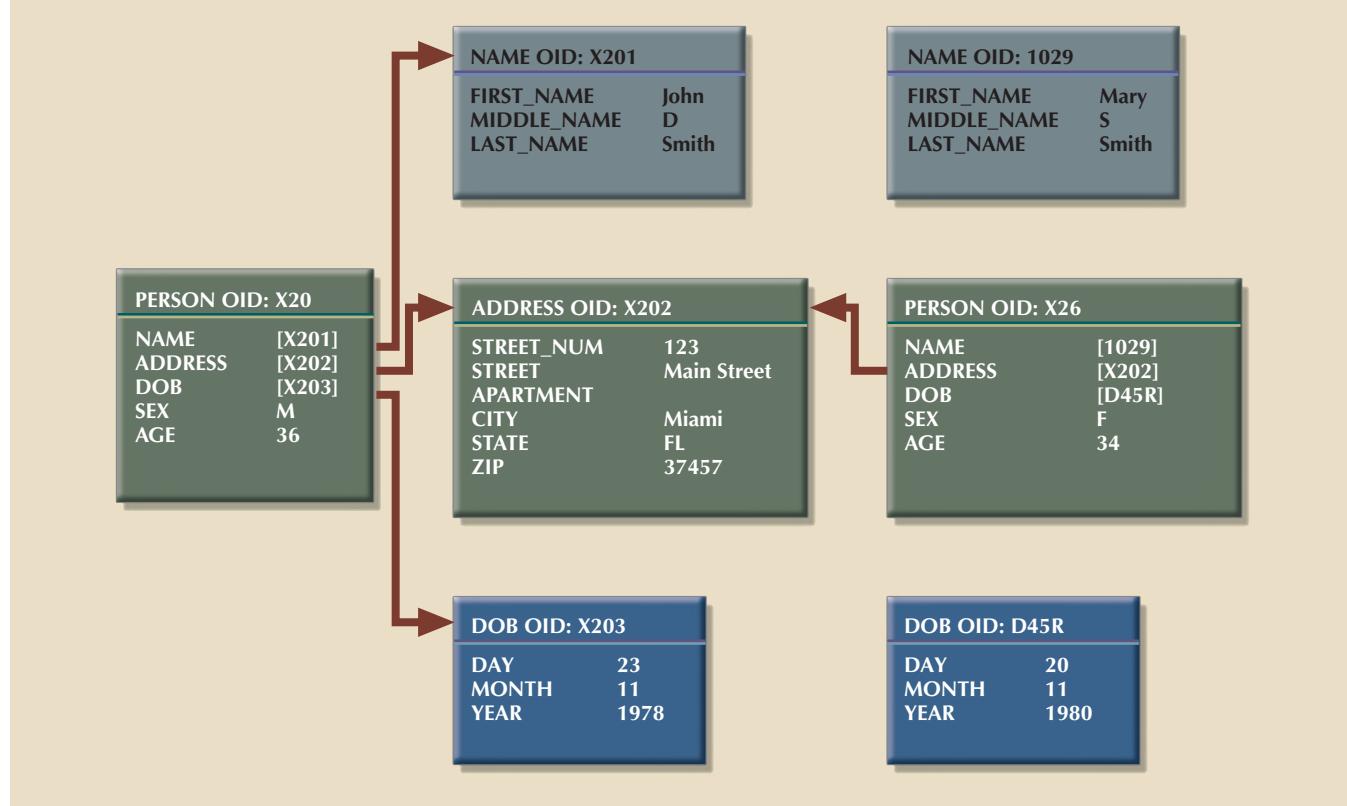
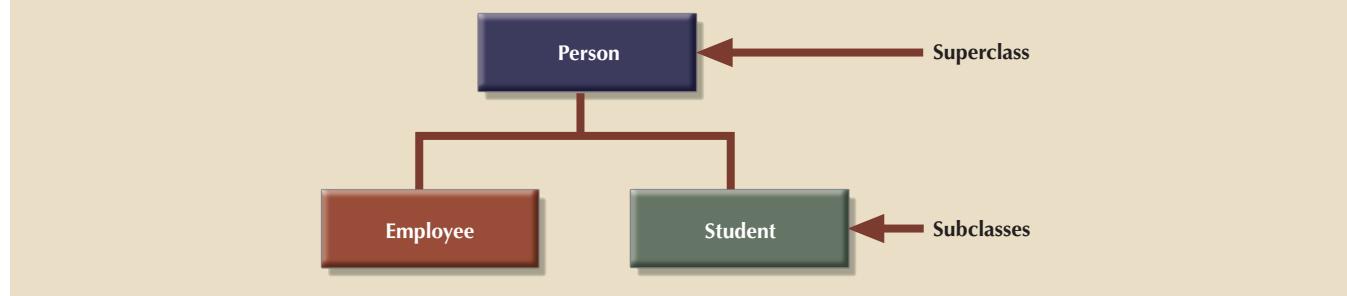
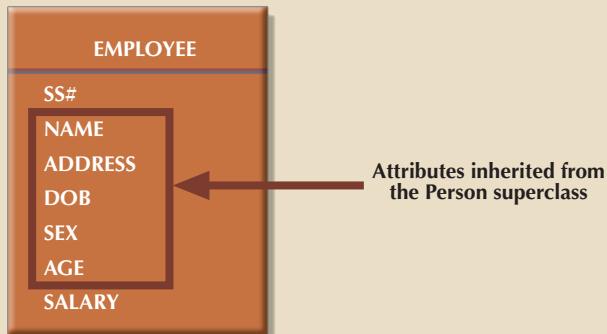


FIGURE G.21 CLASS HIERARCHY



In the hierarchy shown in Figure G.21, the Employee object is described by seven attributes, shown in Figure G.22. Social Security number (SS#) is recorded as a string base data type, and SALARY is recorded as an integer base data type. The NAME, ADDRESS, DOB, SEX, and AGE attributes are all inherited from the Person superclass.

FIGURE G.22 EMPLOYEE OBJECT REPRESENTATION



interobject relationship

An attribute-class relationship created when an object's attribute references another object of the same or a different class.

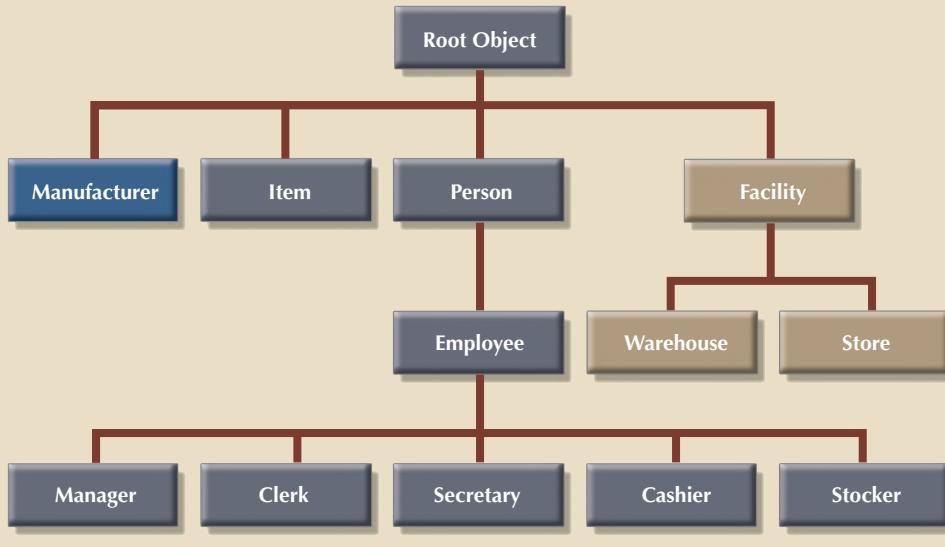
That example is based on the fact that the OODM supports the *class-subclass relationship*, for which it enforces the necessary integrity constraints. Note that the relationship between a superclass and its subclasses is 1:M; that is, if you assume single inheritance, each superclass can have many subclasses and each subclass is related to only one superclass.

G-4c Interobject Relationships: Attribute-Class Links

In addition to supporting the class-subclass relationship, the OODM supports the attribute-class relationship. An attribute-class relationship, or **interobject relationship**, is created when an object's attribute references another object of the same or different class.

The interobject relationship is different from the class-subclass relationship explored earlier. To illustrate this difference, let's examine the class hierarchy for the EDLP (Every Day Low Prices) Retail Corporation, shown in Figure G.23.

FIGURE G.23 CLASS HIERARCHY FOR THE EDLP RETAIL CORPORATION

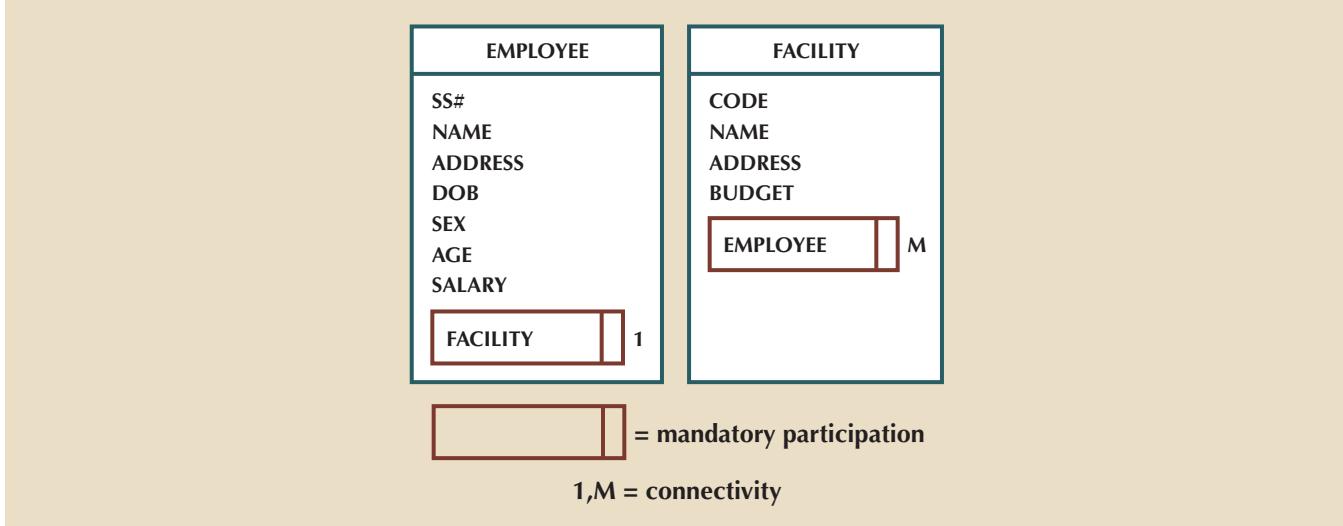


As you examine Figure G.23, note that all classes are based on the Root Object superclass. The class hierarchy contains the classes Manufacturer, Item, Person, and Facility.

The Facility class contains the subclasses Warehouse and Store. The Person class contains the subclass Employee, which, in turn, contains the subclasses Manager, Clerk, Secretary, Cashier, and Stocker. The following discussion will use the simple class hierarchy shown in Figure G.23 to illustrate basic 1:M and M:N relationships.

Representing 1:M Relationships Based on the class hierarchy in Figure G.23, a one-to-many relationship exists between Employee and Facility: each Employee works in only one Facility, but each Facility has several Employees. Figure G.24 shows how that relationship may be represented.

FIGURE G.24 REPRESENTING A 1:M RELATIONSHIP



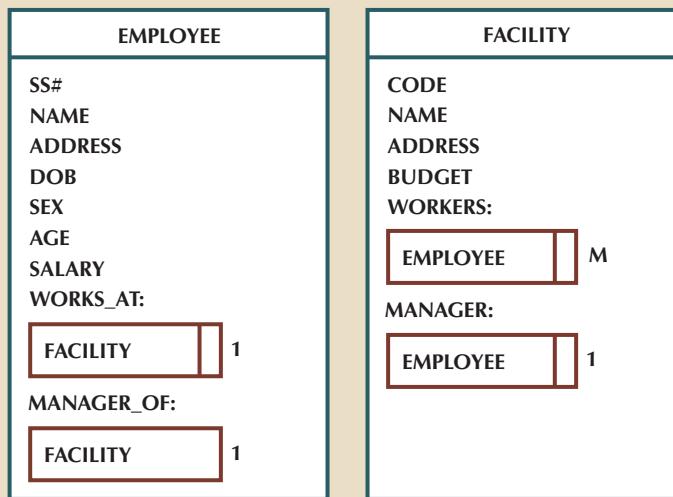
As you examine the relationship between Employee and Facility portrayed in Figure G.24, note that the Facility object is included within the Employee object and vice versa; that is, the Employee object is also included within the Facility object. The following techniques will be used to examine the relationships in greater detail:

- Related classes are enclosed in boxes to make relationships more noticeable.
- The double line on the boxes' right side indicates that the relationship is mandatory.
- Connectivity is indicated by labeling each box. In this case, a *1* was put next to Facility in the Employee object to indicate that each employee works in only one facility. The *M* beside Employee in the Facility object indicates that each facility has many employees.

Note that the ER notation is used to represent a mandatory entity and to indicate the connectivity of a relationship (1:M). The purpose of the notation is to maintain consistency with earlier diagrams.

Rather than just include the object box within the class, the preference is to use a name that is descriptive of the class *characteristic* being modeled. That procedure is especially useful when two classes are involved in more than one relationship. In those cases, the attribute's name should be written above the class box, and the class box should be indented to indicate that the attribute will reference the class. For example, two relationships between Employee and Facility can be represented by using WORKS_AT and WORKERS, as indicated in Figure G.25. Note that two relationships exist:

FIGURE G.25 REPRESENTING 1:1 AND 1:M RELATIONSHIPS



Note: the Manager attribute indicates the facility's general manager

1. The 1:M relationship is based on the business rule “each facility employs many employees, and each employee is employed by only one facility.”
2. The 1:1 relationship is based on the business rule “each facility is managed by only one employee, and each manager manages only one facility.”

As you examine Figure G.25, note that the relationships are represented in both participating classes. That condition allows you to invert the relationship, if necessary. For example, the Facility object within the Employee object represents the “Manager_of” relationship. In this case, the Facility object is optional and has a maximum connectivity of 1. The Employee and Facility objects are examples of compound objects. Another type of 1:M relationship can be illustrated by examining the relationship between employees and their dependents. To establish that relationship, you first create a Dependent subclass, using Person as its superclass. *Note that a Dependent subclass cannot be created by using Employee as its superclass because the class hierarchy represents an “is a” relationship.* In other words, each Manager is an Employee, each Employee is a Person, each Dependent is a Person, and each Person is an Object in the object space—but each Dependent is not an Employee. Figure G.26 shows the proper presentation of the relationship between Employee and Dependent.

As you examine Figure G.26, note that Dependent is optional to Employee and that Dependent has a 1:M relationship with Employee. However, Employee is mandatory to Dependent. *The weak entity concept disappears in the OODM because each object instance is identified by a unique OID.*

Representing M:N Relationships Using the same EDLP Retail Corporation class hierarchy, a many-to-many (M:N) relationship can be illustrated by exploring the relationship between Manufacturer and Item, as represented in Figure G.27. Figure G.27 depicts a condition in which each Item may be produced by many Manufacturers, and each Manufacturer may produce many Items. Thus, Figure G.27 represents a conceptual view of the M:N relationship between Item and Manufacturer. In this representation, Item and Manufacturer are both compound objects.

Also note that the CONTACT attribute in the Manufacturer class in Figure G.27 references only one instance of the Person class. A slight complication arises at this point. It is likely that each contact (person) has a phone number, yet no phone number attribute

FIGURE G.26 EMPLOYEE-DEPENDENT RELATIONSHIP

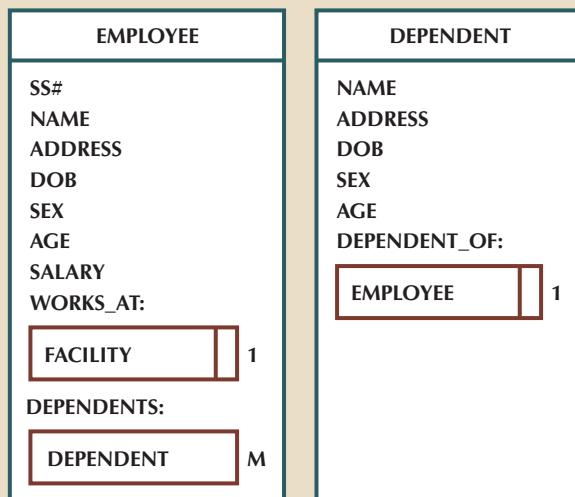
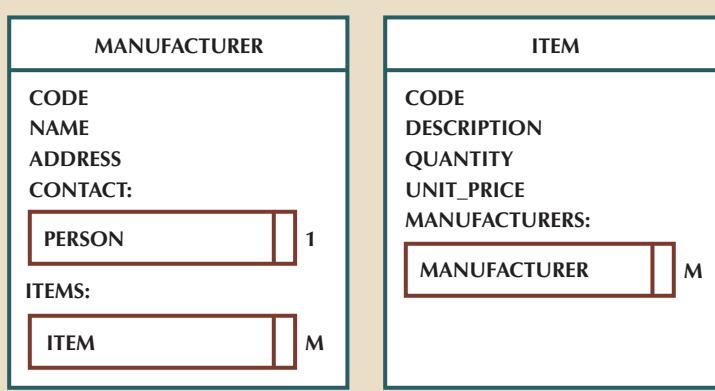


FIGURE G.27 REPRESENTING THE M:N RELATIONSHIP

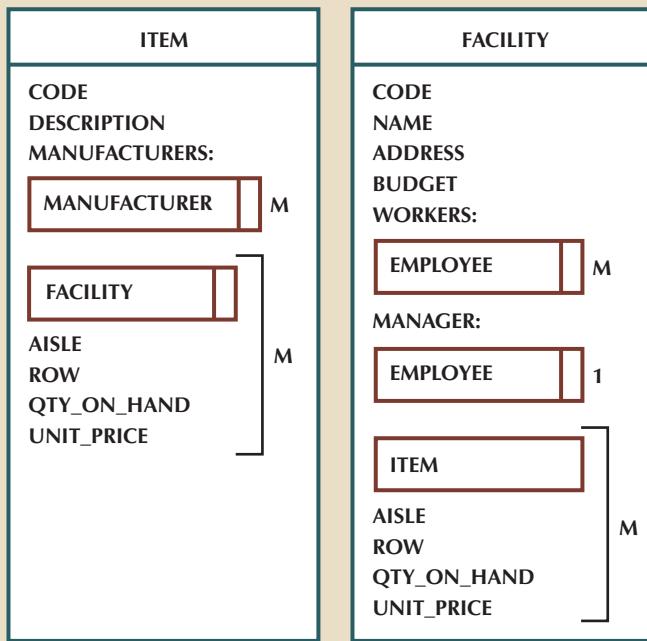


was included in the Person class. In that case, the designer may add the attribute so it will be available to all Person subclasses.

Representing M:N Relationships with an Intersection Class Suppose you add a condition to the just-explored ITEM class that allows you to track additional data for each item. For example, let's represent the relationship between Item and Facility so that each Facility may contain several Items, and each Item may be located at several Facilities. In addition, you want to track the quantity and location (aisle and row) of an Item at each Facility. Those conditions are illustrated in Figure G.28. The right square bracket "]" in Figure G.28 indicates that the included attributes are treated as one logical unit. Therefore, each Item instance may contain several occurrences of Facility, each accompanied by related values for the AISLE, ROW, QTY_ON_HAND, and UNIT_PRICE attributes. The inverse is true for each instance of Facility. The Item and Facility objects in this relationship are hybrid objects with a repeating group of attributes. Note that the semantic requirements for this relationship indicate that the Item or Facility objects are accessed first so the aisle, row, and quantity on hand are known for each item.

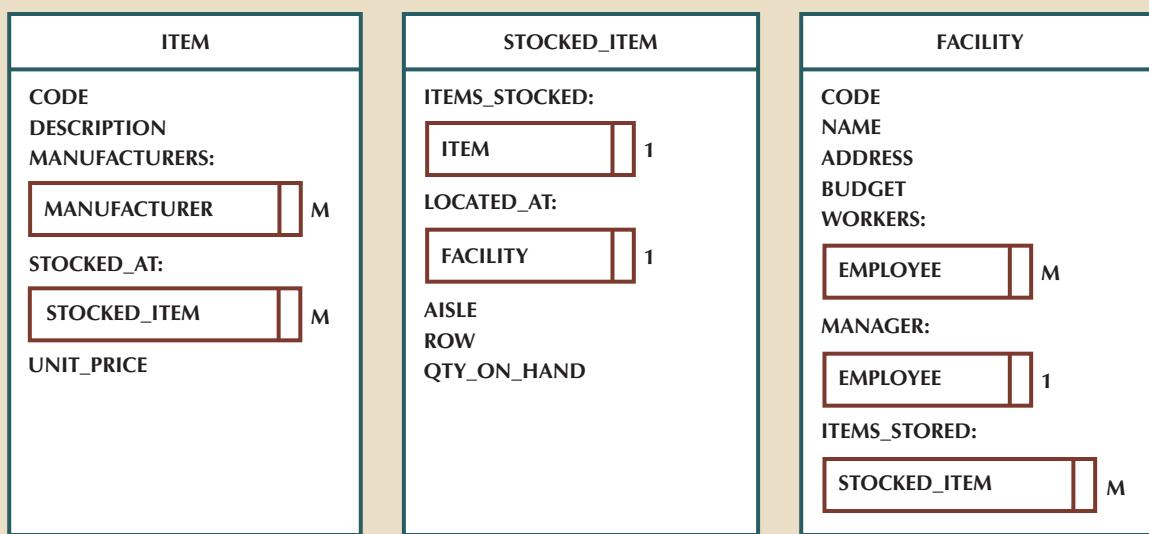
To translate the preceding discussion to a more relational view of the M:N scenario, you would have to define an **intersection** (bridge) **class** to connect both Facility and

FIGURE G.28 REPRESENTING THE M:N RELATIONSHIP WITH ASSOCIATED ATTRIBUTES



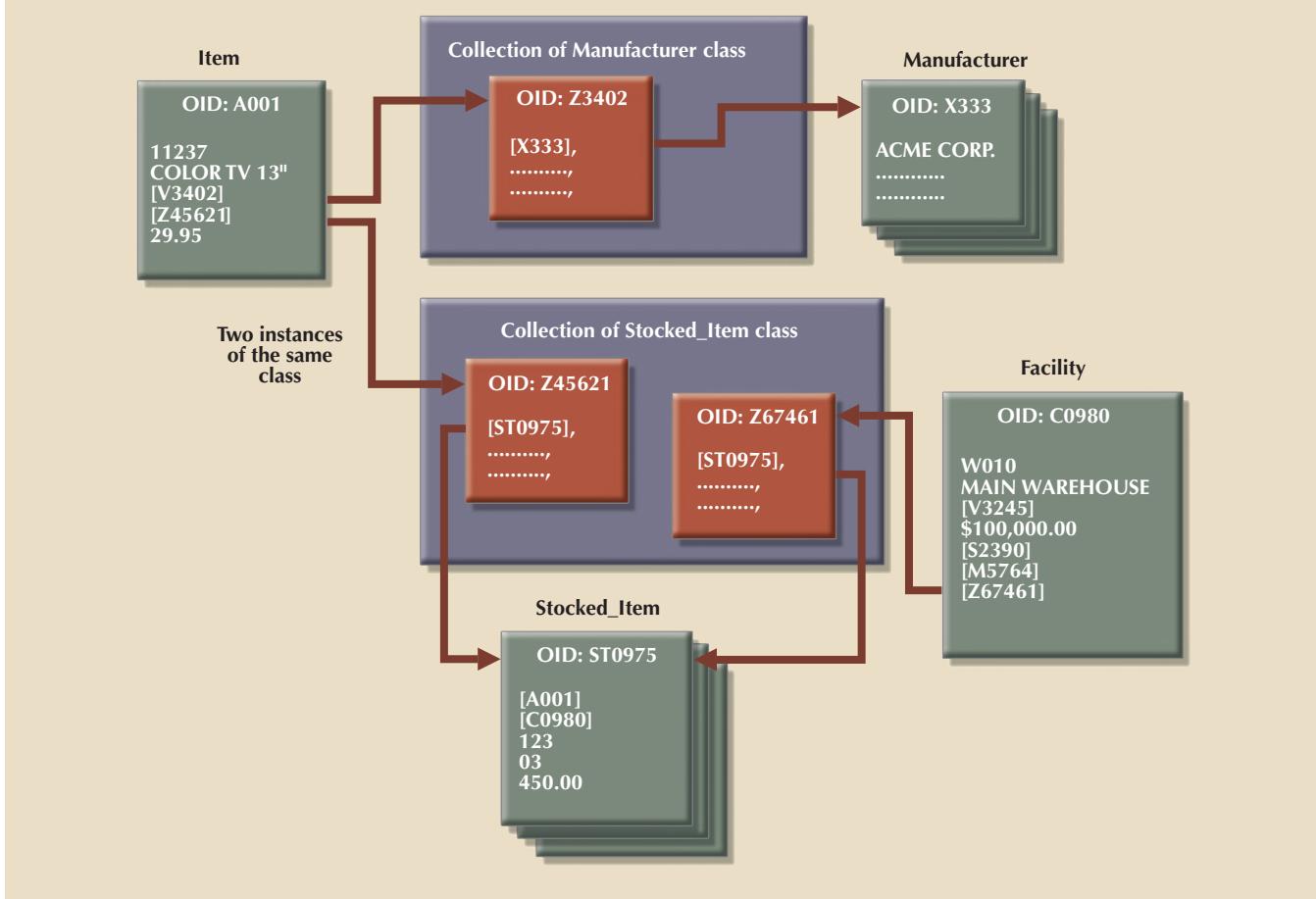
Item and store the associated attributes. In that case, you might create a Stocked_Item associative object class to contain the Facility and Item object instances and the values for each of the AISLE, ROW, QTY_ON_HAND, and UNIT_PRICE attributes. Such a class is equivalent to the Interclass_Connection construct of the Semantic Data Model. Figure G.29 shows how the Item, Facility, and Stocked_Item object instances might be represented.

FIGURE G.29 REPRESENTING THE M:N RELATIONSHIP WITH INTERSECTION CLASS



Having examined the depiction of the basic OO relationships, you can represent the object space as shown in Figure G.30.

FIGURE G.30 OBJECT SPACE REPRESENTATION



Because Figure G.30 contains much critical design information, you should examine the following points in particular:

- The Stocked_Item associative object instance contains references to an instance of each related (Item and Facility) class. The Stocked_Item intersection class is necessary only when you must keep track of the additional information referred to earlier.
- The Item object instance in this object schema contains the collection of Stocked_Item object instances, each one of which contains a Facility object instance. The inverse of that relationship is also true: a Facility object instance contains the collection of Stocked_Item object instances, each one of which contains an Item object instance. *You should realize that those two relationships represent two different application views of the same object schema.* It is desirable for a data model to provide such flexibility.
- The interobject references use the OID of the referenced objects in order to *access* and *include* them in the object space.
- The values inside square brackets “[]” represent the OID of an object instance of some class. *The “collection of” classes represent a class of objects in which each object instance contains a collection of objects of some class.* For example, the Z3402 and Z45621 OIDs reference objects that constitute a collection of Manufacturers and a collection of Stocked_Items, respectively.

- In the relational model, the ITEM table would not contain any data regarding the MANUFACTURERS or the STOCKED_ITEMS in its structure. To provide (combined) information about ITEM, STOCKED_ITEM, and FACILITY, you would have to perform a relational join operation. The OODM does not need joins to combine data from different objects because the Item object *contains* the references to the related objects; those references are automatically brought into the object space when the Item object is accessed.

G-4d Late and Early Binding: Use and Importance

A desirable OODM characteristic is its ability to let any object's attribute contain objects that define different data types (or classes) at different times. With that feature, an object can contain a *numeric value* for a given instance variable, and the next object (of the same class) can contain a *character value* for the same instance variable. That characteristic is achieved through late binding. With **late binding**, the data type of an attribute is not known until execution time or run time. Therefore, two different object instances of the same class can contain values of different data types for the same attribute.

In contrast to the OODM's ability to use late binding, a conventional DBMS requires that a base data type be defined for each attribute at the time of its creation. For example, suppose you want to define an INVENTORY to contain the following attributes: ITEM_TYPE, DESCRIPTION, VENDOR, WEIGHT, and PRICE. In a conventional DBMS, you create a table named INVENTORY and assign a base data type to each attribute, as shown in Figure G.31.

Recall from earlier chapters that when the designer is working with conventional database systems, (s)he must define the data type for each attribute *when the table structure is defined*. That approach to data type definition is called early binding. **Early binding** allows the database to check the data type for each of the attribute's values at compilation or definition time. For instance, the ITEM_TYPE attribute in Figure G.31 is limited to numeric values. Similarly, the VENDOR attribute may contain only numeric values to match the primary key of some row in a VENDOR table with the same numeric value restriction.

Now let's take a look at Figure G.32 to see how an OODM would handle this early-binding problem. As was true in the conventional database environment, the OODM allows the data types to be defined at creation time. However, quite *unlike* the conventional database, the OODM allows the data types to be user-defined ADTs. In this example of early binding, the abstract data types Inv_type, String_of_characters, Vendor, Weight, and Money are associated with the instance variables at definition time. Therefore, the designer may define the required operations for each data type. For example, the Weight data type can have methods to show the weight of the item in pounds or kilograms. Similarly, the Money data type may have methods to return the price as numbers or letters denominated in U.S. dollars, euros, or other currencies. (Remember that abstract data types are implemented through classes.)

In a late-binding environment, the object's attribute data type is not known prior to its use. Therefore, an attribute can have any type of value assigned to it. Using the same basic data set described earlier, Figure G.33 shows the attributes (instance variables) ITEM_TYPE, DESCRIPTION, VENDOR, WEIGHT, and PRICE without a prior data type definition. Because no data types are predefined for the class instance variables, two different objects of the Inventory class may have different value types for the same attribute. For example, ITEM_TYPE can be assigned a character value in one object instance and a numeric value in the next instance. Late binding also plays an important role in polymorphism, allowing the object to decide which implementation method to use at run time.

late binding

A characteristic in which the data type of an attribute is not known until execution time or run-time.

early binding

A property by which the data type of an object's attribute must be known at definition time, bonding the data type to the object's attribute. Characteristic of an object oriented data model. See also *late binding*.

FIGURE G.31 INVENTORY TABLE WITH PREDETERMINED (BASE) DATA TYPES

Table: INVENTORY

Attributes	Conventional (Base) Data Type
ITEM_TYPE	Numeric
DESCRIPTION	Character
VENDOR	Numeric
WEIGHT	Numeric
PRICE	Numeric

FIGURE G.32 INVENTORY CLASS WITH EARLY BINDING

Class: INVENTORY

Instance variables	Type
ITEM_TYPE	Inv_type
DESCRIPTION	String_of_characters
VENDOR	Vendor
WEIGHT	Weight
PRICE	Money

FIGURE G.33 OODM INVENTORY CLASS WITH LATE BINDING

Class: INVENTORY

Instance variables	Type
ITEM_TYPE	
DESCRIPTION	
VENDOR	
WEIGHT	
PRICE	No data type is known when the class is created

G-4e Support for Versioning

Versioning is an OODM feature that allows you to track the history of change in the state of an object. Versioning is a very powerful modeling feature, especially in computer-aided design (CAD) environments. For example, an engineer using CAD can load a machine component design in his/her workstation, make some changes, and see how those changes affect the component's operation. If the changes do not yield the expected results, the engineer can undo those changes and restore the component to its original state. Versioning is one of the reasons the OODBMS is such a strong player in the CAD and computer-aided manufacturing (CAM) arenas.

G-5 OODM and Previous Data Models: Similarities and Differences

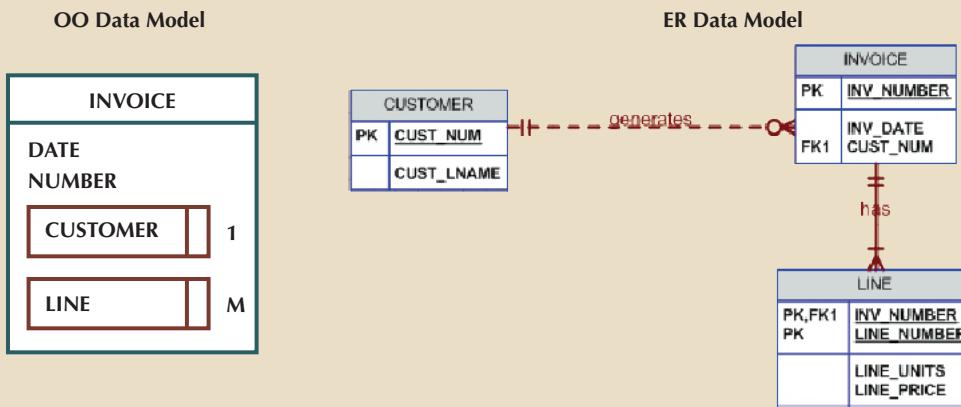
Although the OODM has much in common with relational and ER data models, the OODM introduces some fundamental differences. The following summary is designed to offer detailed comparisons to help clarify the OODM characteristics introduced in this appendix.

G-5a Object, Entity, and Tuple

versioning
A property of an OODBMS that allows the database to keep track of the different transformations performed on an object.

The OODM concept of *object* extends well beyond the concept of *entity* or *tuple* in other data models. Although an OODM object resembles the entity and the tuple in the ER and relational models, an OODM object has additional characteristics, such as behavior, inheritance, and encapsulation. Those OODM characteristics make OO modeling more natural than ER and relational modeling. In fact, the ER and relational models often force the designer to create new artificial entities to represent real-world entities. For example, in the ER model, an invoice is usually represented by two separate entities; the second (LINE) entity is usually weak because its existence depends on the first (INVOICE) entity and its primary key is partially derived from the INVOICE entity. (See Figure G.34.)

FIGURE G.34 AN INVOICE REPRESENTATION



As you examine Figure G.34, note that the ER approach requires the use of two different entities to model a single real-world INVOICE entity. That artificial construct is imposed by the relational model's inherent limitations. The ER model's artificial representation introduces additional overhead in the underlying system. In contrast, the OODM's INVOICE object is directly modeled as an object into the object space, or object schema.

G-5b Class, Entity Set, and Table

The concept of *class* can be associated with the ER and relational models' concepts of *entity set* and *table*, respectively. However, class is a more powerful concept that allows the description of not only the data structure but also the behavior of the class objects. A class also allows for both the concept and the implementation of abstract data types in the OODM. The ADT is a very powerful modeling tool because it allows the end user to create new data types and use them like any other base data type that accompanies a database. Thus, the ADT yields an increase in the *semantic* content of the objects being modeled.

G-5c Encapsulation and Inheritance

ADT brings two other OO features that are not supported in previous models: encapsulation and inheritance. Classes are organized in class hierarchies. An object belonging to a class inherits all properties of its superclasses. Encapsulation means that the data representation and the method's implementation are hidden from other objects and from the end user. In an OODM, only the methods can access the instance variables. In contrast, the conventional system's data components or fields are directly accessible from the external environment.

Conventional models do not incorporate the methods found in the OODM. The closest thing to methods is the use of triggers and stored procedures in SQL databases. However, because triggers do not include the encapsulation and inheritance benefits that are typical of the object model's methods, triggers do not yield the same functionality as methods.

G-5d Object ID

The object ID (OID) is not supported in either the ER or the relational model. Although database users may argue that Oracle Sequences and MS Access AutoNumber provide the same functionality as an OID, that argument is true *only* to the extent that they can be used to uniquely identify data elements. However, unlike the object model, in which the relationships are implicit, the relational model still uses value-based relationships such as:

```
SELECT      *
FROM        INVOICE, INV_LINE
WHERE       INVOICE.INV_ID = INV_LINE.INV_ID;
```

The hierarchical and CODASYL models support some form of ID that can be considered similar to the OID, thus supporting the argument presented by some researchers who insist that the OO evolution is a step back on the road to the old pointer systems. Therefore, OO-based systems return to the modeling and implementation complexities that were typical of the hierarchical and network models.

G-5e Relationships

The main property of any data model is found in its representation of relationships among the data components. The relationships in an OODM can be of two types: inter-class references or class hierarchy inheritance. The ER and the relational models use a *value-based* relationship approach. Using a value-based approach means that a relationship among entities is established through a common value in one or several of the entity attributes. In contrast, the OODM uses the object ID, which is identity-based, to establish relationships among objects, and *those relationships are independent of the state of the object*. (While that property makes it easy to deal with the database objects at the end-user applications level, you may have concluded that the price of the convenience is greater conceptual complexity.)

G-5f Access Methods

The ER and relational data models depend on the use of SQL to retrieve data from the database. SQL is a set-oriented query language that is based on a formally defined mathematical model. Given its set-oriented heritage and based on the value of some of its attributes, SQL uses associative access methods to retrieve related information from a database. For example, to retrieve a list of customer records based on the value of their year-to-date purchases, SQL would use:

```
SELECT      *
FROM        CUSTOMER
WHERE       CUS_YTD_BUYS >= 5000;
```

If no CUS_YTD_BUYS value parameter is specified, SQL “understands” that condition to mean “any value,” thus reducing the query statement to:

```
SELECT      *
FROM        CUSTOMER;
```

As a consequence of having more semantics in its data model, the OODM produces a schema in which relations form part of the structure of the database. Accessing the structured object space resembles the record-at-a-time access of the old structured hierarchical and network models, especially if you use a 3GL or even the OOPL supported by the OODBMS. The OODM is suited to support both navigational (record-at-a-time) and set-oriented access. The navigational access is provided and implemented directly by the OODM through the OIDs. The OODM uses the OIDs to navigate through the object space structure developed by the designer.

Associative set-oriented access in the OODM must be provided through explicitly defined methods. Therefore, the designer must implement operations to manipulate the object instances in the object schema. The implementation of those operations will have an effect on performance and on the database’s ability to manage data. This is where the main problem of the object model appears: the lack of a universally accepted underlying mathematical model for data manipulation. Not having a universal access standard hampers the OODM because it forces each implementation to create its own version of an **object query language (OQL)**. The OQL is the database query language used by an OODBMS. Of course, different vendors create different versions of OQL, and that, in turn, limits true interoperability. However, several groups, such as the Object Management Group (OMG, www.omg.org)², are currently working on the development of

object query language (OQL)

The database query language used by an object oriented database management system.

²Check the OMG main website (www.omg.org) for the most recent object standards developments. In spite of the fact that OMG develops standards, the authors see little evidence that the OMG standards are being widely adopted in the database modeling marketplace, which is the focus of their interest.

standards for object-oriented technology. For example, to facilitate modeling in an OO environment, the OMG has developed the Unified Modeling Language (UML) standard. UML represents an attempt to create a universal modeling notation to facilitate activities such as system development, data modeling, and network design. (See Appendix H to learn more about UML.)

Although there is no standard way to manipulate sets with an OQL, the relational model's SQL2 standard did not provide ways to manipulate objects in an object-oriented database, either. However, the mismatch between OQL and SQL was reduced with the publication of the SQL3, or SQL-99, standard in 1999 by the American National Standards Institute (ANSI). SQL3 paves the way toward the integration of object-oriented extensions within relational databases. For more information about this standard, see document number ANSI/ISO/IEC 9075, Parts 1–5, at www.ansi.org.

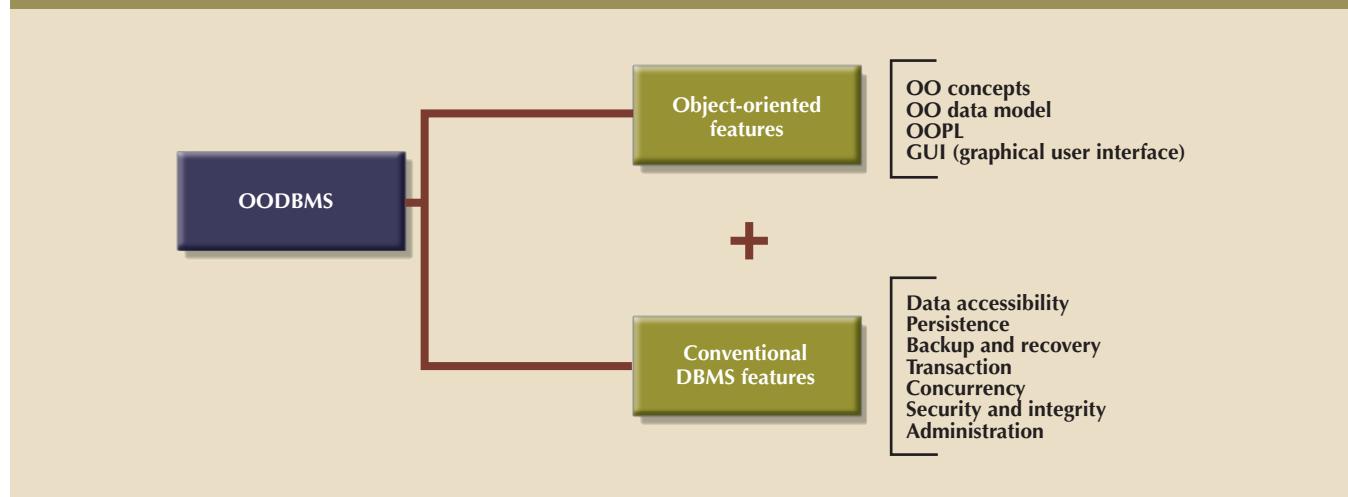
Previous sections discussed the object-oriented concepts that were derived from OOPLs. Those concepts were used to establish the characteristics of the object-oriented data model (OODM) and to study its graphical representation. This section compared the OODM to previous data models and explained that one of the major problems of the OODM is that it fails to conform to a universally accepted standard. Yet in spite of the lack of standards, there is agreement about *minimal* OODBMS characteristics. Those characteristics will be explored in the next section.

G-6 Object-Oriented Database Management Systems

During the past few years, the data management and application environment has become far more complex than the one envisioned by the creators of the hierarchical, network, or relational DBMSs. Those complex application environments may best be served by an **object-oriented database management system (OODBMS)**. The OODBMS is a database management system that integrates the benefits of typical database systems with the more powerful modeling and computational (programming) characteristics of the object-oriented data model. (See Figure G.35.)

object-oriented database management system (OODBMS)
Data management software used to manage data in an object-oriented database model.

FIGURE G.35 OBJECT-ORIENTED DATABASE MANAGEMENT SYSTEMS





Note

A DBMS based on the object model may be labeled an object-oriented database management system (OODBMS) or an object database management system (ODBMS). Given the frequent use of “OO” and “OODBMS” labels in the early stages of object-oriented research, the OODBMS label will be used here as a matter of personal preference.

OODBMS products are used to develop complex systems such as:

- Medical applications that handle digitized data such as x-rays, MRI scans, and ultrasounds, together with textual data used for medical research and patient medical history analysis.
- Financial applications in portfolio and risk management. These applications yield a real-time view of data that is based on multiple computations and aggregations applied to data acquired from complex stock transactions around the world. These applications can handle “time series” data as a user-defined data type with its own internal representation and methods.
- Telecommunications applications such as network configuration management applications that automatically monitor, track, and reconfigure communications networks based on hundreds of parameters in real time. Companies such as Ericsson, AT&T, and China Telecom use OODBMSs to support their telecommunications management applications. Motorola’s Iridium global communications system manages its complex network of satellites and ground stations using an OODBMS.
- The BaBar physics experiment at the Stanford Linear Accelerator Center, which enters 1 terabyte of data per day into an OODBMS.
- Computer-aided design (CAD) and computer-aided manufacturing (CAM). These applications make use of complex data relations as well as multiple data types.
- Computer-aided software engineering (CASE) applications, which are designed to handle very large amounts of interrelated data.
- Multimedia applications, such as geographic information systems (GIS), that use video, sound, and high-quality graphics that require specialized data-management features such as intersect, inside, within, point, line, and polygon.

Many OODBMSs use a subset of the object-oriented data model features. Therefore, those who create the OODBMS tend to select the OO features that best serve the particular OODBMS’s purpose, such as support for early or late binding of the data types and methods and support for single or multiple inheritance. Whatever the choices, the critical factor for a successful OODBMS implementation appears to be finding the best mix of OO and conventional DBMS features that will not sacrifice the benefits of either one.

G-6a Features of an Object-Oriented DBMS

As shown in Figure G.35, an OODBMS is the result of combining OO features such as class inheritance, encapsulation, and polymorphism with database features such as data integrity, security, persistence, transaction management, concurrency control, backup, recovery, data manipulation, and system tuning. The “Object-Oriented Database System Manifesto” (Atkinson et al., 1989)³ was the first comprehensive attempt to define

³Malcolm Atkinson et al., “The Object-Oriented Database System Manifesto.” This white paper first presented at the First International Conference on Deductive and Object-Oriented Databases, Kyoto, Japan, in 1989, may be downloaded from the Object Data Management Group website at www.odmg.org. Select “White papers,” then “Database,” then “OO Database System Manifesto.” You will also find a series of follow-up white papers that explore the 1997–1999 OO rule extensions and standards. The extensions and standards are designed to augment the manifesto, and none claims to replace any part thereof.

OODBMS features. The document included 13 mandatory features as well as optional characteristics of the OODBMS. The 13 rules are divided into two sets: the first eight characterize an OO system, and the last five characterize a DBMS. The 13 rules are listed in Table G.4. Each rule will be discussed briefly.

TABLE G.4

THE 13 OODBMS RULES

RULES THAT MAKE IT AN OO SYSTEM	
Rule 1	The system must support complex objects.
Rule 2	Object identity must be supported.
Rule 3	Objects must be encapsulated.
Rule 4	The system must support types or classes.
Rule 5	The system must support inheritance.
Rule 6	The system must avoid premature binding.
Rule 7	The system must be computationally complete.
Rule 8	The system must be extensible.
RULES THAT MAKE IT A DBMS	
Rule 9	The system must be able to remember data locations.
Rule 10	The system must be able to manage very large databases.
Rule 11	The system must accept concurrent users.
Rule 12	The system must be able to recover from hardware and software failures.
Rule 13	Data query must be simple.

- *Rule 1. The system must support complex objects.* It must be possible to construct complex objects from existing objects. Examples of such object constructors are sets, lists, and tuples that allow the user to define aggregations of objects as attributes.
- *Rule 2. Object identity must be supported.* The OID must be independent of the object's state. This feature allows the system to compare objects at two different levels: comparing the OID (identical objects) and comparing the object's state (equal or shallow equal objects).



Note

If the objects have different OIDs but their attribute values are equal, the objects are not identical, but they are considered to be "shallow equal." To use an analogy, identical twins are alike, yet different.

- *Rule 3. Objects must be encapsulated.* Objects have a public interface, but private implementation of data and methods. The encapsulation feature ensures that only the public aspect of the object is seen, while the implementation details are hidden.
- *Rule 4. The system must support types or classes.* This rule allows the designer to choose whether the system supports types or classes. Types are used mainly at compile time to check type errors in attribute value assignments. Classes are used to store and manipulate similar objects at execution time. In other words, class is a more dynamic concept, and type is a more static one.

- *Rule 5. The system must support inheritance.* An object must inherit the properties of its superclasses in the class hierarchy. This property ensures code reusability.
- *Rule 6. The system must avoid premature binding.* This feature allows you to use the same method's name in different classes. Based on the class to which the object belongs, the OO system decides which implementation to access at run time. This feature is also known as late binding or dynamic binding.
- *Rule 7. The system must be computationally complete.* The basic notions of programming languages are augmented by features common to the database data manipulation language (DML), thereby allowing you to express any type of operation in the language.
- *Rule 8. The system must be extensible.* The final OO feature concerns the system's ability to define new types. There is no management distinction between user-defined types and system-defined types.
- *Rule 9. The system must be able to remember data locations.* The conventional DBMS stores its data permanently on disk; that is, the DBMS displays data persistence. OO systems usually keep the entire object space in memory; once the system is shut down, the entire object space is lost. Much of the OODBMS research has focused on finding a way to permanently store objects and to retrieve them from secondary storage (disk).
- *Rule 10. The system must be able to manage very large databases.* Typical OO systems limit the object space to the amount of primary memory available. For example, Smalltalk cannot handle objects larger than 64K. Therefore, a critical OODBMS feature is the ability to optimize the management of secondary storage devices by using buffers, indexes, data clustering, and access path selection techniques.
- *Rule 11. The system must support concurrent users.* Conventional DBMSs are especially capable in this area. The OODBMS must support the same level of concurrency as conventional systems.
- *Rule 12. The system must be able to recover from hardware and software failures.* The OODBMS must offer the same level of protection from hardware and software failures that the traditional DBMS provides; that is, the OODBMS must provide support for automated backup and recovery tools.
- *Rule 13. Data query must be simple.* Efficient querying is one of the most important features of any DBMS. Relational DBMSs have provided a standard database query method through SQL, and the OODBMS must provide an object query language (OQL) with similar capability.

Optional OODBMS features include:

- *Support for multiple inheritance.* Multiple inheritance introduces greater complexity by requiring the system to manage potentially conflicting properties between classes and subclasses.
- *Support for distributed ODBMSs.* The trend toward systems application integration constitutes a powerful argument in favor of distributed databases. If the OODBMS is to be integrated seamlessly with other systems through networks, the database must support some degree of distribution.
- *Support for versioning.* Versioning is a new characteristic of the OODBMS that is especially useful in applications such as CAD and CAM. Versioning allows you to maintain a history that tracks all object transformations. Therefore, you can browse through all of the different object states, in effect letting you walk back and forth in time.

G-6b Oracle Object Examples

Oracle databases (since version 8) support object-oriented extensions. The extensions allow users to create object types, using DDL commands. Those object types are the equivalent of classes (see Section G-3f) or abstract data types (ADT) in the object model (see Section G-3j). Oracle supports various object types:

- *Column type.* This object type provides data type extensibility by allowing the user to define his/her own data types. A column object type is the equivalent of an abstract data type (ADT). An ADT can be used when defining a column data type within a relational table. An ADT can also have methods associated with it; those methods are implemented using PL/SQL or C++ or Java.
- *Row type.* A row type object is used to define an object table object. An **object table** is the equivalent of a relational table composed of many rows where each row is an object of the same type. Each row object has a unique system-generated object ID (OID), or object identifier. (See Section G-3b.)
- *Collection objects.* Oracle also provides support for two types of collection objects. (Note the discussion that accompanies Figure G.29 in Section G-4c.)
 - *Variable length arrays (VARRAY).* Enables the user to create an object type as an array of objects of a given type.
 - *Nested tables.* Allows the creation of a relational table in which one of the attributes is a table. Specifically, a relational table contains an attribute with an object table type.

To illustrate creating and using various object types, Oracle 11g will be used in the following discussion.

Column Type A column type is basically a new data type (abstract data type) you can use when you define an attribute in a table. By creating a column type, you are defining a new class with shared attributes and methods. To illustrate the use of column types, let's create two column types named T_ADDRESS and T_JOB. The T_ADDRESS column type will contain the street, city, state, and zip code attributes. The T_JOB column type will be used to store data about a job (company name, start date, end date, and monthly salary). The T_JOB column type will have two methods: *monthsonjob*, which returns the number of months spent in a given job, and *totalearned*, which returns the result of the multiplication of the number of months employed and the monthly salary. Figure G.36 shows the use of the CREATE TYPE command to create the two column types.

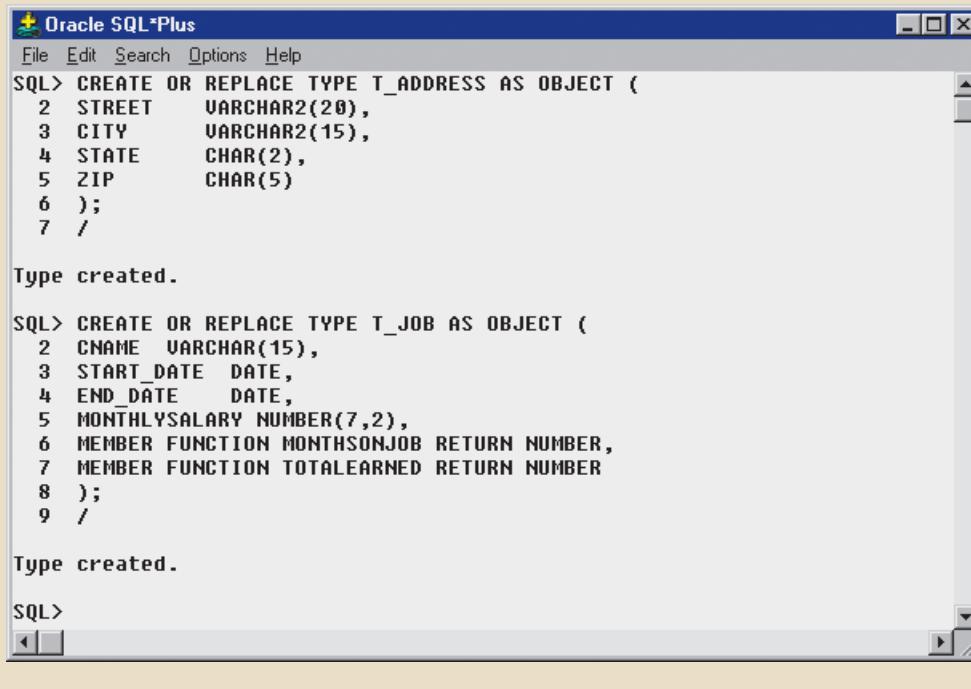
In Figure G.36, note that the T_JOB column data type creation command includes references to the methods that are to be created. By using the MEMBER FUNCTION clause, you define the name of the method to be created, any optional parameters that may be required (shown in parentheses), and the type of value (such as number or character) to be returned. To actually create the methods, you use the CREATE TYPE BODY command, as shown in Figure G.37.

As you examine Figure G.37, note that the method definition uses standard PL/SQL commands. You might also define methods using other languages, such as C++ or Java. Each method definition starts with the MEMBER FUNCTION keywords. The actual method code is contained within the BEGIN and END clauses.

object table

The equivalent of a relational table composed of many rows, where each row is an object of the same type. Each row object has a unique system generated object ID (OID) or object identifier.

FIGURE G.36 CREATION OF THE T_ADDRESS AND T_JOB COLUMN DATA TYPES



The screenshot shows the Oracle SQL*Plus interface with the following SQL code:

```

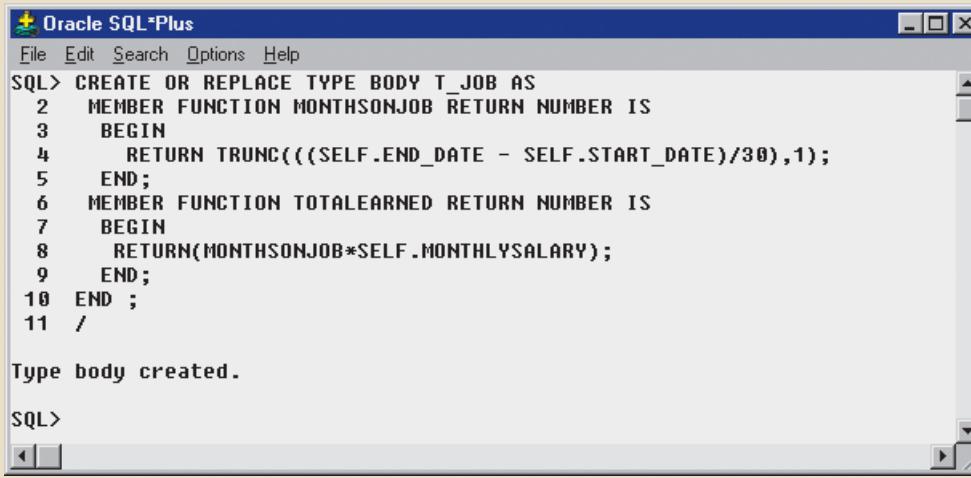
SQL> CREATE OR REPLACE TYPE T_ADDRESS AS OBJECT (
  2 STREET      VARCHAR2(20),
  3 CITY        VARCHAR2(15),
  4 STATE       CHAR(2),
  5 ZIP         CHAR(5)
  6 );
  7 /
Type created.

SQL> CREATE OR REPLACE TYPE T_JOB AS OBJECT (
  2 CNAME        VARCHAR(15),
  3 START_DATE   DATE,
  4 END_DATE     DATE,
  5 MONTHLYSALARY NUMBER(7,2),
  6 MEMBER FUNCTION MONTHSONJOB RETURN NUMBER,
  7 MEMBER FUNCTION TOTALEARNED RETURN NUMBER
  8 );
  9 /
Type created.

SQL>

```

FIGURE G.37 CREATION OF THE T_JOB METHODS



The screenshot shows the Oracle SQL*Plus interface with the following SQL code:

```

SQL> CREATE OR REPLACE TYPE BODY T_JOB AS
  2 MEMBER FUNCTION MONTHSONJOB RETURN NUMBER IS
  3 BEGIN
  4   RETURN TRUNC(((SELF.END_DATE - SELF.START_DATE)/30),1);
  5 END;
  6 MEMBER FUNCTION TOTALEARNED RETURN NUMBER IS
  7 BEGIN
  8   RETURN(MONTHSONJOB*SELF.MONTHLYSALARY);
  9 END;
 10 END ;
 11 /
Type body created.

SQL>

```

Figure G.38 shows the creation of a WORKER table that uses the T_ADDRESS and T_JOB column types defined earlier.

FIGURE G.38 CREATION OF THE WORKER TABLE, USING T_ADDRESS AND T_JOB COLUMN TYPES

```

Oracle SQL*Plus
File Edit Search Options Help
SQL> CREATE TABLE WORKER(
 2  WRK_NUM      NUMBER PRIMARY KEY,
 3  WRK_LNAME    VARCHAR2(15) NOT NULL,
 4  WRK_FNAME    VARCHAR2(15) NOT NULL,
 5  WRK_ADDRESS  T_ADDRESS,
 6  WRK_PREUJOB T_JOB);

Table created.

SQL>

```

Once you have created the table, you use standard SQL commands to insert data. However, to enter data in a column type attribute, you must use the column type name, as indicated in Figure G.39, with the INSERT statements.

FIGURE G.39 WORKING WITH COLUMN TYPES IN THE WORKER TABLE

```

Oracle SQL*Plus
File Edit Search Options Help
SQL> INSERT INTO WORKER
 2  VALUES (1, 'VILLEGAS', 'ROBERTO',
 3           T_ADDRESS('123 Main St.', 'London', 'OH', '76987'),
 4           T_JOB('GLOBALCOM', '01-AUG-2005', '03-DEC-2008', 3000));

1 row created.

SQL> INSERT INTO WORKER
 2  VALUES (2, 'GORTTY', 'JANE',
 3           T_ADDRESS('453 Sulla St.', 'London', 'OH', '76987'),
 4           T_JOB('SURESTART', '15-MAY-2006', '15-OCT-2008', 2500));

1 row created.

SQL> INSERT INTO WORKER
 2  VALUES (3, 'SMITH', 'MARGO',
 3           T_ADDRESS('7854 Court Av.', 'London', 'OH', '76984'),
 4           T_JOB('TELCO-R-US', '01-JUL-2007', '15-SEP-2008', 4000));

1 row created.

SQL> SELECT WRK_LNAME || ', ' || WRK_FNAME AS NAME,
 2       W.WRK_ADDRESS.STREET AS STREET,
 3       W.WRK_ADDRESS.CITY AS CITY,
 4       W.WRK_ADDRESS.STATE AS STATE,
 5       W.WRK_PREUJOB.MONTHSONJOB() AS MONTHS,
 6       W.WRK_PREUJOB.TOTALEARNED() AS TOTAL
 7  FROM WORKER W;

NAME                      STREET          CITY      ST   MONTHS    TOTAL
VILLEGAS, ROBERTO        123 Main St.    London    OH    40.6    121800
GORTTY, JANE              453 Sulla St.   London    OH    29.4    73500
SMITH, MARGO              7854 Court Av.  London    OH    14.7    58800

SQL>

```

To retrieve data from a column type attribute using a SELECT statement, you must first declare an alias for the table. In this case, the alias W has been used. Next, you can refer to a column type attribute or method using a dot-separated notation such as W.WRK_ADDRESS.STREET or W.WRK_PREVJOB.TOTALEARNED(), as shown in Figure G.39.

Row Type A row type enables you to create a table in which each row is an object instance. That table is called an object table to differentiate it from a relational table. To demonstrate the use of row types, let's create the OTBL_BAND object table in which each row is a Musician object. To accomplish that task, let's first create a T_MUSICIAN column type. (Remember that a column type is an object.) The T_MUSICIAN column type will have an AGE method that uses the system date and the musician's date of birth to return the age of each musician. To create the object table, you use the CREATE TABLE OF command, as shown in Figure G.40.

FIGURE G.40 CREATION OF THE OTBL_BAND OBJECT TABLE

The screenshot shows the Oracle SQL*Plus interface with the title bar "Oracle SQL*Plus". The menu bar includes "File", "Edit", "Search", "Options", and "Help". The main window displays the following SQL commands:

```

SQL> CREATE TYPE T_MUSICIAN AS OBJECT (
  2   NAME          VARCHAR(15),
  3   DOB           DATE,
  4   INSTRUMENT    CHAR(8),
  5   MEMBER FUNCTION AGE RETURN NUMBER
  6 );
  7 /
Type created.

SQL> CREATE OR REPLACE TYPE BODY T_MUSICIAN AS
  2   MEMBER FUNCTION AGE RETURN NUMBER IS
  3     BEGIN
  4       RETURN TRUNC(((SYSDATE - SELF.DOB)/365),1);
  5     END;
  6   END ;
  7 /
Type body created.

SQL> CREATE TABLE OTBL_BAND OF T_MUSICIAN OBJECT ID IS SYSTEM GENERATED;
Table created.

SQL> |

```

To insert data to the newly created object table, you use the INSERT command, as shown in Figure G.41. Note that you do not have to identify the column type, as was required with the WORKER table. The difference is that the WORKER table is a relational table containing attributes with abstract data types. In such cases, you need to specify the abstract data (column type). In the case of the OTBL_BAND table, you are adding rows to an *object table*. However, you still must use an alias to invoke a method. (See Figure G.41.)

FIGURE G.41 WORKING WITH THE OTBL_BAND OBJECT TABLE

```

Oracle SQL*Plus
File Edit Search Options Help
SQL> INSERT INTO OTBL_BAND
2  VALUES ('TOM JONES', '01-AUG-1943', 'SINGER');
1 row created.

SQL> INSERT INTO OTBL_BAND
2  VALUES ('FLEETWOD MAC', '15-MAY-1940', 'DRUMS');
1 row created.

SQL> INSERT INTO OTBL_BAND
2  VALUES ('JIMY HENDRIX', '01-JUL-1928', 'GUITAR');
1 row created.

SQL> SELECT NAME, DOB, INSTRUMENT, B.AGE() AS AGE
2  FROM  OTBL_BAND B;
NAME          DOB        INSTRUME      AGE
-----        -----        -----
TOM JONES     01-AUG-43  SINGER         66
FLEETWOD MAC  15-MAY-40  DRUMS          69.2
JIMY HENDRIX   01-JUL-28  GUITAR         81.1

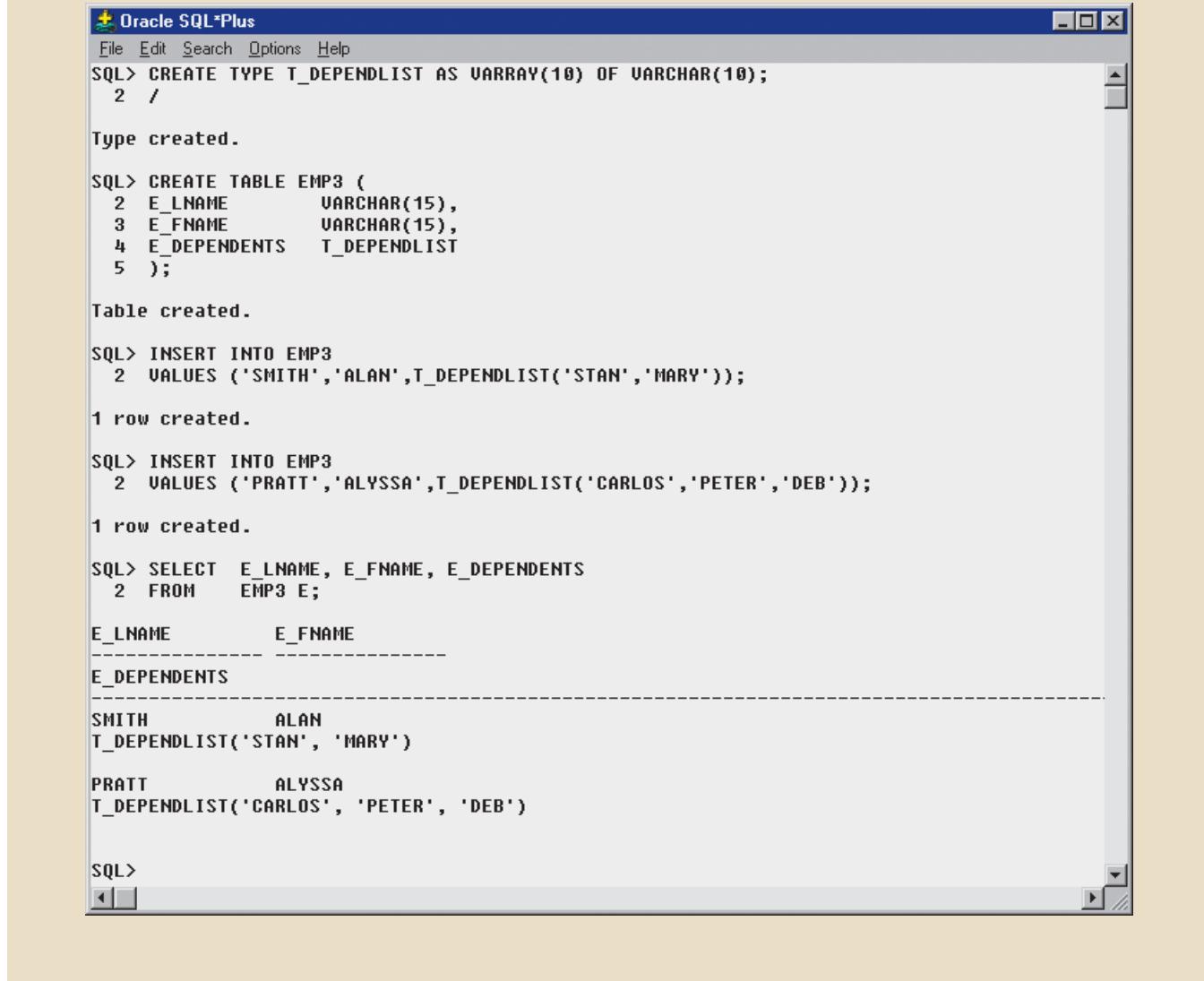
SQL> |

```

VARRAY Collection Type The variable length array creates a new object type that represents a collection of objects of a similar type (objects or base data types). For example, an employee may have multiple dependents. In that case, you can store all of the dependents in an array for each of the employees. Figure G.42 shows the commands required to create the T_DEPENDLIST variable array object type and the EMP3 table containing the E_DEPENDENTS attribute, which uses the T_DEPENDLIST data type. Note that the variable array has been defined to hold a maximum of 10 dependent names.

Nested Table Collection Type When you have related data that are more extensive than you would expect to find in an array, you can use a nested table. A nested table is created when an attribute within a relational table definition (CREATE TABLE) is assigned a table data type. For example, Figure G.43 shows the creation of the EMP4 table containing the E_DEPENDTS attribute, which uses the T_DEPTAB data type. In turn, the T_DEPTAB data type is defined as a table type. Conceptually speaking, the attribute is, in effect, a table.

FIGURE G.42 CREATING AND WORKING WITH THE VARRAY OBJECT TYPE



The screenshot shows a window titled "Oracle SQL*Plus". The menu bar includes "File", "Edit", "Search", "Options", and "Help". The main area displays the following SQL session:

```

SQL> CREATE TYPE T_DEPENDLIST AS VARRAY(10) OF VARCHAR(10);
2 /

Type created.

SQL> CREATE TABLE EMP3 (
2   E_LNAME      VARCHAR(15),
3   E_FNAME      VARCHAR(15),
4   E_DEPENDENTS T_DEPENDLIST
5 );

Table created.

SQL> INSERT INTO EMP3
2   VALUES ('SMITH','ALAN',T_DEPENDLIST('STAN','MARY'));

1 row created.

SQL> INSERT INTO EMP3
2   VALUES ('PRATT','ALYSSA',T_DEPENDLIST('CARLOS','PETER','DEB'));

1 row created.

SQL> SELECT   E_LNAME, E_FNAME, E_DEPENDENTS
2   FROM     EMP3 E;

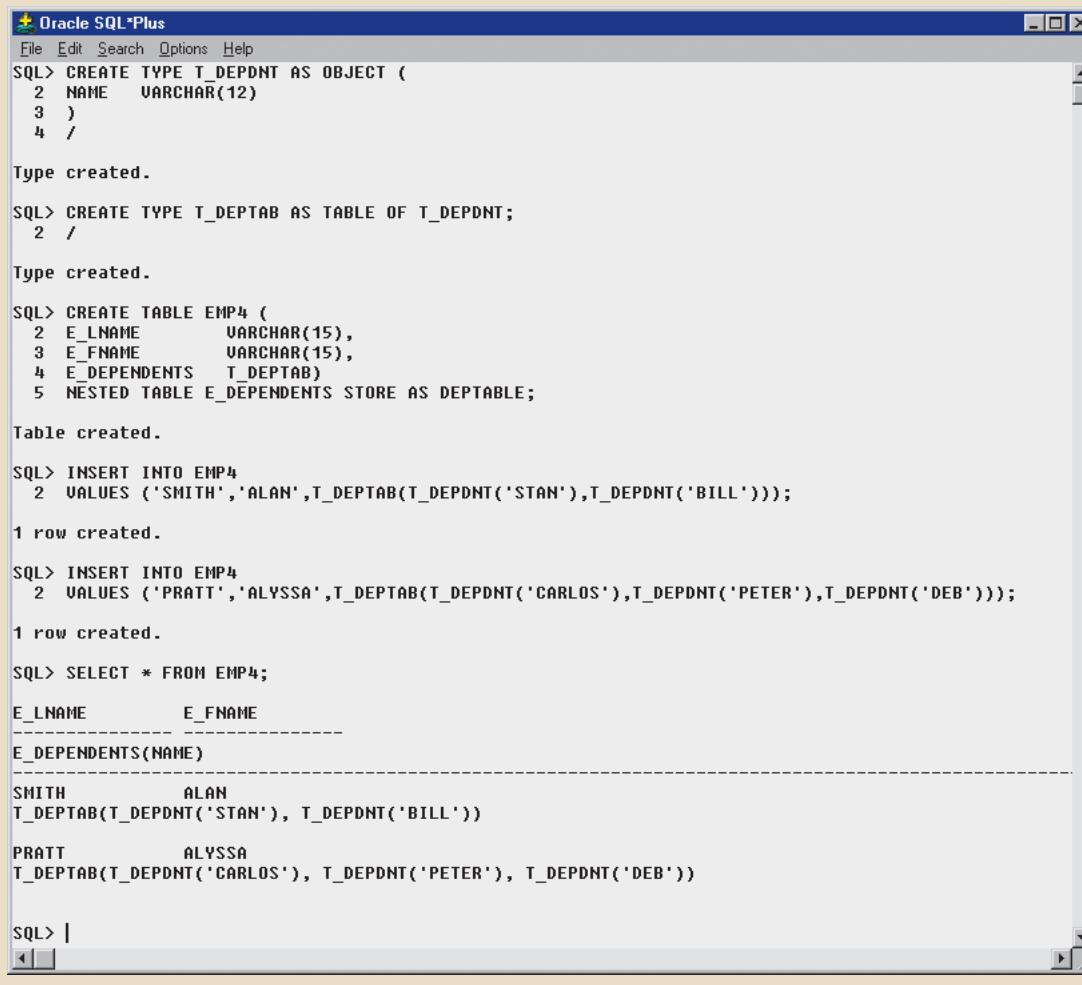
E_LNAME      E_FNAME
-----
E_DEPENDENTS
-----
SMITH        ALAN
T_DEPENDLIST('STAN', 'MARY')

PRATT        ALYSSA
T_DEPENDLIST('CARLOS', 'PETER', 'DEB')

SQL>

```

FIGURE G.43 CREATING AND WORKING WITH A NESTED TABLE OBJECT TYPE



The screenshot shows a session in Oracle SQL*Plus. The user creates a nested table object type T_DEPDNT, then creates a table EMP4 with a column E_DEPENDENTS of type T_DEPTAB, which stores instances of T_DEPDNT. Data is inserted into EMP4, and a select statement retrieves the data.

```

Oracle SQL*Plus
File Edit Search Options Help
SQL> CREATE TYPE T_DEPDNT AS OBJECT (
 2   NAME  VARCHAR(12)
 3 )
 4 /
Type created.

SQL> CREATE TYPE T_DEPTAB AS TABLE OF T_DEPDNT;
 2 /
Type created.

SQL> CREATE TABLE EMP4 (
 2   E_LNAME      VARCHAR(15),
 3   E_FNAME      VARCHAR(15),
 4   E_DEPENDENTS T_DEPTAB)
 5 NESTED TABLE E_DEPENDENTS STORE AS DEPTABLE;

Table created.

SQL> INSERT INTO EMP4
 2   VALUES ('SMITH','ALAN',T_DEPTAB(T_DEPDNT('STAN'),T_DEPDNT('BILL')));
1 row created.

SQL> INSERT INTO EMP4
 2   VALUES ('PRATT','ALYSSA',T_DEPTAB(T_DEPDNT('CARLOS'),T_DEPDNT('PETER'),T_DEPDNT('DEB')));

1 row created.

SQL> SELECT * FROM EMP4;
E_LNAME      E_FNAME
-----
E_DEPENDENTS(NAME)
-----
SMITH          ALAN
T_DEPTAB(T_DEPDNT('STAN'), T_DEPDNT('BILL'))

PRATT          ALYSSA
T_DEPTAB(T_DEPDNT('CARLOS'), T_DEPDNT('PETER'), T_DEPDNT('DEB'))

```

G-7 How Object Orientation Affects Database Design

A conventional relational database design process involves the application of ER modeling and normalization techniques to develop and implement a design. During a design process, emphasis is placed on modeling real-world objects through simple tabular relations, usually presented in 3NF. Unfortunately, as you have already seen, sometimes the relational and ER models cannot adequately represent some objects. Consequently, the ER model makes use of constructs such as bridge (composite) entities that widen the semantic gap between the real-world objects and their corresponding representations.

You may have noticed the database design process generally focusing on identification of the data elements, rather than including data operations as part of the process. In fact, the definition of data constraints and data transformations is usually considered late in the database design process. Those definitions are implemented by external application program code. In short, operations are not a part of the database model.

Why does the conventional model tolerate and even require the existence of the data/procedures dichotomy? After all, the idea of object-oriented design had been contemplated even in the classical database environment. The reason is simple: until recently, database designers simply had no access to tools that bonded data and procedures.

The object-oriented database design approach answers the problem of a split between data and procedures by providing both data identification and the procedures or manipulations to be performed on the data. Object-oriented database design forces you to think of data and procedures as a self-contained entity. Specifically, the OO design requires the database description to include the objects and their data representation, constraints, and operations. That design can produce a more complete and meaningful description of the database than was possible in the conventional database design.

OO design is iterative and incremental in nature. The database designer identifies each real-world object and defines its internal data representation, semantic constraints, and operations. Next, the designer groups similar objects in classes and implements the constraints and operations through methods. At this point, the designer faces two major challenges:

1. Build the class hierarchy or the class lattice (if multiple inheritance is allowed), using base data types and existing classes. This task will define the superclass-subclass relationships.
2. Define the interclass relationships (attribute-class links), using both base data types and ADTs.

The importance of those tasks can hardly be overestimated because the better the use of the class hierarchy and the treatment of the interclass relationships, the more flexible and closer to the real world the final model will be.

Code reusability does not come easy. One of the hardest tasks in OODB design is creation of the class hierarchy, using existing classes to construct new ones. Future DBAs will have to develop specialized skills to perform that task properly and to incorporate code that represents data behavior. Thus, DBAs are likely to become surrogate database programmers who must define data-intrinsic behavior. The role of DBAs is likely to change when they take over some of the programming burden of defining and implementing operations that affect the data.

Both DBAs and designers face additional problems. In contrast to the relational or ER design processes, there are few computerized OODB design tools, and if the design is to be implemented in any of the conventional DBMSs, it must be translated carefully. The reason is that conventional databases do not support abstract data types, non-normalized data, encapsulation, or other OO features.

As is true in any of the object-oriented technologies, the lack of standards also affects OO database design. There is neither a widely accepted standard methodology to guide the design process, nor a set of rules (like the normalization rules in the relational model) to evaluate the design. This situation is improving. The Object Management Group (OMG), mentioned earlier, produces vendor-independent standards and specifications for object-based systems and components. The OMG created the Unified Modeling Language (UML), a graphical language for the modeling, design, and visualization of object-oriented systems. UML is used to model not only the database component of a system but also its processes, modules, and network components and the interaction among them. OMG also created object standards that define the Object Management Architecture (OMA), which allows the interoperation of objects across diverse systems and platforms. The OMA standard includes the Common Object Request Broker Architecture (CORBA) and the Common Object Services Specifications (COSS). That framework is used by OODBMS vendors and developers to implement systems that are highly interoperable with other OODBMSs, as well as with RDBMSs and older DBMS systems.



Note

Appendix H provides an introduction to the Unified Modeling Language (UML).

Some vendors are already offering products that comply with the OMG's CORBA and COSS specifications, such as IBM's System Object Model (SOM) and HP's Object Request Broker (ORB). Other object architectures have emerged as alternatives to the concerted standards efforts, especially Microsoft's object linking and embedding (OLE) and Component Object Model (COM). Although the OLE/COM specification is not a standards-based effort, the sheer established market volume is making it the de facto object standard for the Microsoft Windows environment.

G-8 OODBMS: Advantages and Disadvantages

Compared to the RDBMS market share, OODBMSs have a long way to go before they can claim double-digit market percentage. In fact, at this point, the OODBMS occupies a strong niche market, as the Apple Mac does in the microcomputing arena. As with the Mac's impact on microcomputing, the OODBMS has been the vehicle for technological innovation, but it has not been the beneficiary of market share growth based on its technological innovations. Yet in spite of its lack of market share, the OODBMS is worth examining, especially because its OO features drive the changes in database technology that define today's object relational DBMS.

One reason for the OODBMS's lack of market acceptance is that the RDBMS has incorporated many OO features while retaining its conceptual simplicity, thus diminishing the OODBMS's allure. Nevertheless, as long as the RDBMS does not incorporate C. J. Date's recommended domain implementation, the OODBMS offers benefits that are worth examining. Most of those benefits are expressed in terms of the complex object management capabilities you have explored in some detail. To obtain those benefits, the OODBMS depends on the use of an OOPL. That is why you examine some of the OODBMS's benefits with reference to programming issues.

G-8a Advantages

- OODBMSs allow the inclusion of more semantic information in the database, thus providing a more natural and realistic representation of real-world objects.
- OODBMSs provide an edge in supporting complex objects, which makes them especially desirable in specialized application areas. Conventional databases simply lack the ability to provide efficient applications in CAD, CAM, medical imaging, spatial imaging, and specialized multimedia environments.
- OODBMSs permit the extensibility of base data types, thereby increasing both the database functionality and its modeling capabilities.
- *If the platform allows efficient caching*, when managing complex objects, OODBMSs provide dramatic performance improvements compared to relational database systems.
- Versioning is a useful feature for specialized applications such as CAD, CAM, medical imaging, spatial imaging, engineering, text management, and desktop publishing.
- The reusability of classes allows for faster development and easier maintenance of the database and its applications.

- Faster application development time is obtained through inheritance and reusability. This benefit is obtained only after mastering the use of OO development features such as:
 - Proper use of the class hierarchy; for example, how to use existing classes to create new classes.
 - OO design methodology.
- The OODBMS provides a possible solution to the problem of integrating existing and future DBMSs into a single environment. This solution is based on the OODBMS's strong data-abstraction capabilities and its promise of portability.

G-8b Disadvantages

- OODBMSs face strong and effective opposition from the firmly established RDBMSs, especially when those RDBMSs—such as IBM's DB2 Universal Database and Oracle—incorporate many OO features that would otherwise have given the OODBMS the clear competitive edge in a complex data environment. Therefore, the OODBMS's design and implementation complexities become more difficult to justify.
- The OODBMS is based on the object model, which lacks the solid theoretical foundation of the relational model on which the RDBMS is built.
- In some sense, OODBMSs are considered a throwback to the traditional pointer systems used by hierarchical and network models. This criticism is not quite true when it associates the pointer system with the navigational data manipulation style and fixed access paths that led to the relational system's dominance. Nevertheless, the *complexity* of the OODBMS pointer systems cannot be denied.
- OODBMSs do not provide a standard ad hoc query language, as relational systems do. At this point, development of the object query language (OQL) is far from complete. Some OODBMS implementations are beginning to provide extensions to the relational SQL to make the integration of the OODBMS and RDBMS possible.
- The relational DBMS provides a comprehensive solution to business database design and management needs, supplying both a data model and a set of fairly straightforward normalization rules for designing and evaluating relational databases. OODBMSs do not yet provide a similar set of tools.
- The initial learning curve for the OODBMS is steep. If you consider the direct training costs and the time it takes to fully master the uses and advantages of object orientation, you will appreciate why OODBMSs seldom are rated as the first option when solutions are sought for noncomplex business-oriented problems.
- The OODBMS's low market presence, combined with its steep learning curve, means that few people are qualified to make use of the presumed power of OO technology. Most of the technology is currently focused on engineering application areas of software development. Therefore, only companies with the right mix of resources (money, time, and qualified personnel) can afford to invest in OO technology.
- The lack of compatibility between different OODBMSs makes switching from one piece of software to another very difficult. With RDBMSs, different products are very similar, and switching from one to another is relatively easy.

A few years ago, the authors speculated that future systems would manage objects with embedded data and methods, rather than with records, tuples, or files. The authors also suggested that although the portability details were not clear yet, they would have a major and lasting impact on how databases would be designed and used. Given the benefit of hindsight, the authors now know that the OODBMS's

reach has been limited by the object-relational DBMS's successful integration of many OO concepts. In any case, the OODBMS has had a major impact on how databases are viewed and managed, and the battle of the relational and object titans is far from over. Finally, because the object concepts are likely to remain the focus for future DBMS developments, they continue to be worth understanding.

G-9 How OO Concepts Have Influenced the Relational Model

Most relational databases are designed to serve general business applications that require ad hoc queries and easy interaction. The data types encountered in those applications are well defined and are easily represented in common tabular formats with equally common short and well-defined transactions. However, RDBMSs are not as well suited as OODBMSs to the complex requirements of some applications, and the RDBMS is beginning to reach its limits in a business data environment that is changing with the advent of mixed-media data storage and retrieval.

The fast-changing data environment has forced relational model advocates to respond to the OO challenge by extending the relational model's conceptual reach. The result of their efforts is usually referred to as the extended relational model (ERM) or, more appropriately, the object/relational model (O/RM). Although this O/RM effort is still a work in progress, its basic features provide support for:

- Extensibility of new user-defined (abstract) data types
- Complex objects
- Inheritance
- Procedure calls (rules or triggers)
- System-generated identifiers (OID surrogates)

That is not an exhaustive list of the extensions added to the relational model, nor do all extended relational models incorporate all of the listed additions. However, the list contains the most crucial and desirable extended relational features.



Note

It's worth noting again that C. J. Date's "Third Manifesto" is based on Date's observation that the relational model already contains the desired capabilities through its support of domains. Therefore, the implementation of that domain support will yield the benefits now claimed for the OO "extensions" of the relational model. However, the relational domain implementations have not (yet?) been developed commercially, while the OO "extensions" to the relational database model are a commercial fact of life.

The enhancements to the relational model are based on the following concepts:

- Semantic and object-oriented concepts are necessary to support the new generation of applications—especially if those applications will be deployed through the Internet.
- The concepts can and must be added to the relational model.
- The benefits of the relational model must be preserved to protect the investment in relational technology and to provide downward compatibility.

Most current extended relational DBMSs conform to the notions expressed in C. J. Date's "Third Manifesto." (See preceding note.) They also provide the following useful features:

- Oracle Corporation and IBM have developed suites of products marketed as Universal Database Servers. Although the Universal Database Server is not a pure object-oriented DBMS—it lacks the object storage component—this product supports complex data types such as multimedia data and spatial data, and it's Internet-ready. The Internet feature allows users to query the database using the World Wide Web (WWW). Oracle also includes support for object-oriented extensions and storage. IBM's DB2 Universal Database Server has similar capabilities.
- IBM's DB2 Universal Database system is a proven database that is used by many Fortune 1000 corporations. IBM's system supports digitized data (video and audio) as well as user-defined data types and procedures. The Universal Database is also being positioned as a key player in the Internet arena with its support for web access and Java programming interfaces.

G-10 The Next Generation of Database Management Systems

The adaptation of OO concepts in several computer-related areas has changed both systems design and system behavior. The next generation of DBMSs is likely to incorporate features borrowed from object-oriented database systems, artificial intelligence systems, expert systems, distributed databases, and the Internet.

OODBMSs represent only one step toward the next generation of database systems. The use of OO concepts will enable future DBMSs to handle more complex problems with normalized and non-normalized data. The extensibility of database systems is one of the many major object-oriented contributions that enable databases to support new data types such as sets, lists, arrays, video, bitmap pictures, voice, and maps. The SQL3 standard provides such extensibility by supporting user-defined data types in addition to its predefined data types (numeric, integer, string, and so on). For example, in SQL3, a DBA can create a new abstract data type that represents a collection of objects, then use that data type in a table definition. That procedure enables a database column to contain a *collection* of values instead of a single value.

Recent market history indicates that the OODBMS will probably continue to occupy a niche within the database market. That niche will be characterized by applications that require very large amounts of data with several complex relations and with specialized data types. For example, the OODBMS seems likely to maintain its standing in CAD, CAM, computer-integrated manufacturing, specialized multimedia applications, medical applications, architectural applications, mapping applications, simulation modeling, and scientific applications.

However, current market conditions seem to dictate that the object/relational databases will become dominant in most complex business applications. That conclusion is based on the need to maintain compatibility with existing systems, the universal acceptance of the relational model as a standard, and the sheer weight of the relational database's considerable market share.

Key Terms

abstract data type (ADT), G-14	inheritance, G-10	object-oriented programming (OOP), G-2
associative object, G-15	instance variables, G-4	object-oriented programming languages (OOPLs), G-2
base data types, G-4	interobject relationship, G-20	
class, G-8	interrogate, G-7	object query language (OQL), G-30
class hierarchy, G-9	intersection class, G-23	object space (object schema), G-17
class instance, G-8	late binding, G-26	object state, G-6
class lattice, G-9	message, G-7	object table, G-35
collection object, G-5	method, G-6	polymorphism, G-13
complex object, G-14	multiple inheritance, G-11	protocol, G-9
composite object, G-15	object, G-3	referential object sharing, G-18
compound object, G-15	object ID (OID), G-4	simple object, G-15
conventional data types, G-4	object instance, G-8	single inheritance, G-11
domain, G-4	object orientation, G-2	subclasses, G-9
early binding, G-26	object-oriented data model (OODM), G-15	superclass, G-9
encapsulation, G-7	object-oriented database management system (OODBMS), G-31	versioning, G-28
extensible, G-15		
hybrid object, G-15		

Review Questions

1. Discuss the evolution of object-oriented concepts. Explain how those concepts have affected computer-related activities.
2. How would you define object orientation? What are some of its benefits? How are OO programming languages related to object orientation?
3. Define and describe the following:
 - a Object
 - b Attributes
 - c Object state
 - d Object ID (OID)
4. Define and contrast the concepts of method and message. What OO concept provides the differentiation between a method and a message? Give examples.
5. Explain how encapsulation provides a contrast to traditional programming constructs such as record definition. What benefits are obtained through encapsulation? Give an example.
6. Using an example, illustrate the concepts of class and class instances.

7. What is a class protocol, and how is it related to the concepts of methods and classes? Draw a diagram to show the relationships among these OO concepts: object, class, instance variables, methods, object state, object ID, behavior, protocol, and messages.
8. Define the concepts of class hierarchy, superclasses, and subclasses. Explain the concept of inheritance and the different types of inheritance. Use examples in your explanations.
9. Define and explain the concepts of method overriding and polymorphism. Use examples in your explanations.
10. Explain the concept of abstract data types. How do they differ from traditional or base data types? What is the relationship between a type and a class in OO systems?
11. What are the five minimum attributes of an OO data model?
12. Describe the difference between early and late binding. How does each of these affect the object-oriented data model? Give examples.
13. What is an object space? Using a graphic representation of objects, depict the relationship(s) that exist between a student taking several courses and a course taken by several students. What type of object is needed to depict that relationship?
14. Compare and contrast the OODM with the ER and relational models. How is a weak entity represented in the OODM? Give examples.
15. Name and describe the 13 mandatory features of an OODBMS.
16. What are the advantages and disadvantages of an OODBMS?
17. Explain how OO concepts affect database design. How does the OO environment affect the DBA's role?
18. What are the essential differences between the relational database model and the object database model?
19. Using a simple invoicing system as your point of departure, explain how its representation in an entity relationship model (ERM) differs from its representation in an object data model (ODM). (*Hint:* See Figure G.34.)
20. What are the essential differences between an RDBMS and an OODBMS?
21. Discuss the object/relational model's characteristics.

Problems

1. Convert the following relational database tables to the equivalent OO conceptual representation. Explain each of your conversions with the help of a diagram. (*Note:* The RRE Trucking Company database includes the three tables shown in Figure PG.1.)
2. Using the tables in Figure PG.1 as a source of information:
 - a. Define the implied business rules for the relationships.
 - b. Using your best judgment, choose the type of participation of the entities in the relationship (mandatory or optional). Explain your choices.
 - c. Develop the conceptual object schema.

FIGURE PG.1 THE RRE TRUCKING COMPANY DATABASE

Table name: TRUCK

Database name: RRE_Trucking

VEHICLE_NUM	BASE_CODE	TYPE_CODE	VEHICLE_MILES	VEHICLE_BUY_DATE	VEHICLE_VIN
5001	101	1	162123.50	08-Nov-03	AA-322-12212-W11
5002	102	1	276984.30	23-Mar-01	AC-342-22134-Q23
5003	101	2	212346.60	27-Dec-02	AC-445-78656-Z99
5004	101	1	99894.30	21-Feb-03	WQ-112-23144-T34
5005	103	2	245673.10	15-Apr-02	FR-998-32245-W12
5006	101	6	293245.70	30-Aug-01	AD-456-00845-R45
5007	102	3	132012.30	01-Dec-02	AA-341-96573-Z84
5008	102	3	144213.60	21-Sep-02	DR-559-22189-D33
5009	103	2	80932.90	16-Jan-04	DE-887-98456-E94
5010	104	1	34213.40	12-Apr-05	FD-221-21100-F32
5011	101	1	42326.80	09-Nov-05	DT-324-04056-H22
5012	104	6	152339.40	23-May-03	GF-657-22134-K48
5013	101	3	298145.80	11-Apr-01	HR-344-54560-J92
5014	104	2	8122.20	09-Sep-03	RW-289-38956-H87
5015	103	1	154667.90	26-Mar-03	HH-231-55498-K37
5016	105	1	1200.60	18-Mar-06	FR-332-23459-G55
5017	105	1	3345.50	23-Feb-06	DD-545-78896-X39

Table name: BASE

BASE_CODE	BASE_CITY	BASE_STATE	BASE_AREA_CODE	BASE_PHONE	BASE_MANAGER
101	Nashville	TN	615	123-4567	Andrea D. Gallagher
102	Lexington	KY	606	234-5678	George H. Delarosa
103	Kansas City	MO	573	345-6789	Maria J. Talindo
104	Athens	GA	901	456-7890	Peter F. McAfee
105	Gainesville	FL	904	678-6543	Lee A. Chau

Table name: TYPE

TYPE_CODE	TYPE_DESCRIPTION
1	Single box, double-axle
2	Single box, single-axle
3	Tandem trailer, single-axle
4	Tandem trailer, double-axle
5	Utility
6	Dump truck, single-axle
7	Dump truck, double-axle

3. Using the data presented in Problem 1, develop an object space diagram representing the object's state for the instances of Truck listed below. Label each component clearly with proper OIDs and attribute names.
- The instance of the class Truck with TRUCK_NUM = 5001.
 - The instances of the class Truck with TRUCK_NUM = 5003 and 5004.
4. Given the information in Problem 1, define a superclass Vehicle for the Truck class. Redraw the object space you developed in Problem 3, taking into consideration the new superclass that you just added to the class hierarchy.
5. Assume the following business rules:
- A course contains many sections, but each section has only one course.
 - A section is taught by one professor, but each professor may teach one or more different sections of one or more courses.
 - A section may contain many students, and each student is enrolled in many sections, but each section belongs to a different course. (Students may take many courses, but they cannot take many sections of the same course.)

- Each section is taught in one room, but each room may be used to teach several different sections of one or more courses.
- A professor advises many students, but a student has only one advisor.

Based on those business rules:

- a. Identify and describe the main classes of objects.
- b. Modify your description in (a) to include the use of abstract data types such as Name, DOB, and Address.
- c. Use object representation diagrams to show the relationships between:
 - Course and Section.
 - Section and Professor.
 - Professor and Student.
- d. Use object representation diagrams to show the relationships between:
 - Section and Students.
 - Room and Section.

What type of object is necessary to represent those relationships?

- e. Using an OO generalization, define a superclass Person for Student and Professor. Describe this new superclass and its relationship to its subclasses.
6. Convert the following relational database tables to the equivalent OO conceptual representation. Explain each of your conversions with the help of a diagram.
(Note: The R&C Stores database includes the three tables shown in Figure PG.6.)

FIGURE PG.6 THE R&C STORES DATABASE

Table name: EMPLOYEE

EMP_CODE	EMP_TITLE	EMP_LNAME	EMP_FNAME	EMP_INITIAL	EMP_DOB	EMP_SERVICE	STORE_CODE	EMP_AREACODE	EMP_PHONE	EMP_ADDRESS	EMP_CITY	EMP_STATE	EMP_ZIPCODE
101 Mr.	Williamson	John	Viv		21-Jun-1962	Yes	1 545	870-4567	2219 Orchard Road	Flagstaff	AZ	32119	
102 Ms.	Ratula	Nancy			12-Mar-1967	Yes	2 545	873-5467	345 Lake Circle	Flagstaff	AZ	32117	
103 Ms.	Greenboro	Lotta	R		02-Nov-1959	No	4 615	366-8967	Rt. 23, Box 123	Eagleville	TN	30123	
104 Mrs.	Rumpersfro	Jennie	S		01-Jul-1969	No	5 901	224-8332	3425 NW 55th Terrace	Gainesville	FL	38155	
105 Mr.	Smith	Robert	L		23-Dec-1957	No	3 615	123-7009	1234 Airport Road	Smyrna	TN	30118	
106 Mr.	Rensselaer	Cary	A		25-Jan-1964	Yes	1 545	870-0705	1108 Orchard Road	Flagstaff	AZ	32119	
107 Mr.	Ogallo	Roberto	S		30-Aug-1960	No	3 615	876-1004	2345 Meadow View	Murfreesboro	TN	32130	
108 Ms.	Johnson	Elizabeth	I		11-Oct-1966	No	1 545	224-7531	1016 Orchard Road	Flagstaff	AZ	32119	
109 Mr.	Eindismer	Jack	W		19-May-1953	Yes	2 545	224-9245	9829 East Main Str.	Flagstaff	AZ	32120	
110 Ms.	Jones	Rose	R		05-Apr-1964	No	4 703	123-9358	6543 Snowview Circle	Aspen	CO	41234	
111 Mr.	Broderick	Tom			21-Nov-1970	No	3 615	123-2214	4256 Greenbriar Road	Murfreesboro	TN	32130	
112 Mr.	wWashington	Alan	Y		08-Oct-1972	No	2 545	875-4447	2896 Tall Pine Road	Flagstaff	AZ	32119	
113 Mr.	Smith	Robert	N		25-Sep-1962	No	3 615	224-8999	4345 Oak Terrace	Murfreesboro	TN	32128	
114 Mr.	Smith	Sherry	H		24-Jun-1964	No	4 703	224-8999	2693 Edelweiss Lane	Aspen	CO	41234	
115 Mr.	Olenko	Howard	U		24-Jun-1962	No	5 901	123-8878	2314 NW 23rd Place	Gainesville	FL	38152	
116 Mr.	Archialo	Berry	V		04-Oct-1958	No	5 901	876-3428	6541 Clear Lake Drive	Lake City	FL	38167	
117 Ms.	Grimaldo	Jeanine	K		12-Dec-1968	Yes	4 703	123-7890	4356 Snowflake Road	Aspen	CO	41234	
118 Mr.	Rosenberg	Andrew	D		23-Feb-1969	Yes	4 703	123-5360	5112 Avalanche View	Aspen	CO	41235	
119 Mr.	Rosten	Peter	F		03-Nov-1966	No	4 703	224-7211	3256 Tall Timber Lane	Aspen	CO	41234	
120 Mr.	Zack	Robert	S		05-Apr-1968	Yes	1 545	873-2218	3567 Deep Water Drive	Flagstaff	AZ	32117	
121 Ms.	Mcbee	Jennifer	A		10-Jan-1972	Yes	1 545	875-7768	3256 Treebranch Lane	Flagstaff	AZ	32120	
122 Mr.	Ryan	Herman	G		06-Feb-1967	Yes	3 615	567-8903	4436 Hadley Ct.	Smyrna	TN	37123	

Database name: RC_Stores

Table name: STORE

STORE_CODE	STORE_NAME	STORE_YTD_SALES	REGION_CODE	EMP_CODE	STORE_ADDRESS	STORE_CITY	STORE_STATE	STORE_ZIP
1	Access Junction	1403456.00	2	108	1234 Cactus Circle	Flagstaff	AZ	32117
2	Database Corner	1821987.00	2	112	2345 Longview Pike	Flagstaff	AZ	32121
3	Tuple Charge	1366783.00	1	107	9876 Brandywood Road	Murfreesboro	TN	30130
4	Attribute Alley	1344569.00	2	103	7654 Mountainview Drive	Aspen	CO	40123
5	Primary Key Point	3330099.00	1	115	4567 Palmetto Road	Gainesville	FL	38762

Table name: REGION

REGION_CODE	REGION_LOCATION
1	East
2	West

7. Convert the following relational database tables to the equivalent OO conceptual representation. Explain each of your conversions with the help of a diagram. (Note: The Avion Sales database includes the tables shown in Figure PG.7.)

FIGURE PG.7 THE AVION SALES DATABASE

Table name: PRODUCT

PROD_CODE	PROD_TYPE	PROD_SUBTYPE	PROD_MODE	PROD_MANUFACT	PROD_DESCRIPTION	PROD_COST	PROD_PRICE	PROD_QOH	PROD_MIN_QOH	PROD_LAST_ORDER
ADF-841	ADF	Standard	Panel	Narco	Digital display ADF, keep-alive memory/auto dim, combined k	\$1,699.00	\$2,595.00	11	5	11-Mar-2014
AIRMAP	GPS	Moving map	hand-held	Lowrance	High-density display with accu cartridge, auto zoom, backlit	\$619.00	\$895.00	34	15	07-Sep-2013
APOLLO20001	GPS	Standard	Panel	II Morrow	LED display, airspace alerts with user-programmable penetr	\$1,529.00	\$2,245.00	17	10	12-Dec-2013
APOLLO360	GPS	Moving map	Panel	II Morrow	Round GPS with moving map, standard 3-1/8" instrument ho	\$1,229.00	\$1,995.00	32	20	19-Jan-2014
APOLLO-920	GPS	Moving map	Hand-held	II Morrow	Auto zoom, 20 reversible 30-leg flight plans, nearest waypc	\$816.00	\$1,225.00	26	12	16-Feb-2014
AT-150	Transponder	Standard	Panel	Narco	Mode C TSO'd, 250-watt transmitter, compatible with all lead	\$649.00	\$950.00	16	10	24-Jun-2013
CP-136M	Audio panel	Standard	Panel	Narco	Pushbutton, LED, tuning function, internal marker beacon rec	\$605.00	\$980.00	15	5	26-May-2013
EC-10X	GPS	Moving map	Lap	Magellan	6x4.5-in backlit LCD, GPS/electronic chart	\$1,545.00	\$1,999.00	12	5	08-Jun-2013
FLTPRO	GPS	Standard	Hand-held	Trimble	Palm-sized GPS, 4-line interface, LCD, updatable through RS	\$499.00	\$725.00	12	5	20-May-2013
GPS-150	GPS	Standard	Panel	Garmin	Internal rechargeable battery, up to 4 hours use in the event	\$1,349.00	\$1,995.00	29	15	20-Nov-2013
GPS-155	GPS	Standard	Panel	Garmin	Front-loading data card, interfaces with fuel mgt, EFRS, HSI	\$3,888.00	\$5,995.00	14	8	14-Jan-2014
GPS-55AVD	GPS	Standard	Hand-held	Garmin	Alkaline battery pack, yoke mount adapter, power/data cable	\$439.00	\$625.00	31	12	11-Dec-2013
GPS-95XL	GPS	Moving map	Hand-held	Garmin	CDI and Jeppesen database, cigarette lighter adapter, remov	\$699.00	\$1,075.00	27	12	10-Feb-2014
KLN-89	GPS	Moving map	Panel	Bendix/King	VFR Moving map, 4-line gas discharge display, up to 500 u	\$2,289.00	\$3,195.00	18	12	11-Dec-2013
KLN-89B	GPS	Moving map	Panel	Bendix/King	IFR-certifiable, 4-line gas display, certified to TSO C 129 A-1	\$3,899.00	\$5,895.00	14	8	11-Dec-2013
KLN-90B	GPS	Moving map	Panel	Bendix/King	Updatable via PC-compatible computer or with exchangeabl	\$6,311.00	\$8,400.00	11	5	22-Jun-2013
KMA-24	Audio Panel	Standard	Panel	Bendix/King	Push button selection and control for three transceivers and	\$399.00	\$599.00	17	12	12-Feb-2014
KMA-24H	Audio Panel	Standard	Panel	Bendix/King	Addit intercom to KMA-24. Hot mike, voice-activated, push b	\$459.00	\$699.00	12	10	12-Nov-2013
KN-62A	DME	Standard	Panel	Bendix/King	Solid state, 200-channel, distance, groundspeed, time-to-st	\$1,344.00	\$1,899.00	12	5	09-Jun-2013
KX-155	Nav/Com	Standard	Panel	Bendix/King	Electronic digital display, 760 channel, 10 watt, 200 channel	\$1,119.00	\$1,599.00	26	12	12-Nov-2013
KX-165	Nav/Com	Standard	Panel	Bendix/King	Electronic tuning, digital flip-flop, 760-channel, built-in 40-ch	\$1,299.00	\$1,695.00	16	10	12-Nov-2013
KY-195A/197A	Com	Standard	Panel	Bendix/King	Simultaneous display of 2 preselected comm. freq., button-p	\$659.00	\$999.00	12	5	09-Jun-2013
SKYNAV5000	GPS	Standard	Panel	Magellan	High-contrast wide-angle display, front-loading data card, 2C	\$1,249.00	\$1,799.00	16	10	21-Dec-2013
TMA-350D	Audio panel	Standard	Panel	Terra	4-place voice-activated intercom built-in, 3-position toggle s	\$499.00	\$850.00	15	10	17-Jan-2014
TNL1000(DC)	GPS	Standard	Panel	Trimble	Backlit 2x20-character display, 250 waypoint total, interface	\$1,250.00	\$1,695.00	35	10	10-Jan-2014
TNL-2000	GPS	Standard	Panel	Garmin	TSO'd for IFR. Uses Jeppesen NavData card.	\$3,999.00	\$6,650.00	12	5	11-Mar-2014
TNL-2000A	GPS	Standard	Panel	Trimble	Backlit LCD, vertical vav, interface to moving maps, autopilot	\$1,999.00	\$2,695.00	27	10	11-Jan-2014
TRT-250D	Transponder	Standard	Panel	Terra	Solid state, 750 mA @ 14v., gas discharge display, direct to	\$699.00	\$1,070.00	19	10	27-Jan-2014

Database name: Avion_Sales

CUS_NUM	CUS_LNAME	CUS_FNAME	CUS_INITIAL	CUS_AREACODE	CUS_PHONE	CUS_YTD_BUY	CUS_CREDIT	CUS_BALANCE
+ 10010	Ramas	Alfred	A	615	844-2573	2011.56	5000.00	0.00
* 10011	Dunne	Leona	K	713	894-1238	4613.29	5000.00	1708.51
* 10012	Smith	Kathy	vW	615	894-2285	3217.94	5000.00	0.00
* 10013	Olowksi	Paul	F	615	894-2180	10020.53	5000.00	9562.67
* 10014	Orlando	Myron		615	222-1672	14234.85	7500.00	4358.55
* 10015	O'Brian	Amy	B	713	442-3381	3005.77	5000.00	0.00
* 10016	Brown	James	G	615	297-1228	5432.76	5000.00	1217.84
* 10017	vWilliams	George		615	290-2556	1629.95	1000.00	1629.95
* 10018	Farris	Anne	G	713	382-7185	3856.31	2500.00	1232.42
* 10019	Smith	Olette	K	615	297-3809	8938.43	7500.00	854.76

Table name: CUSTOMER

Table name: SELLER

EMP_NUM	SEL_YTD_SALES	SEL_PCT	SEL_YTD_COMMISSION
102	27037.65	8.00	2163.01
105	21945.51	8.00	1755.64
106	30543.82	10.00	3054.38
108	25139.94	5.00	1257.00
109	29515.68	8.00	2361.25
110	20012.45	5.00	1000.62

Table name: EMPLOYEE

Table name: INV_LINE

INV_NUM	INVLINE_NUM	PROD_CODE	INVLINE_UNITS	INVLINE_PRICE	INVLINE_TOTAL
10001	1	GPS-55AVD	1	625.00	625.00
10001	2	TMA-350D	2	850.00	1700.00
10002	1	KX-155	1	1599.00	1599.00
10003	1	TNL1000(DC)	1	1695.00	1695.00
10003	2	KMA-24	1	599.00	599.00
10003	3	CP-136M	4	980.00	3920.00
10003	4	TRT-250D	1	1070.00	1070.00

Table name: INVOICE

INV_NUM	CUS_NUM	EMP_NUM	INV_DATE	INV_SUB	INV_TAX8%	INV_TOTAL	INV_PYMT	INV_BALANCE
10001	10015	103	14-Jan-14	2325.00	186.00	2511.00	2511.00	0.00
10002	10018	105	14-Jan-14	1599.00	127.92	1726.92	1726.92	0.00
10003	10010	105	15-Jan-14	7284.00	582.72	7866.72	5000.00	2866.72

8. Using the ERD shown in Appendix C, The University Lab: Conceptual Design Verification, Logical Design, and Implementation, Figure C.22 (the Check_Out component), create the equivalent OO representation.
9. Using the contracting company's ERD in Chapter 6, Normalization of Database Tables, Figure 6.16, create the equivalent OO representation.

Appendix H

Unified Modeling Language (UML)

Preview

The Unified Modeling Language (UML) is an object-oriented modeling language sponsored by the Object Management Group (OMG) and published as a standard in 1997. UML is the result of an effort headed by the OMG to develop a common set of object-oriented diagrams and notations (symbols and constructs) for the analysis, design, and modeling of systems. Because the origin of UML is closely related to the object-oriented concepts you explored in Appendix G, Object-Oriented Databases, object terminology is used throughout this section.

Keep in mind that UML is not a methodology or procedure for developing databases. Rather, UML is a language that describes a set of diagrams and symbols that can be used to model a system graphically. UML diagrams encompass static data components (classes and their associations) and components such as business processes, data flows, and hardware. Table H.1 shows nine different types of diagrams that the UML standard offers.

Data Files and Available Formats

[MS Access](#) | [Oracle](#) | [MS SQL](#) | [My SQL](#)

[MS Access](#) | [Oracle](#) | [MS SQL](#) | [My SQL](#)

There are no data files for this appendix.

Data Files Available on cengagebrain.com

TABLE H.1

UML DIAGRAMS

DIAGRAM NAME	USAGE
Activity diagram	Describes the behavior of a system. Very similar to data flow diagrams that model specific business processes. Related to Use Case diagrams.
Class diagram	Describes the static components of object classes. (Remember, a class is a collection of similar objects.) Similar in function to an ER diagram in a relational database modeling.
Collaboration diagram	Describes the interaction between objects in a system—messages sent among objects, parameters passed, actions taken, and so on. An alternative to the Sequence diagram.
Component diagram	Describes the arrangement of software components that form a system and the way those components interact.
Deployment diagram	Describes the arrangement of hardware components within a system. Describes what objects run in each component.
Sequence diagram	Describes the interaction between objects in a system—(what messages are invoked and in what order). Very similar to Collaboration diagrams and related to Use Case diagrams.
State diagram	Describes the object's state during object interactions. Models the changes in an object's state during its interactions with other objects.
Object diagram	Describes the static nature of object instances within a system at a given point in time.
Use Case diagram	Describes business processes within a system. Very similar to data flow diagrams.

Because the main focus here is on database design, not all of the different types of diagrams that UML offers are covered. Instead, the content focuses on the use of Class diagrams to model the static data components (object classes and their relationships) that are part of a database *system*.

H-1 Using Class Diagrams to Model Database Tables

The UML Class diagram is the equivalent of the ER diagram in the relational model. The Class diagram is used to model object classes and their associations. Because an object class is a collection of similar objects, a class is the equivalent of an entity set in the ER model. Therefore, a class is described by its attributes—and by its methods.



Note

MS Visio Professional has been used to develop the examples shown in this appendix. To create Class diagrams in MS Visio Professional, from the menu select **File, New, Software, UML Model Diagram, UML Static Structure**. The sequence is illustrated in Figure H.1.

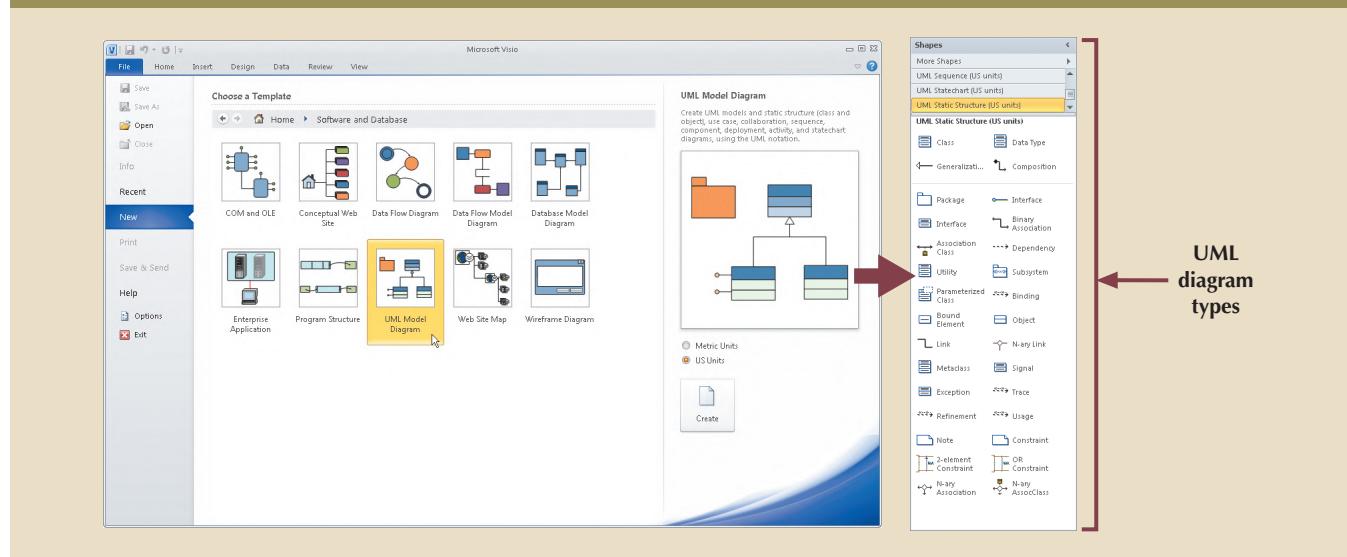
H-1a Classes to Represent Entity Sets

In a UML Class diagram, a class is represented by a box that is subdivided into three parts.

1. The top part is used to name the class.
2. The middle part is used to name and describe the class attributes. (A class attribute is identified by a name and a data type.)

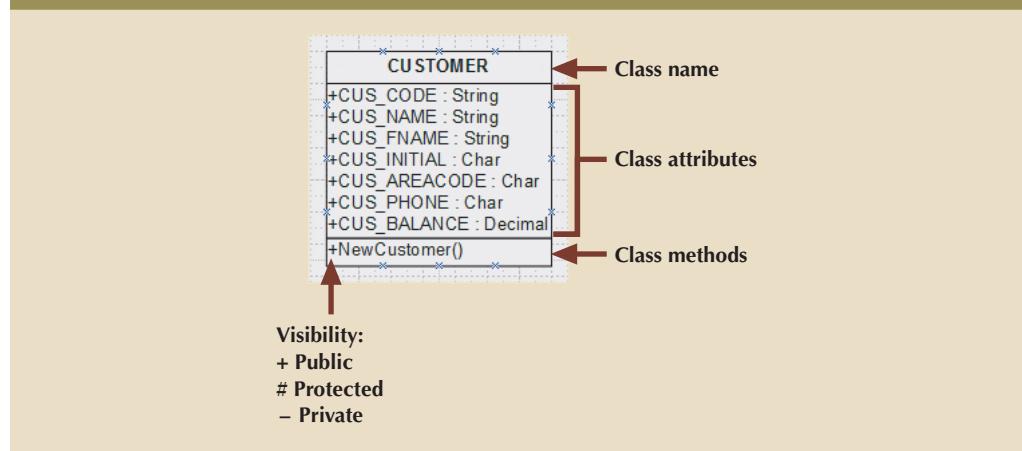
3. The bottom part is used to list the class methods. Both the attributes and the methods are displayed.

FIGURE H.1 CREATING CLASS DIAGRAMS IN VISIO: STARTING THE PROCESS



The three parts are illustrated in Figure H.2.

FIGURE H.2 UML REPRESENTATION OF THE CUSTOMER CLASS



As you can see in Figure H.2, the UML representation of a class is very similar to the ER representation of an entity, but there are some important differences.

- A class box also lists the methods of the class in the bottom part of the box.
- A + symbol is placed before attributes and methods. The + symbol indicates the visibility of the UML element.

H-1b Visibility

The visibility concept is derived from object-oriented programming. *Visibility* describes the availability of an object attribute or method to other objects or methods. Visibility characteristics are summarized in Table H.2.

TABLE H.2

ATTRIBUTE AND METHOD VISIBILITY

	PUBLIC (+)	PROTECTED (#)	PRIVATE (-)
Attribute	The attribute is available for read/write purposes to any method of any class.	The attribute is available for read/write purposes <i>only</i> to the methods of the class and its subclasses.	The attribute is available only to the methods of the class.
Method	The method can be invoked by any method of any class.	The method can be invoked only by the methods of the class and its subclasses.	The method is available only to the methods of the class.

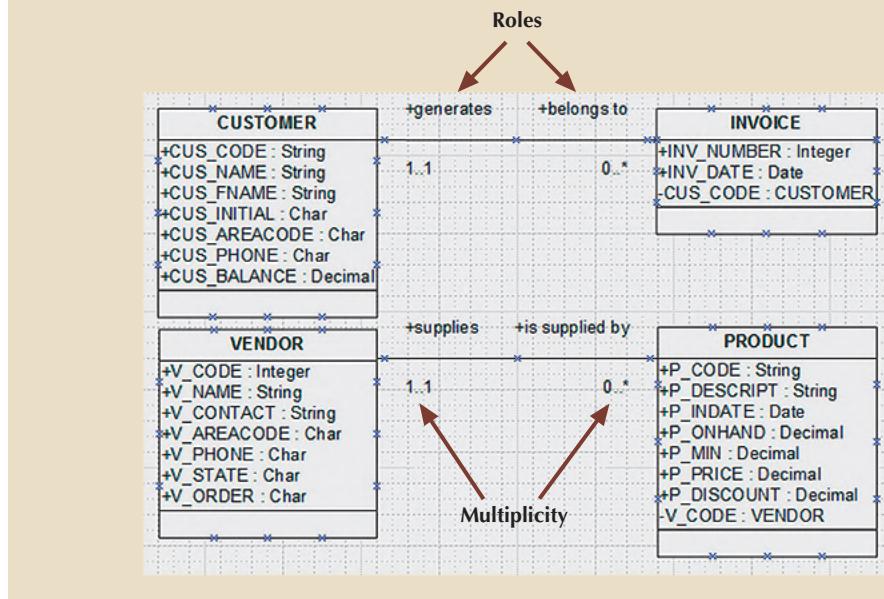
H-2 Associations to Represent Relationships

The UML Class diagram represents relationships as associations among objects. (An object is an instance of a class.) Because associations among classes are critical for database design purposes, you begin by studying how the UML Class diagram represents 1:M associations.

H-2a Representing 1:M Associations

Figure H.3 shows a UML Class diagram with two 1:M relationships: a CUSTOMER generates many INVOICEs, and a VENDOR provides many PRODUCTS.

FIGURE H.3 REPRESENTING 1:M RELATIONSHIPS WITH CLASS DIAGRAMS





Note

UML Class diagrams do not require the foreign key attribute to be added to the “many” side of the 1:M relationship. The object-oriented model implements class associations through the use of object IDs, which are internally managed by the OODBMS. (See Appendix G.) However, because the focus here is on the use of UML Class diagrams to model relational databases, the foreign key attributes are shown in the class diagrams.

By examining Figure H.3, you can see that associations are represented by lines that connect the classes. Associations have several characteristics.

- **Association name.** Each association has a name. Normally, the name of the association is written over the association line. In the example, the association name is not shown; instead, role names are used.
- **Role name.** The participating classes in the relationship can also have role names. A role name expresses the role played by a given class in the relationship. In Figure H.3, the role names represent the relationship “as seen” by each class; for example:
A CUSTOMER *generates* an INVOICE, and each INVOICE *belongs to* a CUSTOMER.
A VENDOR *supplies* a PRODUCT, and each PRODUCT *is supplied by* a VENDOR.
- **Association direction.** Associations also have a direction, represented by an arrow (\rightarrow) pointing to the direction in which the relationship flows. (Relationship direction is not displayed in Figure H.3.)
- **Multiplicity.** Multiplicity refers to the number of instances of one class that are associated with one instance of a related class. Multiplicity in the UML model provides the same information as the connectivity, cardinality, and relationship participation constructs in the ER model. For example:
 - One (and only one) CUSTOMER generates zero to many INVOICES, and one INVOICE belongs to one and only one CUSTOMER.
 - One (and only one) VENDOR supplies zero to many PRODUCTS, and one PRODUCT is supplied by one and only one VENDOR.

Table H.3 shows the different multiplicity values that can be used.

TABLE H.3

MULTIPLICITY	
MULTIPLICITY	DESCRIPTION
0..1	A minimum of zero and a maximum of one instance of this class are associated with an instance of the other related class (indicates an optional class).
0..*	A minimum of zero and a maximum of many instances of this class are associated with an instance of the other related class (indicates an optional class).
1..1	A minimum of one and a maximum of one instance of this class are associated with an instance of the other related class (indicates a mandatory class).
1..*	A minimum of one and a maximum of many instances of this class are associated with an instance of the other related class (indicates a mandatory class).
1	Exactly one instance of this class is associated with an instance of the other related class (indicates a mandatory class).

*Many instances of this class are associated with an instance of the other related class.

The multiplicity symbols implicitly describe the relationship participation concept used in the ER model. For example, a “1..1” multiplicity on the CUSTOMER side indicates a mandatory participation. A “0..*” multiplicity on the INVOICE side indicates an optional participation.

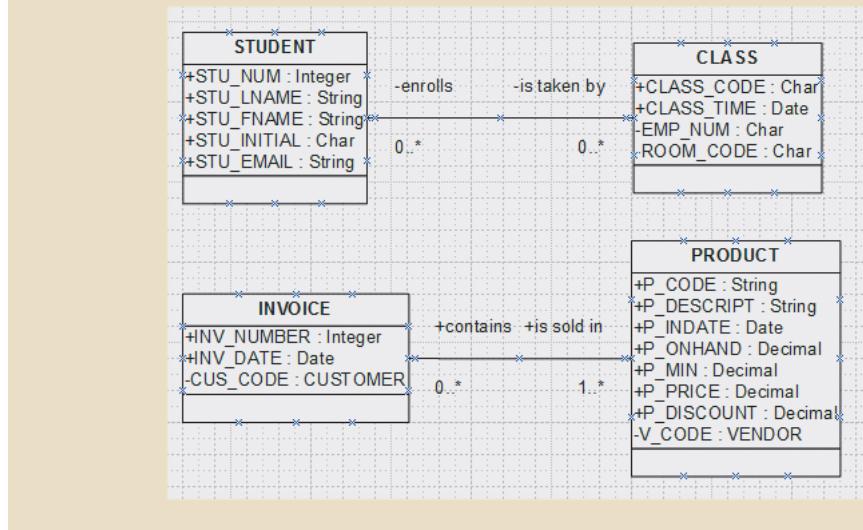
If you examine Figure H.3, you will note that the visibility of the foreign key attributes CUS_CODE and V_CODE have been defined as private (-). Although there is no requirement to specify the foreign key attributes in the UML class diagram, this option has been chosen to highlight the use of the visibility property within the attributes of a class.

H-2b Representing M:N Associations

UML Class diagrams can use the multiplicity element to represent M:N relationships directly. For example, Figure H.4 shows the following two examples of M:N associations:

- A STUDENT enrolls in many CLASSES, and each CLASS is taken by many STUDENTS.
- An INVOICE contains many PRODUCTS, and each PRODUCT is sold in many INVOICES.

FIGURE H.4 M:N ASSOCIATIONS IN A UML CLASS DIAGRAM

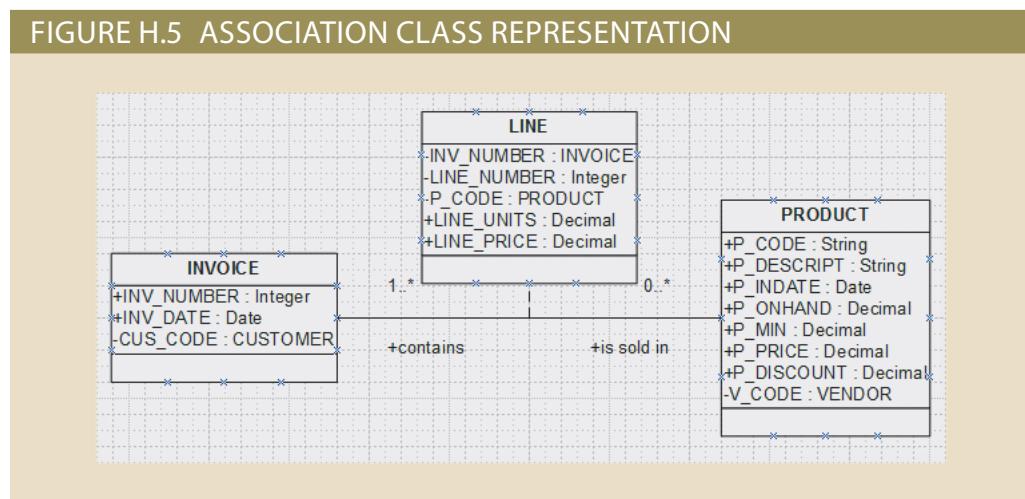


If you examine Figure H.4, you will see that the M:N association between STUDENT and CLASS is optional at both ends. (The optionality is represented by the “0..*” multiplicity.) However, the M:N association between INVOICE and PRODUCT exhibits two different multiplicity values. The “1..*” multiplicity at the PRODUCT end indicates that an INVOICE contains a minimum of one and a maximum of many PRODUCT instances. The “0..*” multiplicity at the INVOICE end indicates that a PRODUCT is sold in a minimum of zero and a maximum of many INVOICE instances.

In the UML diagram, the multiplicity value always refers to the class to which that value is attached. That is, you always try to find out how many instances of a class are associated to one instance of another class. Contrast that to the use of role names, which are always close to the class that plays the role.

H-2c Association Class

An association class is used to represent a M:N association between two classes. The association class exists within the context of the associated objects, and as in the ER model, the association class can have its own attributes. Figure H.5 shows the use of a LINE association class to represent the M:N relationship between INVOICE and PRODUCT.



H-2d Composition and Aggregation

The UML Class diagram uses symbols to indicate the strength of the association between two class instances. In particular, the UML Class diagram uses aggregation and composition to indicate the strength of dependency between two classes participating in an association. Table H.4 summarizes the main characteristics of the aggregation and composition UML constructs.

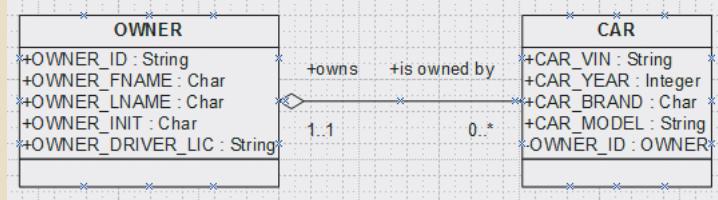
TABLE H.4		
AGGREGATIONS AND COMPOSITIONS		
UML CONSTRUCT	UML SYMBOL	DESCRIPTION
Aggregation	$\diamond \underline{\hspace{1cm}}$	This type of association represents a “has a” type of relationship (that is, an object that is formed as a collection of other objects). An aggregation indicates that the dependent (child) object instance has an optional association with the strong (parent) object instance. When the parent object instance is deleted, the child object instances are not deleted. The aggregation association is represented by an empty diamond in the side of the parent entity.
Composition	$\blacklozenge \underline{\hspace{1cm}}$	This type of association represents a special case of the aggregation association. A composition indicates that a dependent (child) object instance has a mandatory association with a strong (parent) object instance. When the parent object instance is deleted, all child object instances are automatically deleted. The composition association is represented with a filled diamond in the side of the parent object instance. This is the equivalent of a weak entity in the ER model.

Examine the relationships depicted in Figure H.6 to help you understand the use of aggregation and composition.

FIGURE H.6 AGGREGATION AND COMPOSITION EXAMPLES

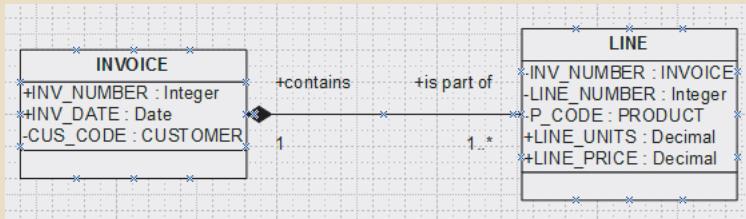
Aggregation

Deleting an OWNER parent instance does not delete all related CAR children instances.



Composition

Deleting an INVOICE parent instance deletes all related LINE children instances.



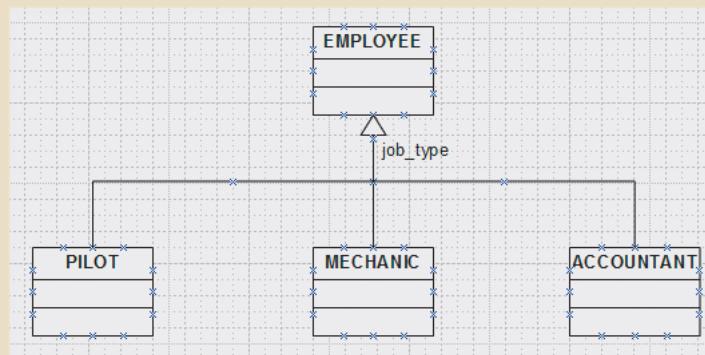
The UML standard guides the use of the aggregation and composition constructs as follows:

- An *aggregation construct* is used when an object is composed of (or is formed by) a collection of other objects, but the objects are independent of each other. That is, the relationship can be classified as a “has a” relationship type. For example, an owner owns many cars, a team has many players, or a band has many musicians.
- A *composition construct* is used when two objects are associated in an aggregation association with a strong identifying relationship. That is, deleting the parent deletes the children instances. For example, an invoice contains invoice lines, an order contains order lines, or an employee has dependents. The use of a composition construct implies the use of the CASCADE DELETE foreign key rule in the relational database model.

H-3 Generalizations to Represent Supertypes and Subtype

The UML diagram also enables you to represent class generalization hierarchies in which a class “is a” subtype of another (supertype) class. You learned about those classification types in Chapter 4, Entity Relationship (ER) Modeling, and in Appendix G. Figure H.7 shows an example of a UML generalization hierarchy.

FIGURE H.7 GENERALIZATION EXAMPLE



The generalization hierarchy is represented by an arrow that points to the parent class. Figure H.7 shows an EMPLOYEE class supertype with three class subtypes: PILOT, MECHANIC, and ACCOUNTANT. In this case, the class hierarchy represents disjoint subtypes; that is, one employee can be related to only one subtype class (a pilot or a mechanic or an accountant).

Generalizations can also have constraints. The *job_type* label next to the generalization line indicates the EMPLOYEE attribute that was used to determine to which class subtype the instance belongs.

H-4 UML and the Relational Model

This brief tutorial has examined the use of the Unified Modeling Language (UML) Class diagram as a database design option. The main focus of UML notation is to facilitate the analysis, design, and implementation of computerized database solutions. To accomplish that task, UML uses a set of diagramming notations derived from object-oriented concepts and, in particular, object-oriented programming. Several points are worth emphasizing:

- UML is *not* a design *methodology*. It is best described as a design *notation*.
- UML notation is *not* geared specifically toward data modeling or relational database design. On the contrary, UML focuses on supporting the process of analyzing and designing information systems.
- One of UML's main characteristics is that it is extensible, thus enabling the designer to create new constructs through the use of so-called stereotypes. As used in the context of UML, a *stereotype* is a new element that represents a distinctive object, characteristic, or functionality in the model. For example, a designer can add new stereotypes to represent primary keys, foreign keys, indexes, triggers, stored procedures, views, and so on.

UML is becoming common in systems analysis and design, but *not* in database design, where relational database modeling tools such as Microsoft's Visio Professional, Computer Associates' ERwin Data Modeler, or Embarcadero's ER/Studio are the norm. Because the relational model is still the dominant data model, the adoption of a relational

data modeling notation within UML would help to increase its penetration in the design and modeling market. If such a merger took place, database designers and system analysts would be able to use the same set of design tools, thus facilitating the creation of information systems.

The extensibility of UML is the characteristic that opens the door for UML to directly support relational database modeling notation. For example, as of this writing, IBM offers its Rational Rose Professional Data Modeler product, which introduces a Data Modeling Profile for UML. This tool allows database designers to create semantically rich relational database models with support for all relational constructs, such as primary keys, foreign keys, indexes, triggers, and stored procedures. Although the Data Modeling Profile is not yet a UML standard, it is backed by IBM, an active member of the OMG consortium. So this merger of concepts and tools could yield the best of both worlds.

Appendix I

Databases in Electronic Commerce

Preview

Electronic commerce (e-commerce) enables organizations—whether they are public or private, for profit or not for profit—to market and sell products and services to a global market of millions of users. Intranets (networks that use Internet technology but operate within an organization) have likewise streamlined internal business operations.

E-commerce companies sell products or services not only to consumers and end users but also to other companies. The Internet has brought about new technologies that facilitate the exchange of business documents and data among business partners. Companies are using the Internet to create new types of systems that integrate their data to increase efficiency and reduce costs.

Online databases are critical components of many e-commerce applications. This appendix introduces the world of e-commerce and illustrates some special concerns that must be addressed in designing e-commerce databases.

Data Files and Available Formats

MS Access **Oracle** **MS SQL** **My SQL**

MS Access **Oracle** **MS SQL** **My SQL**

There are no data files for this appendix.

Data Files Available on cengagebrain.com

I-1 What Is Electronic Commerce?

The term *e-commerce*—short for *electronic commerce*—has had many definitions. The Webopedia online technology dictionary (webopedia.com) defines *e-commerce* as “conducting business online.” However, even a cursory examination of professional publications makes it obvious that the definition of *e-commerce* changes according to whom you ask—and the definitions appear to evolve as fast as the underlying technology. In this book, **electronic commerce (e-commerce)** is defined as the use of electronic networked computer-based technology to:

- Bring new products, services, or ideas to market.
- Support and enhance business operations (including sales of products/services over the web).

Although businesses have used many different types of electronic technology, e-commerce is mainly identified with the use of the Internet as a medium to transact business (buy, sell, and trade products and services) and to add value to an organization. Because the Internet—and, in particular, the web—plays a crucial role in enabling the development and execution of e-commerce, some experts argue that e-commerce should be called Internet commerce (I-commerce), instead.

Although it is easy to view e-commerce as just an online extension of common customer activities, most e-commerce transactions actually take place among businesses. Companies use the Internet and **intranets** (internal networks that use Internet technology but are used within organizations) to streamline their production and distribution processes and to enhance their internal and external operations. In fact, one of the main selling points of e-commerce is that it provides a competitive advantage to the organizations that use it. Because e-commerce operations have become so embedded in the business environment, many people refer to e-commerce as electronic business (e-business).

The external evidence of a company's e-commerce activities is the company's website. A website can be as simple as a few static pages that provide product line and contact information or as complex as a complete online database-driven product catalog with dynamic ordering and credit card payment processing.

E-commerce is now recognized as a prime revenue source. Putting products online makes them immediately available to millions of potential buyers. Companies are competing for a share of the online market by attracting customers and keeping them focused on their websites. But the Internet is a challenging place; many companies are discovering that competing in the online market is more difficult than just creating and using webpages. In short, e-commerce is more than just another marketing channel; e-commerce is the kernel of a new business model dedicated to bringing products, ideas, or services to large markets rapidly in relatively inexpensive ways through the use of Internet technology.

E-commerce is not an end in itself; it is a road that businesses travel to compete and survive in the twenty-first century. On this road, Internet technology has played—and will continue to play—a crucial role. To know where you are going, it is useful to know where you have been. Therefore, the next section provides a brief glimpse of major milestones on the e-commerce road.

I-2 The Road to Electronic Commerce

For many decades, businesses have been using technology to enhance their operations. For example, phones and fax machines are long-standing business uses of technology. Should such technologies be considered part of the e-commerce model? The short answer is no (assuming the phone is not connected to a computer modem). The just-mentioned

examples are not strictly e-commerce because they depend on human intervention for business transactions to take place. Phones and fax machines simply connect the sender with the receiver by transmitting sound or images. To complete the business transaction, a human at the receiving end of the transmission still has to process the information *manually*. The key to e-commerce is using computer networks, especially the Internet, to automate and streamline business transactions. The development of the Internet is closely allied with the development of e-commerce.

The Internet was born as an extension of the DARPA network, started in the early 1960s by the U.S. federal government as a military project to ensure computer communications in case of nuclear attack. Almost immediately after its creation, the Internet was extended to include higher education institutions to facilitate critical research. The Internet's reach did not extend into the business arena until many years later.

The following list summarizes a few of the major technological milestones in business. As you examine the list, note that all of the technological milestones share the common feature of drastically reducing human intervention,

- In the early 1960s, banks created a private telephone network to do electronic funds transfers (EFT). This service allowed two banks to exchange funds electronically in a fast, efficient, and secure manner. The service was restricted to the participating banks, and those banks were required to cover the costs associated with using and maintaining the system.
- In the early 1970s, banks created the automated teller machine (ATM) to provide after-hours services to their customers. In the beginning, ATMs were installed by a few banks nationwide and customers were allowed only a limited number of transactions. As the popularity of the ATMs grew, companies were created to provide ATM service to most banks.
- In the late 1970s and early 1980s, **Electronic Data Interchange (EDI)** emerged. EDI is a communications protocol that enabled companies to exchange business documents over private phone networks. The use of EDI facilitated the coordination of business operations between business partners. The use of EDI became especially well established in the automotive industry. For example, Nissan used EDI to send a request for parts (car seats, tires, or other components) to subcontractors as soon as a car entered the assembly line. The problem with EDI was that its maintenance and implementation costs were high and that the EDI formats varied from company to company. Given those drawbacks, EDI became a secondary (niche) player as Internet communications capabilities grew. However, more recently, EDI has been adapted to Internet technology and now rides the Internet wave, thereby drastically reducing the investment in the communication infrastructure required by previous generations of EDI.
- During the early 1980s and through the 1990s, the personal computer (PC) facilitated the rapid expansion of the Internet and ultimately provided the spark that led to the explosive use of the World Wide Web, usually referred to as “the web.” The web, perhaps the best-known of many Internet services, made the transfer of information among multiple organizations as simple as a mouse click. The web also became the basis for the exploration and exploitation of new Internet-based technologies that led to the enhancement of business processes within and between corporations.
- In the late 1990s and early 2000s, networking technologies blossomed and expanded the reach, the speed, and (in some cases) the security of Internet-based communications and transactions. Many companies began to take advantage of virtual private networks (VPNs), which are private, relatively secure networks that “tunnel” through the Internet, using Internet technology. Wide area networks (WANs) that incorporate

Electronic Data Interchange (EDI)

A communications protocol that enabled companies to exchange business documents over private phone networks.

satellite and microwave links and wireless technologies such as cell phones and wireless networks have also increased the scope and possibilities for businesses large and small. New developments such as **Extensible Markup Language (XML)** are being used to facilitate the dynamic exchange of data among geographically disperse applications. XML has introduced a new dimension in the provision of Internet-based services.

- In the early 2000s, cell phone and wireless technologies began to merge with computer and Internet technologies, creating a whole new arena for e-commerce via cell phones and other wireless devices.



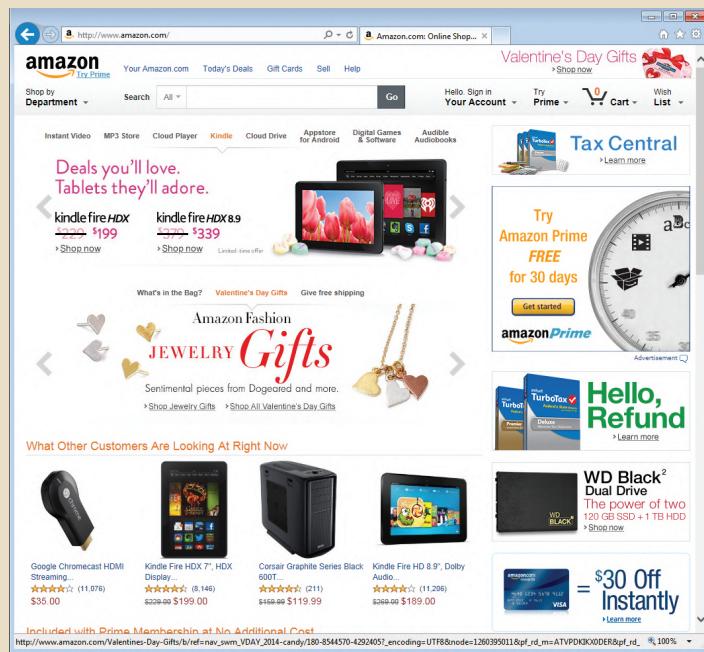
Note

If you want to examine a more detailed history of the Internet, visit the following website:

- www.zakon.org/robert/internet/timeline

As an example of how e-commerce has developed, it is worthwhile to take a look at one well-known e-commerce success story. In 1992, Jeff Bezos launched the website Amazon.com to sell books directly to consumers over the Internet. Five years later, sales exceeded \$131 million. Soon Amazon began expanding into other retailing areas, selling everything from baby goods to music and electronics to clothing and home goods. (See Figure I.1.) Amazon pioneered the “personalized” webpage based on tracking customers’ buying habits and preferences, and it established alliances and cross-links with other major retailers such as Target. In 2014, net sales were reported to be over \$89.9 billion. (For the most current data, visit *Amazon.com* and select *Investor Relations*.) In just a few years, Amazon.com had become one of the most successful retail websites and a model for many other Internet-based businesses.

FIGURE I.1 AMAZON.COM HOME PAGE



Extensible Markup Language (XML)

A metalanguage used to represent and manipulate data elements. Unlike other markup languages, XML permits the manipulation of a document’s data elements. XML facilitates the exchange of structured documents such as orders and invoices over the Internet.

Why did businesses jump on the Internet bandwagon with such abandon? Previous technologies took much longer to find a place at the business table. The next section examines why the web's acceptance was so broad and deep.

I-3 The Impact of E-Commerce

In the 1990s, economists coined the term *the new economy* to refer to the business marketplace based on computer technologies and delivered through the Internet. And although the “tech bubble burst” of 2000 scrubbed the luster off of some e-commerce businesses, many of those businesses quickly adapted to the new market realities and continued to expand. Today a majority of businesses, even traditional (or so-called brick-and-mortar) businesses, have a presence on the web. (Can you think of a major business that does not have a substantial web presence?)

Aside from the aspects of marketing that are enhanced by e-commerce, organizations have many other reasons for implementing e-commerce. The reasons—including rapid response to competitive pressures and customer service requirements, facilitation of transaction management, and inventory management—are given added urgency in today’s world of global markets, mergers, and acquisitions.

The Internet economy works within a global market of interconnected consumers and sellers. Businesses no longer compete only with businesses down the street, across town, or in the same country. Even small organizations have discovered that there is, at least potentially, a global market for their products, services, and ideas. This global market has attracted millions of businesses and organizations to the Internet and e-commerce. The Internet has reached critical mass with hundreds of millions of consumers worldwide, and it has become the new frontier for organizations in the quest for profits and enhanced public services.

For IS departments, the new frontier is the use of Internet technologies to provide services to customers, partners, employees, and the general public. The Internet is driving the development of a new generation of information systems. For many IS applications, the use of Internet technologies facilitates sharing heterogeneous information in an environment that provides multiple benefits at a fraction of the cost.

I-3a Advantages of E-Commerce

E-commerce has benefits for both buyers and sellers, including:

- *Easy comparison shopping.* Consumers can quickly compare prices for just about anything, using sites such as www.mysimon.com or www.pricewatch.com. Such sites have had a major impact on industries such as insurance (www.insure.com), automobiles (www.autobytel.com), and air travel (www.LowFares.com).
- *Reduced costs and increased competition.* Online comparison shopping means intense competition by the suppliers of goods and services, which means lower costs for consumers. For businesses, cost reductions are reflected in a lower cost per transaction. Even though initial Internet infrastructure and implementation costs are high, the cost per transaction tends to be lower because of the marginal cost of each additional transaction and the growing volume of customers using the Internet.
- *Convenience.* Online shoppers can purchase products from the convenience of their homes. By not having to drive to a store, shoppers save substantial time and transportation costs.

- *Operations 24/7/365.* Online stores, unlike their brick-and-mortar counterparts, remain open all year, including weekends and holidays. (Most transactions are automated to the extent that they do not require full-time staffing.) The benefit to businesses is clear, too—they can gain new customers day or night all year-round.
- *Global access.* Through the Internet, businesses have access to millions of users worldwide. Companies can develop national and international exposure. That exposure creates brand awareness and, with a bit of luck, customer loyalty.
- *Lower entry barriers.* An organization looking to establish an online presence expends less in physical location start-up costs; in the hiring and training of a large sales and managerial staff; and in expensive rental, utilities, and advertising fees. Because there is no need to build and furnish attractive offices and/or store interiors, the total time required to launch an online store is reduced. E-commerce has revolutionized the concepts of business time and place.
- *Increased market (customer) knowledge.* The Internet environment makes it relatively easy to compile and track customer information. Businesses can create extensive customer profiles that include purchasing preferences, demographic and geographic information, online behavior, and personal preferences. That information can be used to design websites that attract more users, target specific markets, and display features customized to individuals' interests. Businesses benefit by increasing customer loyalty and generating repeat sales.

I-3b Disadvantages of E-Commerce

Although the list of e-commerce advantages is long, the e-commerce environment is far from perfect. In fact, some disadvantages cause consumers and businesses to suffer considerable anxiety.

- *Hidden costs.* Online purchases are often accompanied by high shipping and restocking fees, a lack of warranty coverage, and unacceptable delivery times. In fact, excessive shipping fees is one of the most frequently cited reasons why shoppers choose not to buy online.
- *Vulnerability to technical failure.* When an e-commerce website cannot service customers because of a network failure, a software virus, or internal hardware failure, the organization loses sales, credibility, and customers.
- *Cost of staying in business.* Although getting into business is relatively easy in an e-commerce environment, staying in business may be more difficult. Increased competition means businesses operate with very thin profit margins. To be profitable, e-businesses must maintain high sales volumes, which means developing and maintaining large and loyal customer bases. Also, to survive and remain competitive, businesses must invest heavily in often costly technology.
- *Lack of security.* One of the main roadblocks to the wide acceptance of e-commerce by businesses and consumers alike is the perceived lack of adequate security for online transactions. For example, many consumers are wary of providing credit card information over the Internet. Several credit card companies are currently working on a set of standards to make online credit card transactions more secure.

- *Invasion of privacy.* The incredible capacity for online data collection is a mixed blessing to customers. Information about their purchases, purchasing habits, demographics, credit history, and so on is stored in databases connected to web servers, potentially exposing the information to cybercriminals. Another concern is that customer information is often sold to marketing companies who engage in email campaigns to attract new customers. The customer's email box is soon filled with unwanted and unsolicited email, or spam.
- *Low service levels.* Another common complaint about doing business online is the low level of customer service that online companies tend to provide. Although technology has automated business transactions to a large extent, a need for the human touch still remains. Therefore, customer service has become a major differentiating factor.
- *Legal issues.* Legal problems encountered in the e-commerce environment include software and copyright infringements, credit card fraud and stolen identities, and online fraud (failure to deliver products and/or services to the customers who paid for them).

I-4 E-Commerce Styles

E-commerce transactions can be grouped according to whom the sellers and the buyers are. Using that distinction, the principal e-commerce styles can be classified as:

- **Business to business (B2B):** Electronic commerce between businesses.
- **Business to consumer (B2C):** Electronic commerce between a business and consumers.
- **Intrabusiness:** Internal electronic commerce activities, most of which involve interactions between employers and their employees.

Although some may argue that **government to business (G2B)** and **government to consumer (G2C)** should be included as additional e-commerce styles, this book considers them to be special cases of B2B and B2C. Consumer-to-consumer (C2C) transactions, particularly C2C transactions facilitated by an intermediary business (C2B2C), are also growing, facilitated by websites, such as eBay and Amazon, that provide avenues for consumers to buy and sell from each other. Figure I.2 identifies the primary e-commerce players and their interactions.

I-4a Business to Business (B2B)

Although B2B transactions include intangibles such as transmitting contracts and moving accounts, most transactions involve sellers and buyers of products and/or services. Generally, the seller is any company that sells a product and/or service, using electronic exchanges such as the Internet or EDI. Examples of B2B transactions are as follows:

- A Nissan Corporation manufacturing plant issues an electronic order for a number of tires of a given model and size. One of the approved tire providers receives and fulfills the order. When Nissan receives the tires, it issues a payment electronically through a bank funds transfer.

business to business (B2B)

Electronic commerce between businesses.

business to consumer (B2C)

Electronic commerce between a business and consumers.

intrabusiness

A style of e-commerce that involves interactions internal to a company.

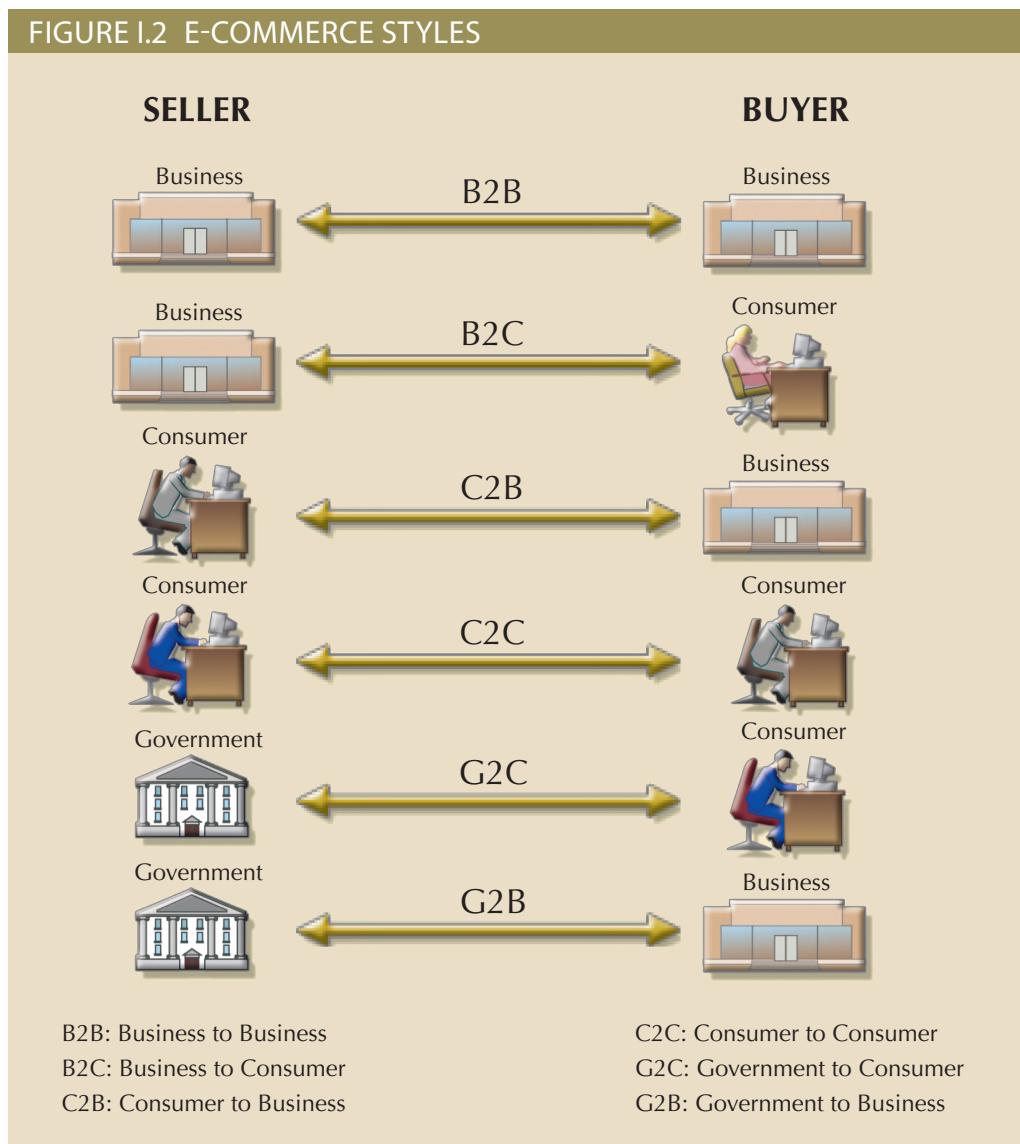
government to business (G2B)

Special case of the Business to Business and Business to Commerce e-commerce styles. See also *government to consumer (G2C)*.

government to consumer (G2C)

Special case of the Business to Business and Business to Commerce e-commerce styles. See also *government to business (G2B)*.

FIGURE I.2 E-COMMERCE STYLES



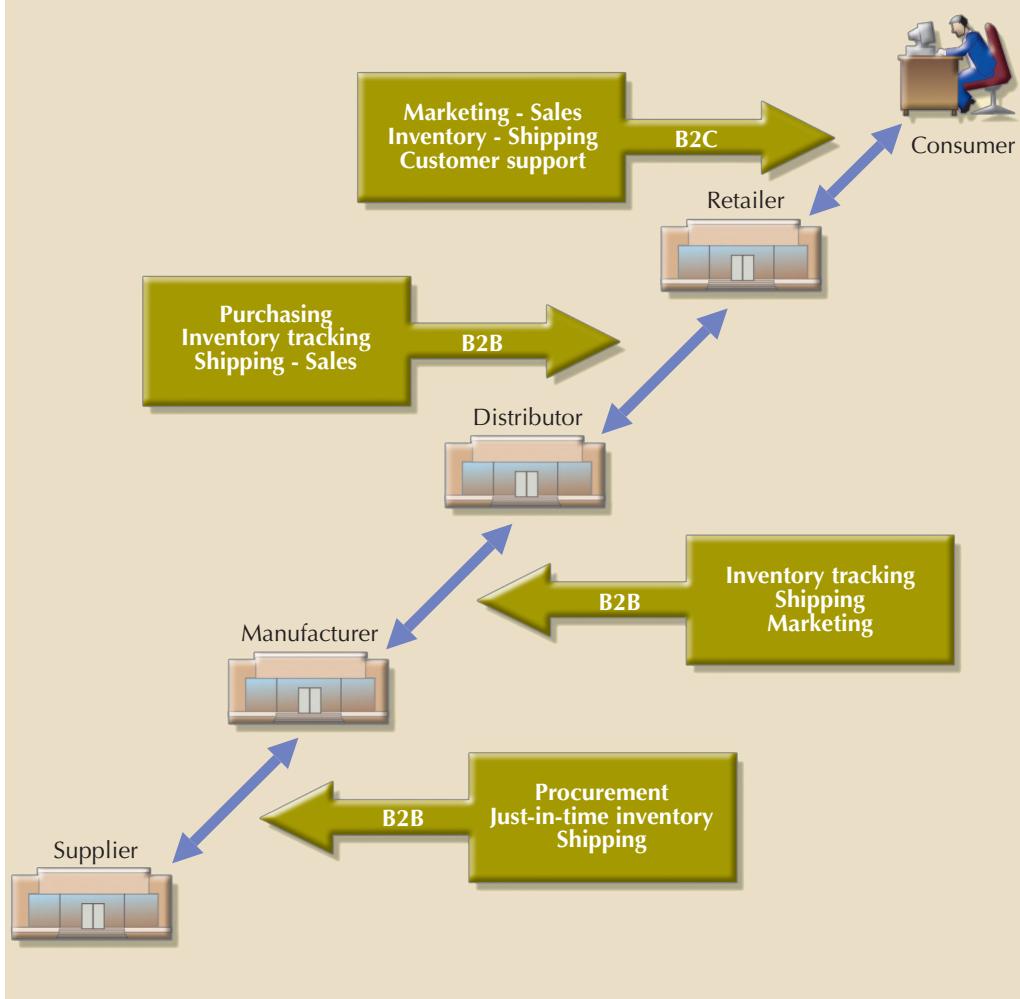
- Using the web, an assistant at the University of Tennessee makes hotel reservations for a group of researchers who will be conducting a seminar at the university.
- The Red Cross uses the web to compare hardware and software prices and subsequently orders hardware and software via the web.

B2B is the biggest and fastest-growing component of the e-commerce market, and the Internet is clearly a dominant player in the B2B economy. In that economy, the main focus is on the use of technology to automate the *value chain* of a business. The **value chain** refers to all activities required to design, plan, manufacture, market, sell, and support a product or service. By examining the value chain components with reference to Internet technologies, businesses can automate and enhance their operations. For example, many companies now use Enterprise Resource Planning (ERP) to manage and enhance all aspects of their value chain, from procuring raw materials to promoting customer satisfaction. Figure I.3 illustrates the use of the value chain to automate B2B transactions.

value chain

All activities required to design, plan, manufacture, market, sell, and support a product or service.

FIGURE I.3 E-COMMERCE AUTOMATION OF SUPPLY CHAINS



B2B transactions are subject to different types of implementation. The two most important versions are:

- **B2B integration.** In this scenario, companies establish partnerships to reduce costs and time and to increase business and competitiveness. For example, a company that manufactures computers will partner with suppliers for hard disks, memory, and other components. That partnership allows the company to automate its purchasing system, integrating that system with its suppliers' ordering systems, thereby linking their respective inventory systems. In that case, when a component in Company C gets below the minimum quantity-on-hand requirement, the system will automatically generate an order to Supplier S. Both systems would be integrated and would exchange business data, generally using XML through the Internet. In addition, the company may integrate its distribution system with that of its distributors. Finally, the distributors may integrate their activities with those of their retailers. And the retailers may, in turn, integrate their operations with those of their customers. Given such integration, sellers align their operations with those of buyers, thereby achieving levels of efficiency that make it difficult to switch to other buyers and/or sellers.

- *B2B marketplace.* In this scenario, the objective is to provide a way in which businesses can easily search, compare, and purchase products and services from other businesses. The web-based system basically works as an online broker to service both buyers and sellers. Within this system, the focus shifts to attracting new members, either sellers or buyers. The “broker” offers sellers a way to market their products or services to other businesses, while buyers are attracted by the fact that they can compare products from different buyers and get access to special deals offered only to members. In that scenario, the broker obtains revenue through membership and transaction fees. Figure I.4 shows the website of Tradekey.com, an example of B2B web marketplaces for the manufacturing sector.

FIGURE I.4 TRADEKEY.COM: B2B MARKETPLACE

The screenshot of the TradeKey.com homepage displays the following key features:

- Header:** Includes the logo "TRADEKEY® Your Key To Global Trade", navigation links (Home, Buy Offers, Products, Companies, Member Area), user stats (5,724,620 Registered Users), and a sign-in bar (Sign In | Join Free | Help | Community | Language).
- Search Bar:** Features a "Search Products:" input field, a "Country/Region" dropdown, and a "Search" button.
- Browse Categories:** A sidebar listing various product categories such as Agriculture, Apparel & Clothing, Automobiles, Beauty & Personal Ca..., Business Services, Chemicals, Computer Hardware & ..., Construction & Real ... (with "Updated!" badge), Consumer Electronics, Electrical & Electro..., Energy Products, Environment, Excess Inventory, Fashion Accessories, Food & Beverage, Furniture, Gifts & Crafts, and Hardware & Mechanica... (with "Updated!" badge).
- Discover Section:** Headline "Discover New Markets and Reach Global Traders Instantly" with sub-sections: "Discover New Markets", "Faster Than Ever", "Maximize Your Profits", and "Trade More with Better Prices".
- Trade Globally:** A grid showing latest buy and sell offers from various countries like China, Turkey, and others.
- Featured Products:** A row of six product thumbnails: Studio Headphones, Conference Microphone, White Long Grain Rice, Santa Beads, LED Underwater Lights, and Printing Plates.
- Welcome to TradeKey.com:** Includes a "Join Now" button and a "It's Free, Fast & Easy" message.
- How TradeKey.com works:** Includes a "Watch Movie" link.
- Premium Memberships:** Compares GoldKey and SilverKey membership levels.
- Success Stories:** A section featuring a testimonial from Ms. Helen Zheng about becoming a Goldkey member.

One important aspect involved in implementing B2B solutions is integrating the *databases* to support information (data) exchanges with other database systems.

I-4b Business to Consumers (B2C)

A business-to-consumer (B2C) operation is one that uses the Internet to sell products and/or services directly to consumers and/or end users. In B2C e-commerce, the Internet—and, in particular, the web—is the marketing, sales, and postsales support channel. B2C is oriented toward attracting customers to the websites and offering products and services in new and innovative ways. Table I.1 lists a sample of B2C e-commerce websites.

TABLE I.1

SAMPLE B2C WEBSITES

INDUSTRY	B2C WEBSITES	INDUSTRY	B2C WEBSITES
Travel	<i>Travelocity.com</i> <i>Expedia.com</i> <i>CheapTickets.com</i>	Computer	<i>Dell.com</i> <i>IBM.com</i> <i>Bestbuy.com</i>
Retailing	<i>Landsend.com</i> <i>Spiegel.com</i> <i>Amazon.com</i>	Health Services	<i>HealthNet.com</i> <i>WebMD.com</i>
Financial	<i>Fidelity.com</i> <i>Etrade.com</i>	Auctions	<i>eBay.com</i>
Banking	<i>WellsFargo.com</i> <i>www.hsbcdirect.com</i>	Reverse Auctions	<i>Priceline.com</i> <i>LendingTree.com</i>
Music	<i>www.itunes.com</i>	Insurance	<i>insure.com</i>
Government	Internal Revenue Service (<i>www.irs.com</i>)	Education	<i>www.elearners.com</i>

Two variations of the B2C marketplace are as follows:

- *Consumer to Business to Consumer (C2B2C)*. A consumer offers items for sale to other consumers through a third-party website. The web's many auction sites, such as *www.ebay.com*, are good examples.
- *Business to Business to Consumer (B2B2C)*. A business offers products or services to consumers through a third-party website. A typical example of B2B2C is a reverse auction such as *www.LendingTree.com* where consumers request bids for loans and financial institutions compete for consumers' business.

I-5 E-Commerce Architecture

Companies embracing e-commerce must deal with both managerial and technological issues. Managerial issues range from establishing partnerships with suppliers, distributors, and vendors to designing and developing well-orchestrated business plans. (Although managerial issues are critical to the success of an e-commerce initiative, they are beyond the scope of this book.) Technological issues include the hardware and software components that provide the backbone for reliable and secure e-commerce transactions. Regardless of the type and extent of an organization's information structure prior to the company's decision to embrace e-commerce, there is an obvious need for the proper design, development, and deployment of a well-planned architecture to support e-commerce business transactions, both internal and external. How those issues are confronted depends on whether the information systems architecture is already established and what the e-commerce style is.

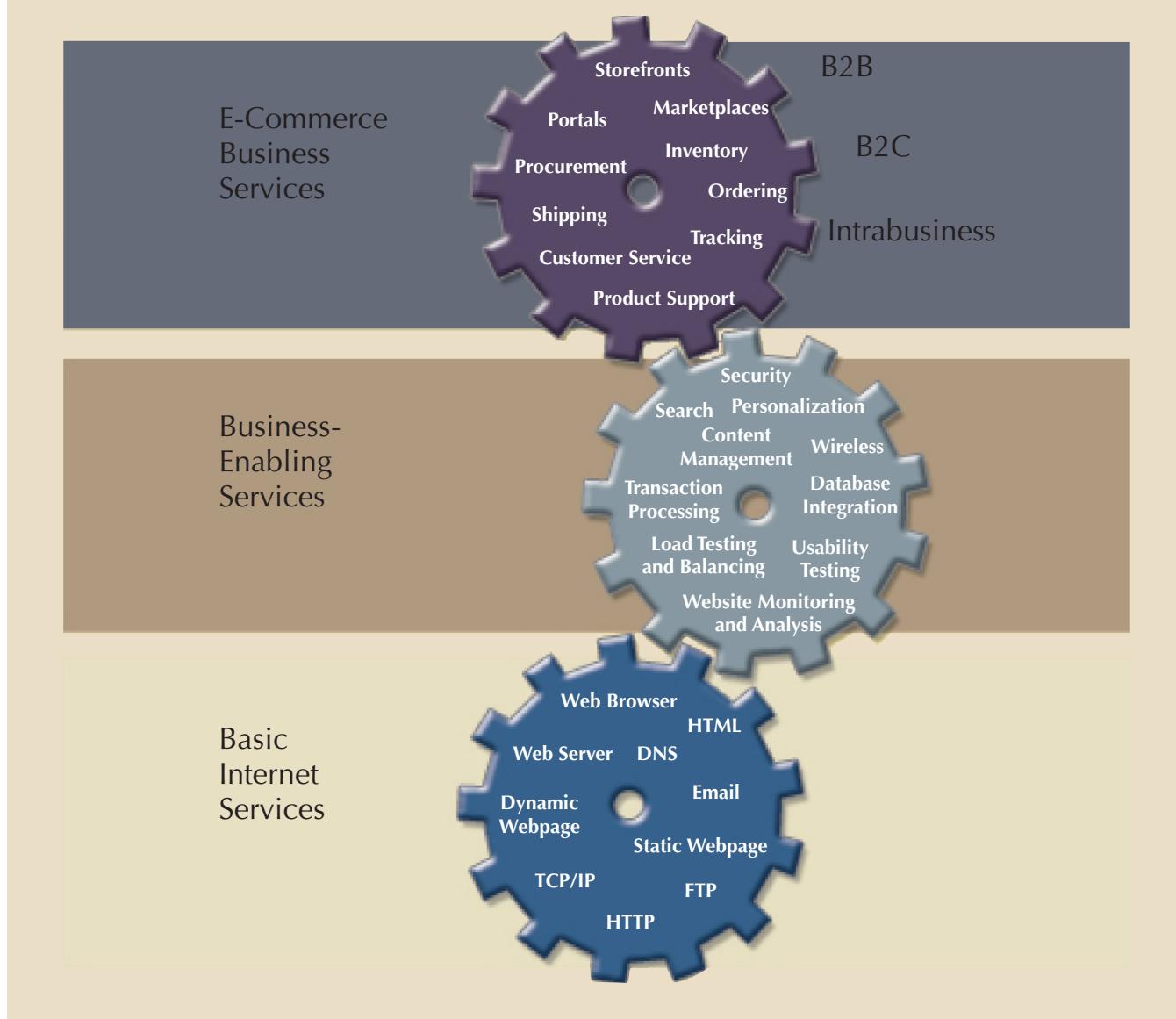
This section describes the basic architectural components that must exist to support e-commerce transactions.

To ensure better understanding of e-commerce architecture, it can be divided into a series of layers. Each layer provides services to the layer below it. Those layers are:

- Basic Internet services.
- Business-enabling services.
- E-commerce business services.

Figure I.5 offers a bird's-eye view of e-commerce architecture.

FIGURE I.5 E-COMMERCE ARCHITECTURE



I-5a Basic Internet Services

The Internet provides the basic services that facilitate the transmission of data and information between computers. The terms *Internet* and *World Wide Web* are often used interchangeably, but they are not synonyms. The World Wide Web functions as one of the many services of the Internet. Table I.2 describes the basic building blocks and services provided by both the Internet and the World Wide Web.

TABLE I.2

INTERNET BUILDING BLOCKS AND BASIC SERVICES

BASIC SERVICE	DESCRIPTION
Internet	A worldwide network of networks. The Internet acts as a “supernetwork” that connects thousands of smaller networks around the world. You can think of the Internet as the “highway” on which data travel, as in the phrase the information superhighway. To connect thousands of heterogeneous networks, the Internet uses a standard network protocol known as TCP/IP and devices known as routers.
TCP/IP	Transmission Control Protocol/Internet Protocol. The basic network protocol that determines the rules used to create and route “packets” of data between computers in the same network or in different networks. Each computer connected to the Internet has a unique TCP/IP address. The TCP/IP address is divided into two parts used to identify the network and the computer (or host).
Router	Special hardware/software equipment that connects multiple and diverse networks. The router is in charge of delivering packets of data from a local network to a remote network. Routers are the traffic cops of the Internet, monitoring all traffic and moving data from one network to another.
World Wide Web (WWW or the web)	A worldwide network collection of specially formatted and interconnected documents known as webpages. The web is just one of many services provided by the Internet.
Webpage	A document containing text and special commands (or tags) written in Hypertext Markup Language (HTML). A webpage can contain text, graphics, video, audio, and other elements.
Hypertext Markup Language (HTML)	The standard document-formatting language for webpages. HTML allows documents to be presented in a web browser in a standard manner.
Hyperlink	Webpages are linked to each other—that is, each webpage calls other webpages creating the effect of a “web.” Because a link can connect to different types of documents, such as text, graphics, animated graphics, video, and audio, it is known as a hyperlink. A hyperlink is generally expressed as an URL in an HTML-formatted webpage.
Uniform Resource Locator (URL) or web address	A URL identifies the address of a resource on the Internet. The URL is an abbreviation (ideally easily remembered) that uniquely identifies an Internet resource. Examples of URLs include www.dell.com , www.ford.com , www.amazon.com , www.faa.gov , and www.mtsu.edu .
Hypertext Transfer Protocol (HTTP)	The standard protocol used by the web browser and web server to communicate—that is, to send requests and replies between servers and browsers. HTTP uses TCP/IP to transmit the data between computers on the Internet.
Domain Name System (Service)	The DNS service translates the “English-like” domain names (such as whitehouse.org and ebay.com) to the appropriate TCP/IP addresses. The DNS service lies at the heart of the Internet because most hyperlinks use URLs to refer to other webpages.
Web browser	The end-user application used to browse or navigate (move from page to page) through the Internet. The browser is a graphical application that runs on the client computer, and its main function is to display webpages. A client uses the web browser (for example, Google Chrome, Microsoft Internet Explorer, or Apple Safari) to request webpages from a web server.
Web server	A specialized application whose only function is to “listen” for client requests, process them, and send the requested webpage back to the client browser. The web server and the web client communicate using a special protocol known as Hypertext Transfer Protocol, or HTTP.
Website	The term used to refer to the web server and the collection of webpages stored on the local hard disk of the server computer or an accessible shared directory.
Static webpage	A webpage whose contents remain the same (when viewed in a browser) unless the page is manually edited. An example of a static webpage is a standard price list posted by a manufacturer for inspection by the manufacturer’s customers.

TABLE I.2

INTERNET BUILDING BLOCKS AND BASIC SERVICES (CONTINUED)

BASIC SERVICE	DESCRIPTION
Dynamic webpage	A webpage whose contents are automatically created and tailored to an end user's needs each time the end user requests the page. For example, an end user can access a webpage that displays the latest stock prices for the companies (s)he selects.
File Transfer Protocol (FTP)	The protocol used to provide file transfer capabilities among computers on the Internet. An FTP client requests a file to an FTP server. The FTP server listens for client's requests, processes them, and sends the requested files back to the client.
Electronic mail (email)	Messages transmitted electronically among computers on the Internet. A mail server stores email messages in end users' mailboxes. Mail clients retrieve email from the mail server. When a client sends an email, it is temporarily stored on the mail server, which then delivers the email to the correct destination.
News and discussion group services	Specialized services that allow the creation of "virtual communities" in which users exchange messages regarding specific topics; for example, aviation, sports, or computers. This service allows end users to post information on shared bulletin boards for public access.

Figure I.6 shows the relationships among the components defined in Table I.2.

As you examine Figure I.6, note that the client enters a web address, or URL, in the web browser. In turn, the web browser issues an HTTP page request to the web server. The request is handled through TCP/IP for transmission over the network. TCP/IP transforms the request into packets and sends them to the router, which, in turn, routes them to the Internet. Once on the Internet, the TCP/IP packets travel across several routers until they reach the destination server. The server receives the webpage request, fetches the page, and sends it to the client's web browser in the same way. The client receives the HTML-formatted webpage and displays it on the screen.

The client and the server can be located in the same building or across the globe, which makes the web a great medium for delivering information across geographic boundaries. The web browser integrates all of the services provided by the website. The end user does not care or know about all of the different applications running on the server side.

Webpages are either static or dynamic. Static webpages display information that does not change much over time or is not time-critical. The content of dynamic webpages change over time and cannot be anticipated, for example, an online ordering system. Static webpages are adequate to display information such as product catalogs or contact information; dynamic webpages are better suited to e-commerce applications such as online ordering with product customization options. For example, at Dell's e-commerce site (www.dell.com), you can dynamically configure your computer. Dynamic webpages are at the heart of most B2B and B2C web transactions.

I-5b Business-Enabling Services

The Internet services described in the previous section are sufficient to operate a basic website. However, they do not provide the support required to conduct even rudimentary business transactions. Business-enabling services are implemented by hardware and software components that work together to provide the additional functionality not provided by basic Internet services. Table I.3 describes the services that are used to enhance websites by providing the ability to perform searches, authenticate and secure business data, manage website contents, and more. The list in Table I.3 is not comprehensive; technological advances continue to enable new services, which are used to bring about even more services.

FIGURE I.6 BASIC INTERNET SERVICES

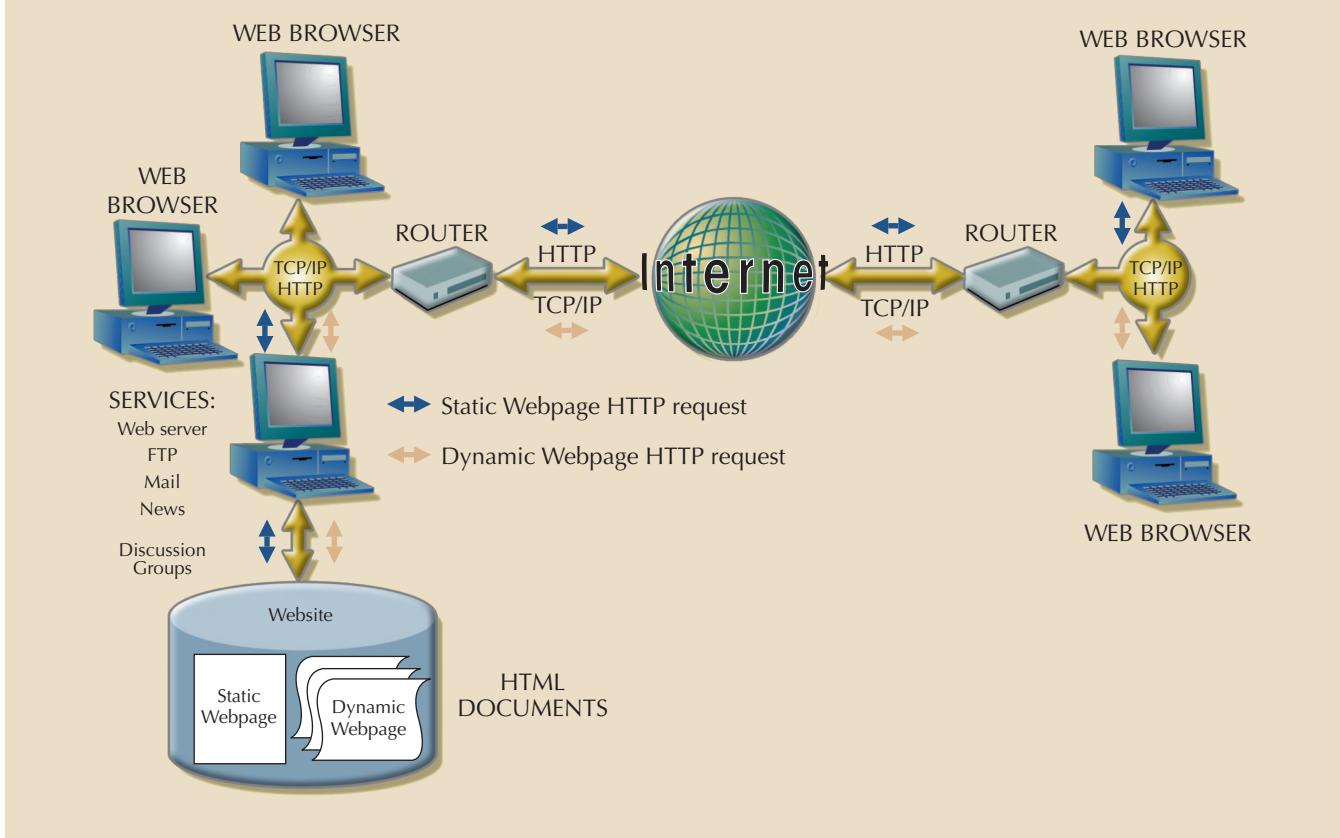


TABLE I.3

BUSINESS-ENABLING SERVICES

SERVICE	DESCRIPTION
Search services	Search services provide websites with the ability to perform searches on their contents. These services can be used in intranets to search for payroll information, benefits, vacation time, contract information, and so on. A B2C website can use this feature to search for product return information or customer support data. Search services are a must-have for all e-commerce websites.
Security	Services that ensure the security and privacy of data by providing encryption, digital certificates, SSL, S-HTTP, firewalls, and proxy servers. Those services are covered in greater detail in Section I.6.
Site monitoring and data analysis	Ensures that the website is performing at an optimal level. Monitors the main indicators of system and network performance. This service also includes the identification of network bottlenecks, often noticed by the end user when webpages load very slowly. Analysis features study network traffic to determine which pages are attracting visitors and which are not. This feedback provides answers to questions about the effectiveness of the pages.
Load testing, balancing, and web caching	Load testing is performed before an e-commerce site goes online. The main objective is to test the website to ensure that it can support the load imposed by thousands of users accessing it. When the expected transaction load is too much for a single server to handle, multiple servers are required. Load balancing ensures that the processing load is distributed evenly among multiple servers. Web caching technologies increase the performance of web servers by creating a "caching" layer between the web server and web client sessions and servicing selected requests directly from the cache without taxing the web server. These services provide performance-enhancing techniques to operate the system at optimum speed.

TABLE I.3

BUSINESS-ENABLING SERVICES (CONTINUED)	
SERVICE	DESCRIPTION
Usability testing	In an e-commerce environment, not having a website is bad enough, but having a badly designed website may be even worse. Usability testing ensures that website features and services are presented in a user-friendly manner. Is the search function hard to find? Are the colors chosen for the website difficult to read on laptop computers? Are the options presented logically?
Personalization	Personalization features allow for the customization of webpages for individual users. The idea behind personalization is making the site user-friendly to attract more users and to keep users coming back. To see personalization in action, go to www.yahoo.com and click on "My Yahoo." You can then specify your interests and your zip code to get a personalized webpage tailored for your local weather, news, and so on.
Web development	E-commerce applications require business logic to be integrated into the website. Web development tools provide the means by which to add business logic to webpages. HTML is not a programming language; it is a document-formatting specification created to present documents properly in a web browser. Business logic can be added to websites by using one of many web-based programming environments, such as Java, JavaScript, JScript, or VBScript. Such programming environments allow the creation of dynamic webpages that form the basis of e-commerce websites.
Database integration	Business transaction data are normally stored in databases. The integration of enterprise databases with the web is a requirement for e-commerce success. Many DBMSs, such as Oracle and Microsoft SQL Server, already come with web development environments that integrate the database with the Internet. Third-party vendors provide solutions that allow corporate databases (including legacy data) to be integrated into a company's website.
Transaction processing	As you might imagine, transaction processing services are very important in e-commerce. Chapter 10, Transaction Management and Concurrency Control, examined the importance of transaction management. In an e-commerce environment, the problem is vastly magnified because transactions often originate from customers around the world.
Content management	Content management automates the creation and management of a website's contents and provides a flexible and consistent way for many different individuals and departments to create webpages. Content management is critical for companies in the business of providing information.
Messaging	The Internet is a "highway" through which applications communicate. An e-commerce application sends messages from a client to a server, and vice versa. Messaging ensures the proper routing and delivery of applications-oriented messages among multiple services.
Wireless device support	Wireless and mobile devices have become big players in e-commerce. Services to support wireless communication have become increasingly important in e-commerce.

I-5c E-Commerce Business Services

E-commerce business services form the top layer of the e-commerce architecture. (See Figure I.5.) That layer uses the services of the two layers below it to map business logic and to automate business processes. Automating business processes enhances the business unit, department, and intrabusiness operations. At the top layer, business-enabling services (layer 2) interact with basic Internet services (layer 1) to provide support for the front-end e-commerce services. Note that it is the business functions that drive the operations of the services below it—not the other way around. The top layer takes

longer to implement because care must be taken to develop a thorough understanding of the business, to establish business partnerships, and to understand customer behavior.

When the web front end is designed and implemented, a decision must be made about what business services to provide and how to provide them. For B2C e-commerce sites, the front end could be as simple as an online product catalog or as sophisticated as a database-based “storefront” with three-dimensional (3D) views of the available products, in addition to a shopping cart application that enables users to order items while they browse the products.

Common services provided by e-commerce websites include automation of the supply chain for B2B purposes. That automation covers online procurement, just-in-time inventory, online ordering, order tracking, product delivery, product support, customer satisfaction management, and so on.

The next two sections introduce two essential features of e-commerce that enable complete online transactions: security and payment processing.

I-6 Security

For e-commerce to be successful, it must ensure the security and privacy of all business transactions and the data associated with those transactions. In an e-commerce context, **security** encompasses all of the activities related to protecting data and other e-commerce components against accidental or intentional (usually illegal) access or use by unauthorized users. **Privacy** deals with the rights of individuals and organizations to determine the “who, what, when, where, and how” of data use.

Even before the emergence of e-commerce, information privacy and security was a major concern in business organizations. In some cases, those concerns merited the hiring of a data security officer to oversee data security and privacy standards and procedures. Integrating databases with the web and using the Internet as a transmission medium for business transactions have made the need for security even greater.

The World Wide Web was originally created with the objective of sharing data easily, rather than securely. For that purpose, the web is a good fit for nonprofit organizations and business/consumer advocacy groups that are more concerned with information distribution than with secure transactions. However, private for-profit organizations require security and privacy to engage in e-commerce business transactions. For example, who would use a credit card to pay online if there were no measures to protect the credit card information from being stolen? How can messages traveling over the Internet be protected against modifications? How can the identity of the sender be verified? How can the web storefront be protected against attacks from cybervandals? Although there is no foolproof way to protect against all possible threats, several technologies do address security problems. Although Internet security is a vast topic, the focus here is on the principal mechanisms used to protect electronic business transactions, web-store front ends, and associated data.

E-commerce data must be secured from the beginning of a transaction to its end. Let's examine the following online purchasing scenario, illustrated in Figure I.7:

1. A customer orders products online, entering order and credit card information on a merchant's webpage.
2. The information travels from the customer's computer over the Internet to the merchant's web server.

security

Activities and measures to ensure the confidentiality, integrity, and availability of an information system and its main asset, data.

privacy

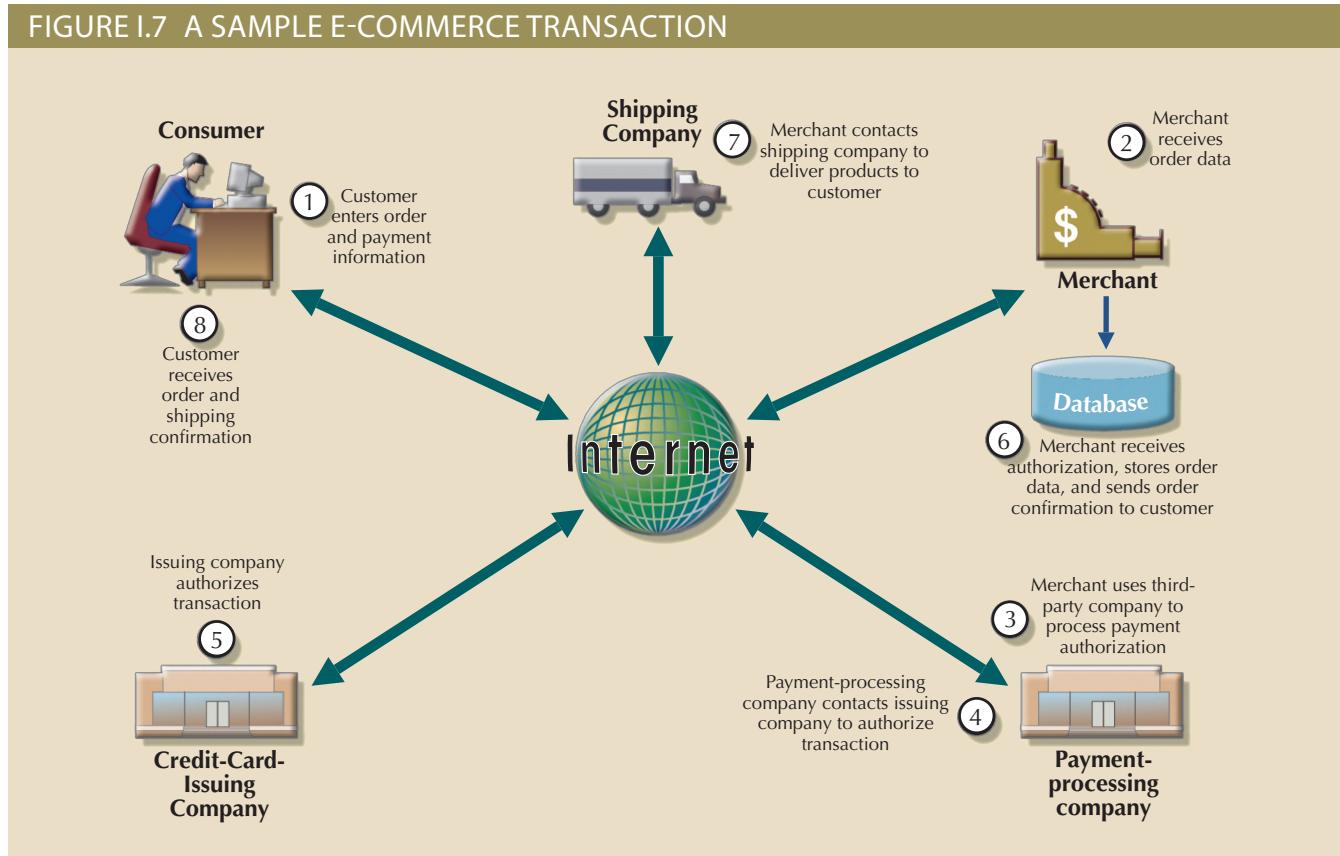
The rights of individuals and organizations to determine access to data about them-selves.

3. The merchant uses a third-party company to process payment authorization.
4. The payment processing company contacts the customer's credit card-issuing company to authorize the transaction.
5. The customer's credit card issuer authorizes the transaction.
6. The merchant receives authorization, stores the order and payment data in a database, and sends order confirmation to the customer.
7. The seller uses a third-party shipping company to deliver the products.
8. The customer receives order and shipping confirmation.

Given the just-described scenario, security (procedures and technology) must be maintained to:

- *Authenticate the identity of the transaction's participants* by ensuring that both the buyer and the seller are who they say they are. In other words, there needs to be a secure way to properly identify transaction participants and the authenticity of their messages.
- *Protect the transaction data from unauthorized modifications* while it travels over the Internet. The Internet is formed by millions of interconnected networks. E-commerce data must pass through several different networks when traveling from the client to the server, thereby increasing the risks of data being stolen, modified, or forged.

FIGURE I.7 A SAMPLE E-COMMERCE TRANSACTION



- *Protect the resources (data and computers).* This includes protecting the end user and the business data stored on the web server and in the databases from unauthorized access. It also includes securing the web server against attacks from hackers wanting to break into the system to modify or steal data or to impair normal operations by limiting resource availability.

I-6a Authentication

Authentication refers to the process of properly and uniquely identifying entities. Such an entity could be a user in a computer system or a database, a computer in a network, or participants in an e-commerce transaction. You probably are familiar with the authentication used in local area network systems in which you are given a unique user ID and password. For example, on most systems that use Microsoft Windows, you must enter your user ID and password to log on. In that way, Windows authenticates who you are. Based on your user ID (identity), Windows assigns you access rights (read, write, and so on) to resources such as printers and files.

In the case of e-commerce, authentication concerns properly identifying a user on the Internet. Suppose that you have placed an online order for a Sony STR-DE525 receiver. After a week, your order still has not arrived. But when you call the merchant, you are informed that the item was shipped several days ago to an address that is not yours. Apparently, somebody changed the shipping address while the order traveled over the Internet prior to being delivered to the merchant's website. From this admittedly simplified scenario, you can see why verifying the identity of participants and hiding the contents of communications from intruders are important.

Internet authentication is a bit more complicated than the Windows scenario. On the Internet, digital certificates issued by a certification authority (CA) are used to authenticate both users and vendors. A **certification authority (CA)** is a private entity or company that certifies that the user or vendor is who (s)he claims to be. Certification authority companies work with credit card verification companies and other financial institutions to verify the identity of the certificate's requesters. When a user registers with a CA such as VeriSign (acquired by Symantec), (s)he will be asked to provide appropriate verification information. (See Figure I.8.) Using the end-user data provided, the CA verifies the identity of the requester and issues a digital certificate to the end user. A **digital certificate** is a unique identifier given to an entity. The holder of the certificate may be an end user, a website, a computer, a webpage, or even a program. Digital certificates are used in combination with encryption to provide security and authentication.

authentication

The process through which a DBMS verifies that only registered users can access the database.

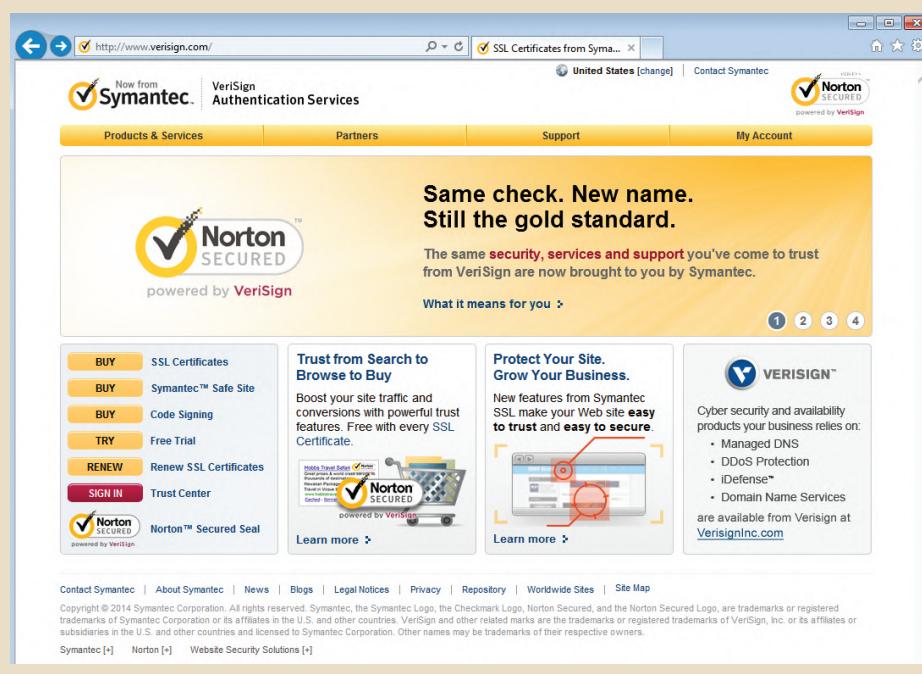
certification authority (CA)

A private entity or company that certifies the user or vendor is who (s)he claims to be.

digital certificate

A unique identifier given to an entity. The certificate holder may be an end user, a website, a computer, a webpage, or even a program. Digital certificates are used in combination with encryption to provide security and authentication.

FIGURE I.8 REGISTERING WITH A CERTIFICATION AUTHORITY



encryption

A process of inputting data in “plain text” to yield an output “encoded” version of the data, making the data unintelligible to unauthorized users.

encryption key

Used by encryption algorithms to encode data. The encryption key is a very large number used to encrypt and decrypt data.

symmetric encryption

Encryption that uses a single numeric key to encode and decode data. Both sender and receiver must know the encryption key. See also *private-key encryption*.

private-key encryption

Encryption that uses a single numeric key to encode and decode data. Both sender and receiver must know the encryption key. See also *symmetric encryption*.

Data Encryption Standard (DES)

The most widely used standard for private-key encryption. DES is used by the U.S. government.

asymmetric encryption

A form of encryption that uses two numeric keys—the public key and the private key. Both keys are able to encrypt and decrypt each other’s messages. See also *public-key encryption*.

public-key encryption

A form of encryption that uses two numeric keys—the public key and the private key. Both keys are able to encrypt and decrypt each other’s messages. See also *asymmetric encryption*.

public key

A key that is available to anyone wanting to communicate securely with the key’s owner.

I-6b Encryption

Digital certification does not totally guard against improper use of data. So what further assurances can be made that the original sender sent the document and that it has not been forged? That is where encryption comes into the picture. **Encryption** is a process of inputting data in “plain text” to yield an “encoded” output of the data, making the data unintelligible to unauthorized users. To secure e-commerce transactions on the web, the client web browser must encrypt the data before sending it over the Internet to the merchant’s web server. The merchant’s web server receives the encrypted transaction data and decrypts it. All sensitive communications between client and server must be encrypted. Encryption works to promote:

- Data privacy to ensure that the data cannot be understood if intercepted.
- Data authenticity to ensure that the data were not forged.

Most encryption algorithms use mathematical formulas and an encryption key to encode the data. The **encryption key** is a very large number used to encrypt and decrypt the data. The length of the key—that is, the number of digits in it—determines how secure the data will be. The longer the key, the more secure the data. Most encryption algorithms use key lengths that range from 40 bits to 128 bits or more. Most modern web browsers such as Firefox and Internet Explorer support either 40-bit or 128-bit encryption.

Encryption algorithms are of two types: symmetric (private-key) or asymmetric (public-key). **Symmetric** or **private-key encryption** uses a *single* numeric key to encode and decode data. Both sender and receiver must know the encryption key. The most widely used standard for private-key encryption is **Data Encryption Standard (DES)** used by the U.S. government. Because each client/server combination requires a different key, this type of encryption is usually used by government entities only and is not used for e-commerce transactions over the Internet.

Asymmetric or **public-key encryption** uses two numeric keys.

- The **public key** is available to anyone wanting to communicate securely with the key’s owner.
- The **private key** is available only to the owner.

Both keys can encrypt and decrypt each other’s messages.

An example of public-key encryption is **Pretty Good Privacy (PGP)** by Pretty Good Privacy, Inc. PGP is a fairly popular and inexpensive method for encrypting email messages on the Internet. The most commonly used public-key encryption technology is RSA encryption by RSA Data Security Inc. RSA has become the de facto standard for Internet encryption. Figure I.9 shows security settings for the most common web browsers.

Digital certificates use public-key encryption techniques to create digital signatures. A **digital signature** is an encrypted attachment added to the electronic message to verify the sender’s identity. The digital certificate received by the user includes a copy of its public key. The owner of the digital certificate makes its public key available to anyone wanting to send encrypted documents to the certificate’s owner.

I-6c Transaction Security

As you know, webpages are plain-text documents created through the use of HTML. In fact, the vast majority of webpages that travel the web are transferred in plain text. Those plain-text transmissions are acceptable for many applications, but they are clearly unacceptable when sensitive e-commerce data such as credit card and bank account information are transmitted. This section examines protocols that are used to transmit HTML documents securely over the Internet.

Secure Sockets Layer (SSL) is a protocol used to implement secure communication channels between client and server computers on the Internet. SSL was originally created by a group of companies led by Netscape Communications, IBM, and Microsoft. The SSL protocol requires that the following conditions be met:

- The client accesses a merchant's web server that supports SSL.
- The server sends its digital certificate, including the server's public key, to the client.
- The client uses the certificate to verify the server with the certification authority (CA).
- The client generates a private key for the session, encrypts the key using the server's public key, and sends it over the Internet to the server.
- The server receives the data, decrypts the data with its private key, and extracts the SSL session private key.

Once the client and the server each have the SSL session private key, they communicate by sending encrypted data back and forth. Transport Layer Security (TLS) is a more recent version of SSL that is even more secure.

The advantage of SSL and TLS is that they can be used with many different Internet services (FTP, Telnet, and HTTP), which means they are widely supported on the Internet.

private key

"A key that is known only to the owner of the key."

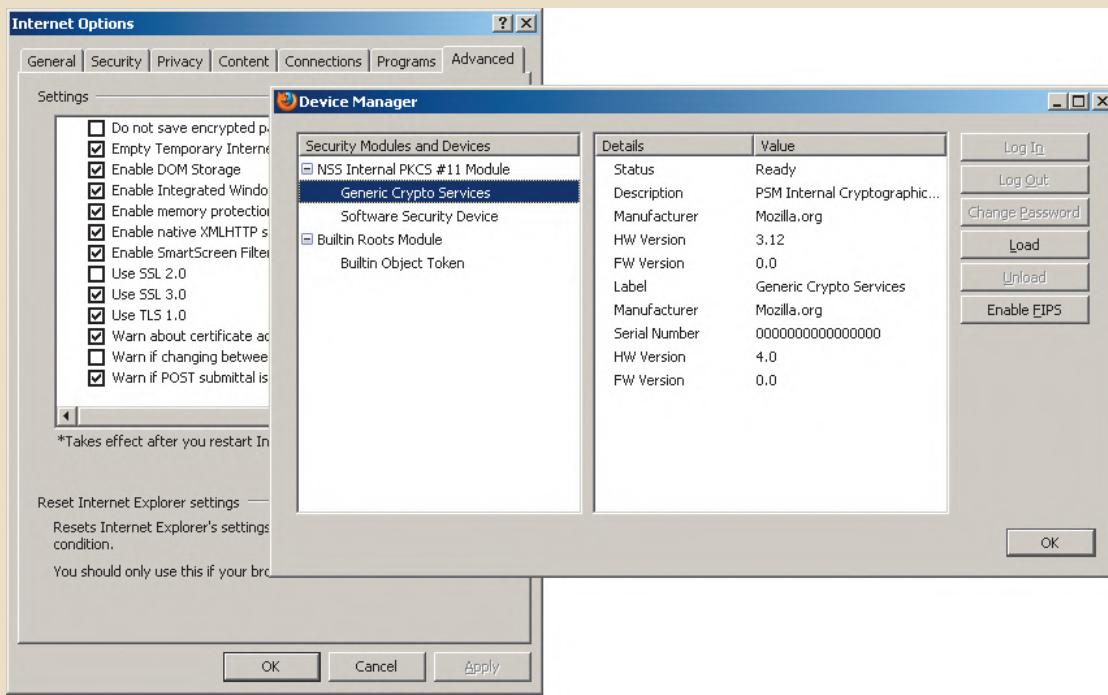
Pretty Good Privacy (PGP)

An example of public-key encryption by Pretty Good Privacy Inc. PGP is a fairly popular and inexpensive method for encrypting e-mail messages on the Internet.

digital signature

An encrypted attachment added to an electronic message to verify the sender's identity.

FIGURE I.9 WEB BROWSER SECURITY SETTINGS



Note

Secure Sockets Layer (SSL) is now being superseded by its successor **Transport Layer Security (TLS)**. TLS uses the same principles as SSL and includes some improvements that are beyond the scope of this discussion.

Secure Sockets Layer (SSL)

A protocol used to implement secure communication channels between client and server computers on the Internet.

Transport Layer Security (TLS)

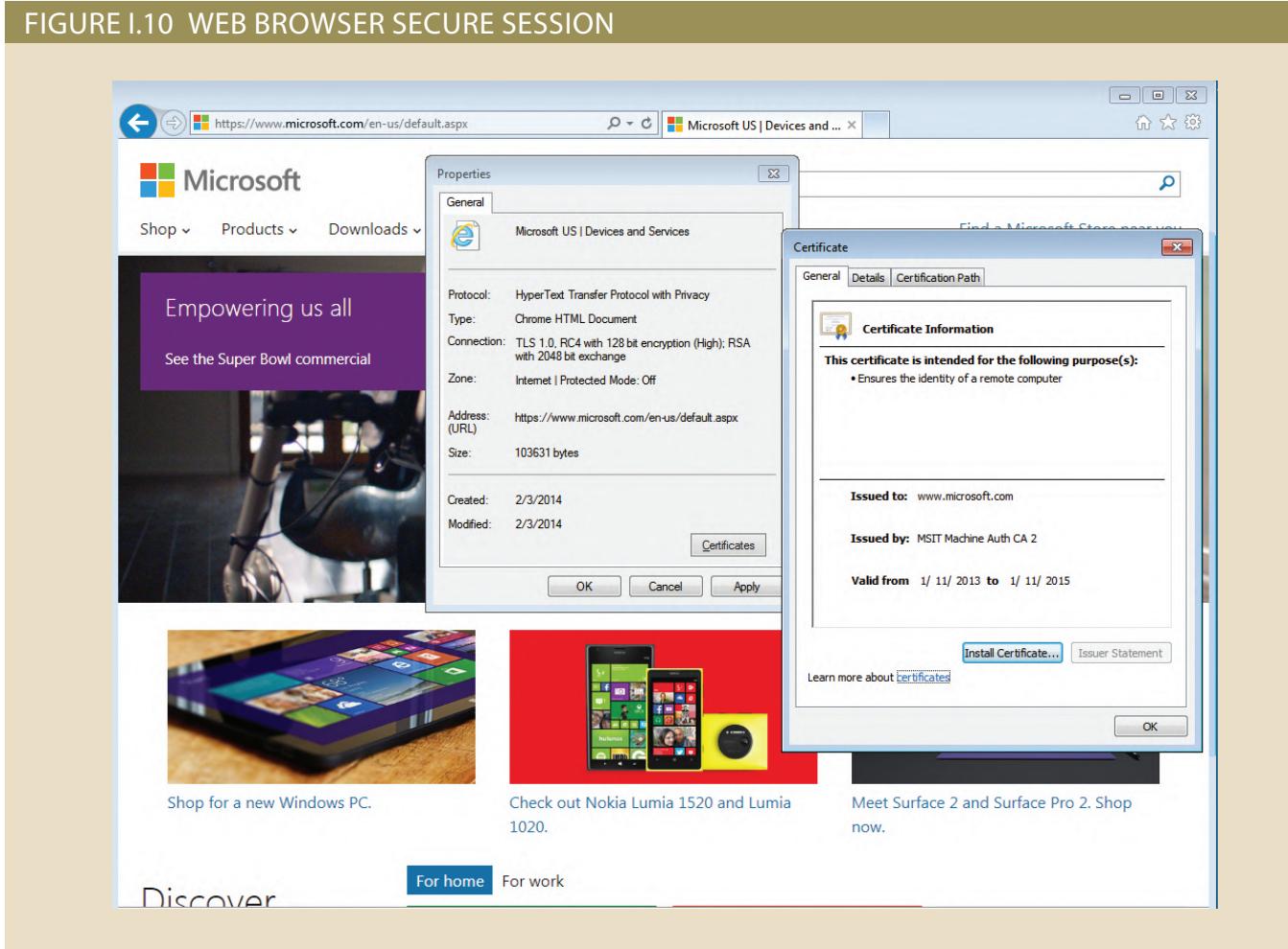
An updated version of Secure Sockets Layer (SSL) that supports more secure encrypted communication between a client and server on the Internet.

To request a secure connection, just add an *https* prefix to the server's web address. For example, to access a secure connection to Microsoft, enter *https://www.microsoft.com* in your browser to generate the screen shown in Figure I.10. (To see the Properties window, right-click the webpage and select "Properties." Note that the Properties window shows a secure connection based on 128-bit encryption.)

Secure Hypertext Transfer Protocol (S-HTTP) is used to transfer web documents securely over the Internet. S-HTTP supports use of private and public keys for authentication and encryption. S-HTTP has not been widely used because it supports only encrypted HTTP data, not other Internet protocols as SSL does.

The next section looks at the technologies that are used to protect the integrity of the resources that are connected to the Internet.

FIGURE I.10 WEB BROWSER SECURE SESSION



I-6d Resource Security

Resource security refers to the protection of the resource(s) connected to the Internet from external and internal threats. Specifically, resource security means protecting the computers connected to the Internet from viruses, intrusion by hackers, and denial-of-service attacks.

The most common threat is probably the **virus**. A virus is a malicious program that affects the normal operation of a computer system. Both client and server computers must be protected against this threat by use of a virus protection program. Virus protection programs must be constantly updated with the latest signature files to identify

new viruses. For more information about viruses, visit the websites of vendors such as Symantec (www.symantec.com) and McAfee (<http://www.mcafee.com>).

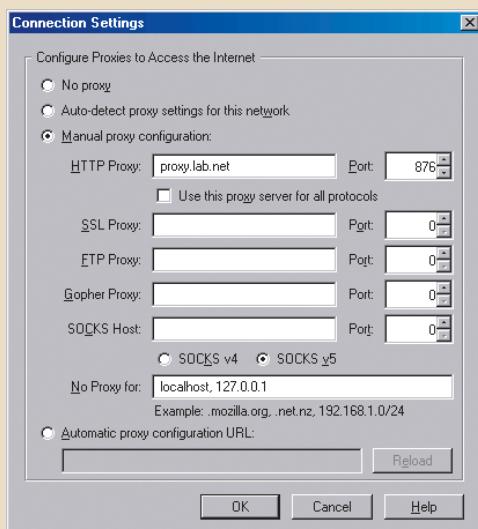
A **hacker** is a person who maliciously and illegally accesses a website with the intention of stealing data, changing webpages, or impairing website operations. Illegally accessing websites with the intention of defacing or changing webpage contents is very common. Because e-commerce web servers connect directly to the Internet, they are prime targets for hackers. One of the most common activities of hackers is to render websites unusable with a denial-of-service attack. A **denial-of-service** attack overloads web servers and routers with millions of requests for service, rendering the services unavailable to legitimate users. A distributed denial-of-service attack originates from many different computers at the same time. Currently, no methodology is totally effective in protecting websites against distributed denial-of-service attacks. However, one of the most effective and commonly used defenses against viruses and hacker attacks is a *firewall*.

A **firewall** is used to protect the network from unauthorized access by the outside world (public Internet). Specifically, a firewall is a hardware and/or software component that is used to limit and control Internet traffic that goes into the company's network infrastructure and data that are allowed to be moved outside the company's network.

Firewalls can be classified as follows:

- **Packet filter firewall.** It works at the TCP/IP packet level, examining every packet that moves into and out of the company's private network.
- **Gateway server firewall.** It works at the application level, examining every application request (HTTP, FTP, Telnet, and so on) that moves into or out of the company's network.
- **Proxy server firewall.** A proxy server is an intermediary between client computers inside a private network and the Internet. The proxy server accepts requests from clients and forwards them to the Internet. When replies come back from the Internet, the proxy server sends the data back to the client that made the request. The client computer application must be configured to work with the proxy server. (Figure I.11 shows an example of the Internet Explorer proxy settings.)
- **State inspection firewall.¹** This new firewall technology compares only parts of incoming packets to parts of the related outgoing packets.

FIGURE I.11 WEB BROWSER PROXY SETTINGS



Secure Hypertext Transfer Protocol (S-HTTP)

Protocol used to securely transfer web documents over the Internet. S-HTTP supports use of private and public keys for authentication and encryption. S-HTTP has not been widely used, because it only supports encrypted HTTP data and does not support other Internet protocols as does SSL.

resource security

The protection of the resource(s) from external and internal threats. Specifically, resource security means protecting the resource from viruses, unauthorized access by hackers, or denial of service attacks.

virus

A malicious program that affects the normal operation of a computer system.

hacker

A person who maliciously and illegally accesses a website with the intention of stealing data, changing webpages, or impairing website operations.

denial-of-service

One of the most common hacker activities. This attack overloads web servers and routers with millions of requests for service, rendering the services unavailable to legitimate users.

¹For additional information about this type of firewall, visit Check Point Software at www.checkpoint.com.

Firewalls, proxy servers, and virus protection programs are just a few of the technologies used to secure and protect e-commerce sites. No single technology can protect an e-commerce web server against viruses, hacker intrusions, and denial-of-service attacks. Implementing resource security includes a mixture of the technologies examined in this section, in addition to others that are not within the scope of this book (for example, intrusion detection systems, vulnerability assessment tools, and secure operating systems).

I-7 Web Payment Processing

A key function of e-commerce websites is their ability to process online payments for products and/or services. In a traditional business transaction such as buying a CD at a music store, you pay for your purchase using cash, a check, a money order, or a credit card. On the web, the most common method of payment is credit card, but there are some less common alternatives. This section briefly introduces two technologies that aid in the processing of electronic payments: digital cash and online credit card processing.

I-7a Digital Cash

One of the risks of paying with credit cards is theft. If your credit card information is stolen, you may find extra charges on your next credit card bill. If somebody steals a \$5 bill from your wallet, you lose only \$5. Thus, the use of cash has certain advantages. **Digital cash** is the digital equivalent of hard currency (coins or bills). Digital cash uses digital certificates to verify the identity of the transaction's participants and requires the existence of a bank or financial institution from which the user will buy digital cash. When the user buys digital cash, (s)he receives digital bills or coins backed by a financial institution and represented by binary IDs. The user will later transfer the digital cash to a merchant to pay for products and/or services. The merchant submits the digital cash to the bank to credit the account. Digital cash remains an evolving technology, with a number of competing companies and few standards.

One of the main advantages of digital cash is its lower cost per transaction when compared to bank checks and credit cards. In fact, given relatively high transaction cost, some online merchants have imposed a minimum purchase for credit card transactions. Nevertheless, digital cash usage pales in comparison to credit card transactions. Reasons for the reluctance to embrace digital cash include the following:

- Difficulty in getting merchants, banks, and customers to change the way they do business.
- Regulatory obstacles.
- Customers' acceptance of credit cards as the most convenient way to pay for online purchases.

Given the current lack of general acceptance, the future of digital cash providers is uncertain.

A more successful approach to web payment processing has been undertaken by companies such as PayPal. PayPal sets up secure transfers from and to users' credit cards or bank accounts. Registered PayPal users can shop at thousands of PayPal-enabled sites. In addition, users can safely send money to or request money from anyone with an email address. PayPal has been popular because it offers business services and is widely used as a payment system for online auctions. eBay acquired PayPal in 2002. By 2013, it had more than 137 million active members in 193 markets. (You can get more up-to-date figures by visiting www.paypal.com and clicking "About.")

firewall

Used to protect a network from unauthorized access from the outside world (public Internet). Specifically, a firewall is a hardware and/or software component that is used to limit and control Internet traffic going into a company's network infrastructure and data that are allowed to be moved outside a company's network.

packet filter firewall

A type of firewall that works at the TCP/IP packet level.

gateway server firewall

A type of firewall that operates at the application level.

proxy server firewall

A firewall that operates as an intermediary between client computers inside a private network and the Internet.

state inspection firewall

A type of firewall that compares parts of incoming packets and related outgoing packets.

digital cash

The digital equivalent of hard currency (coins or bills of a given denomination).

I-7b Online Credit Card Processing

Most online purchases are made with a credit card. Although web merchants accept credit cards as a form of payment, not all merchants use identical methods to process credit card payments. In previous sections, you learned about different security techniques used to protect credit card information. This section briefly explores some of the ways in which merchants can process credit card payments.

The simplest payment processing method is manual processing. In this scenario, a seller collects credit card information from the customer via the website, using a secure connection. (In some cases, the buyer must place the actual order by phone.) The seller then contacts the credit card issuer—by phone or through a credit card verification device—to get authorization information. If problems arise (such as insufficient credit, a wrong card number, or a stolen card), the merchant emails the customer to inform him/her about the problem. Although this multistep manual process continues to survive, thanks to the existence of many small web companies, its inefficiency has caused most sellers to modify or abandon it.

A more efficient variation of the manual method is for the merchant to use third-party credit-card-processing software such as PayEezy (www.payeezy.com) or to employ the services of a third-party processing company such as Converge (www.myvirtualmerchant.com/VirtualMerchant/). In this scenario, the client customer's computer, the merchant's web server, the payment-processing company's server, the customer, and the merchant's bank work together to process the payment. In its simplest form, the procedure requires the following steps:

1. The customer sends credit card information via the web to the seller's web server.
2. The seller's web server verifies the order and sends the credit card information to the credit-card-processing company.
3. The credit-card-processing company verifies the credit card with the customer's bank and obtains authorization information.
4. The merchant's web server receives confirmation of the transaction.
5. The merchant's web server emails the purchase order confirmation to the client.
6. When the merchandise is delivered, the seller and the credit card company settle the payment.

One important feature of all credit card payment processing systems is that the charge is issued only after the seller ships the products and/or services to the client.

Two of the major credit card companies, Visa and MasterCard, have created a system known as *Secure Electronic Transactions*. The **Secure Electronic Transaction (SET)** standard provides for security and privacy of credit card information through the use of digital certificates, digital signatures, and public-key encryption. Security is automatically enforced for all communications between customer, merchant, and financial institutions. One of the main advantages of SET is that it provides a way for the buyer to transfer his/her credit card information to the credit card issuer *without the seller being able to see the credit card information*—by requiring merchants to encrypt all credit card information. SET has been widely accepted by the most prominent Internet companies and vendors, such as Microsoft, IBM, and GTE. SET also defines the minimum network and security infrastructure requirements for companies that want to perform electronic payment verification on the web. That standard includes the use of firewalls, encryption, digital signatures, encrypted database data, antivirus software, operating system patches, end-user authentication, auditing, and other security policies.

Secure Electronic Transaction (SET)

Initiative to provide a standard for secure credit card transactions over the Internet.

Now that you've had an overview of the world of electronic commerce, let's turn to database design for e-commerce applications.

I-8 Database Design for E-Commerce Applications

Through experience, you know that reinventing a heat source is not necessary each time you want hot water for a cup of coffee. Similarly, to design e-commerce databases, you do not need to invent "new" design techniques. To the contrary, the design techniques you have learned in this book (ER modeling, normalization, SDLC, DBLC, transaction management, and so on) provide the basic tools and the knowledge required to build successful e-commerce databases. However, e-commerce databases have a few requirements that you have not encountered yet. That is why this section illustrates a basic database design for an e-commerce application. In Chapter 15, Database Connectivity and Web Technologies, you learned how to set up a web-enabled database and you learned about web database development.

Let's start by defining the scope of the database. The simple e-commerce website must include at least the core features that facilitate the sale of products and/or services. Therefore, the database must support the website's ability to show the available products and/or services and to conduct basic sales transactions. In addition, the e-commerce website should offer features that focus on customer service, product returns, and web customer profiling, which make the customer's web experience a pleasant one. To accomplish that end, an e-commerce database design must include a few additional support tables. However, the focus here is mainly on the database entities that directly support the sale of products in an e-commerce database.

To start the design process, let's establish some basic business rules and their effect(s) on the design.

- The objective of the e-commerce design is to sell products to customers. Therefore, the database's first two tables will be PRODUCT and CUSTOMER.
- Each customer may place one or more orders. Each order is placed by one customer. Therefore, there is a 1:M relationship between CUSTOMER and ORDER.
- Each order contains one or more order lines. Each order line is contained within an order. Therefore, there is a 1:M relationship between ORDER and ORDLINE.
- Each order line references one product. Each product may appear on many order lines. (The company can sell more than one HP ink-jet printer.) Therefore, there is a 1:M relationship between PRODUCT and ORDLINE.
- Customers who browse the product catalog would like to see products grouped by category or type. (For example, customers would find it useful to see product lists broken down as computers, printers, application software, operating systems, and so on.) Therefore, each PRODUCT belongs to one PRODTYPE, and each PRODTYPE has one or many PRODUCTS associated with it.
- Customers who browse the web catalog must be able to select products and store them in an electronic shopping cart. The shopping cart temporarily holds the products until the customer checks out. Therefore, the next entity is SHOPCART. Each SHOPCART belongs to one CUSTOMER and references one or more PRODUCTS.
- When the customer checks out, (s)he enters credit card and shipping information. That information is added to the ORDER. (Note that the business rule identifies required attributes.)

- When the credit card authorization is received, an order is placed for the products found in the shopping cart. The SHOPCART information is used to create an ORDER, which contains one or more ORDLINES. *After the order is placed and the customer leaves the website, the shopping cart data are deleted.*
- Because the merchant offers many shipping options, a SHIPOPTION table is created to store the details of each shipping option, that is, Land, 2-day, Next day, UPS, FedEx, and so on.
- Because the merchant offers many payment options, a PMTOPTION table is created to store the details of each payment option, that is, MasterCard, Visa, American Express, and so on.
- Because each state may have a different tax rate, two tables are created, (STATE and TAXRATE) to keep track of the states (and countries) and the tax rate for each.

Given that brief summary of business rules and their effect(s) on the design, a summary of the entities are shown in Table I.4.

TABLE I.4

MAIN TABLES FOR E-COMMERCE DATABASE

TABLE NAME	TABLE DESCRIPTIONS
CUSTOMER	Contains details for each registered customer. This table contains general customer data, shipping preference, credit card data, and billing data (for customer accounts). Some customers may prefer not to register; if so, they will have to enter the customer details each time they place an order.
PRODUCT	Contains product details such as stock IDs, prices, and quantity on hand.
PRODTYPE	Identifies the main product type classifications.
ORDER	Contains details about general orders, such as date, number, and customer.
ORDLINE	Contains the products ordered for each order.
SUPPORT TABLES	
SHOPCART	Contains the purchase quantity for each product selected by the customer. This is a “working” table—whose contents are deleted when the customer exits the website or closes the browser.
PMTTYPE	Contains the different payment options offered by the merchant.
SHIPTYPE	Contains the different shipping options offered by the merchant.
TAXRATE	Contains the tax rate for each state and/or country.
STATE	Contains the list of each state and/or country for which the tax is charged.
PROMOTION	Includes special promotions such as vouchers and sales discounts.
PRICEWATCH	Includes customers who want to be notified if a product’s price reaches a certain level.
PRODPRIICE	Is an optional table used to manage multiple price levels.



Note

To save space and to recognize your ability to translate the business rules into an ERD, which you learned in Chapter 4, Entity Relationship (ER) Modeling, and Chapter 6, Normalization of Database Tables, you will not develop the ERD for this design.

After defining the tables that are required to support the e-commerce activities, the basic attributes for each table are identified. Note that the following attribute summary is only a sample of the most important—and most commonly used—attributes; it is not meant to be comprehensive. (Your specific environment will determine what attributes are relevant and/or what attributes might be added.)



Note

The discussion will mention some programming and design practices that may be used to build e-commerce websites. For example, because the data stored in the tables exist to support web transactions, it would be advisable to define some fields—such as product names or descriptions—for webpage display purposes. Also, fields such as credit card numbers and passwords should be stored in an encrypted format. The fields themselves also may be encrypted, thus making it unlikely that even a field's contents would be improperly accessed. (Databases such as Oracle and IBM's DB2 support field level encryption.)

I-8a The CUSTOMER Table

The CUSTOMER table contains the details for each registered customer. Keep in mind that some customers may prefer not to register. They may feel uncomfortable providing registration information. Therefore, a decision will have to be made about whether to require registration for all purchases.

- With mandatory registration, the website needs a form for registering new customers. Also, a login form is required for returning customers before they start browsing the product catalog. Given the registration data, the customer's shipping and credit card data can be automatically placed in the order when the customer checks out. (One of the benefits of registration is that customers may be rewarded with discounted prices.)
- With optional registration, there is no need to generate a registration or login form for each sale or visit. The customer must enter all shipping and credit card information with each order.

Obviously, the simplest option is not to require registration. Therefore, the decision has been made here to make registration optional. The main CUSTOMER table attributes are shown in Table I.5.

TABLE I.5

CUSTOMER TABLE		
ATTRIBUTE NAME	DESCRIPTION	PK/FK
CUST_ID	Customer ID—automatically generated	PK
CUST_DATEIN	Date the customer was added to the table	
CUST_LNAME	Last name	
CUST_FNAME	First name	
CUST_ADDR1	Address line 1	
CUST_ADDR2	Address line 2	
CUST_CITY	City	

TABLE I.5

CUSTOMER TABLE (CONTINUED)

ATTRIBUTE NAME	DESCRIPTION	PK/FK
CUST_STATE	State or region if international customer	FK
CUST_ZIP	Zip code	
CUST_CNTRY	Country	
CUST_PHONE	Phone	
CUST_EMAIL	Email address	
CUST_LOGINID	Login ID for registered customers	
CUST_PASSWD	Password for login—encrypted field	
CUST_CCNAME	Name as it appears on credit card	
CUST_CCNUM	Credit card number—encrypted field	
CUST_CCEXDATE	Credit card expiration date in mm/yy format	
CUST_ACRNUM	Accounts Receivable number—to interface with the internal accounts receivable system or a reference PO number for clients set up for net 30 terms	
CUST_BLLADDR1	Billing address line 1	
CUST_BLLADDR2	Billing address line 2	
CUST_BLLCITY	Billing address city	
CUST_BLLSTATE	Billing address state	FK
CUST_BLLZIP	Billing address zip	
CUST_BLLCNTRY	Billing address country	
SHIP_ID	Favorite shipping type	FK
CUST_SHPADDR1	Shipping address line 1	
CUST_SHPADDR2	Shipping address line 2	
CUST_SHPCITY	Shipping address city	
CUST_SHPSTATE	Shipping address state	FK
CUST_SHPZIP	Shipping address zip	
CUST_SHPCNTRY	Shipping address country	
CUST_TAXID	Tax ID for tax-exempt customers	
CUST_MBRTYPE	Membership type—used to identify special promotions and to determine product pricing according to membership level; for example, regular price, member price, or gold member price	

I-8b The PRODUCT Table

The PRODUCT table is the central entity in the database. The PRODUCT table contains the relevant product information about all of the products offered on the website. This table (see Table I.6) is related to the PRODTYPE, ORDLINE, and PROMOTION tables.

TABLE I.6

PRODUCT TABLE

ATTRIBUTE NAME	DESCRIPTION	PK/FK
PROD_ID	Product ID—automatically generated.	PK
PROD_NAME	Product short name—shown in promotions, invoices, and so on; for example, Verbatim CD-R.	
PROD_DESCR	Product description—a long description of the product; used in webpages for product information.	
PROD_OPTIONS	Product options; for example, color, size, and style. (There are many ways to handle sizes or colors for apparel and shoe industries; several of them require separate product entries or the creation of other tables in 1:M relationships.)	
PROD_IMAGE_1	URL of the product's image file; could occur many times (front view, back view, side view, top view).	
PROD_SKU	Stock number used by the vendor or supplier.	
PROD_PARTNUM	Part number from manufacturer; for example, VBTM 34563.	
VEND_ID	Vendor—the vendor ID for the product; for example, Global Suppliers.	FK
PTYPE_ID	Product type (category); for example, Storage.	FK
PROD_UNIT_SIZE	Unit size of the product: box, case, each.	
PROD_UNIT_QTY	Unit quantity in unit size: 12, 6, 1.	
PROD_QOH	Quantity on hand in the warehouse per each product.	
PROD_QORDER	Quantity on order—items that have been ordered but not yet shipped. To determine if an item is in stock, subtract the quantity on order from the quantity on hand.	
PROD_REORD_LEVEL	Reorder level—When the quantity on hand is equal to this amount, the product is reordered.	
PROD_REORD_QTY	How much to reorder from the vendor.	
PROD_REORD_DATE	Estimated date the order will arrive from the vendor.	
PROD_PRICE	Regular price per unit quantity (each); for example, \$1.05 per CD-R.	
PROD_MSRP	Manufacturer's suggested retail price—to show savings.	
PROD_PRICE_D1	Price discount 1—for members or order quantity level; for example, 3%.	
PROD_PRICE_D2	Price discount 2—for gold members or order quantity level; for example, 6%.	
PROD_TAX	Yes or No—Is the product taxable?	
PROD_ALTER_1	Alternative product if not in stock. Could occur many times. This is a foreign key to the same product table. Again, this could be implemented by creating another separate table in a 1:M relationship.	FK
PROD_PROMO	Yes/No. Product participates in promotions? Default Yes.	
PROD_WEIGHT	Weight of product, used for shipping purposes.	
PROD_DIMEN	Product dimensions, used for shipping purposes.	
PROD_NOTES	Notes about the product, shipping, handling instructions, and so on.	
PROD_ACTIVE	Yes/No. If not active, the product is not available to customers. Useful to recall products or to stop sales of a given product.	

Generally, there is one row for each product. The exception is those products that come in various sizes, colors, or styles, such as shoes and shirts. In those cases, there are three options, as follows:

- Enter the product's size, color, and style as additional attributes in the order.
- Create unique product entries for each product size, color, and style combination.
- Create a new product option (PRODOPT) table in a 1:M relationship with the PRODUCT table. This table will have one record for each combination of color, size, and style for a given product.

I-8c The PRODTYPE Table

This table describes the different product categories. The categories could be limited to just one level or could be multiple levels. In this example, two levels are used. This will permit the use of categories such as “printer”; within that category, subcategories such as “laser” or “inkjet” could be referenced. (See Table I.7.)

TABLE I.7		
PRODTYPE TABLE		
ATTRIBUTE NAME	DESCRIPTION	PK/FK
PTYPE_ID	Product Type ID—automatically generated	PK
PTYPE_NAME	Product Type Name—“Inkjet Printer”	
PTYPE_PARENT	Product Type Parent—“Printer”	FK

I-8d The ORDER Table

This table contains all of the customer orders. After the credit card company approves the transaction, the order is added to the ORDER table. If the credit card is rejected (invalid number, expired, or stolen), the order is not added. There will be one ORDER row for each new customer order, regardless of the number of products ordered. If a registered customer places an order, the credit card and shipping information could be automatically entered in the ORDER table. The ORDER table is in a 1:M relationship with ORDLINE table. (See Table I.8.)

TABLE I.8		
ORDER TABLE		
ATTRIBUTE NAME	DESCRIPTION	PK/FK
ORD_ID	Order ID—automatically generated.	PK
ORD_DATE	Date the order was added.	
CUST_ID	Customer ID (optional)—some customers will not register. If this were a registered customer, the CUS_ID would be automatically added by the web system.	FK
PMT_ID	Payment type ID—selected by the customer.	FK
ORD_CCNAME	Name as it appears on credit card—copied from CUSTOMER data; manually entered by an unregistered customer or by electronic wallet software.	
ORD_CCNUM	Credit card number (encrypted field)—copied from CUSTOMER data; manually entered by an unregistered customer or by electronic wallet software.	
ORD_CCEXDATE	Credit card expiration date in mm/yy format—copied from CUSTOMER data; manually entered by an unregistered customer or by electronic wallet software.	
SHIP_ID	Selected shipping type—automatically or manually entered; used when only one company or shipment is used to fulfill the order.	FK
ORD_SHIPADDR1	Shipping address line 1—automatically or manually entered.	
ORD_SHIPADDR2	Shipping address line 2—automatically or manually entered.	
ORD_SHIPCITY	Shipping address city—automatically or manually entered.	
ORD_SHIPSTATE	Shipping address state—automatically or manually entered.	FK

TABLE I.8

ORDER TABLE (CONTINUED)

ATTRIBUTE NAME	DESCRIPTION	PK/FK
ORD_SHIPZIP	Shipping address zip—automatically or manually entered.	
ORD_SHIPCTRY	Shipping address country—automatically or manually entered.	
ORD_SHIPDATE	Date the order shipped if complete shipment. If partial shipment, see ORDLINE for shipment dates for each product line.	
ORD_SHIPCOST	Total shipment cost—estimated shipment cost for order. This is the result of applying a given shipment cost formula according to the shipment method.	
ORD_PROD COST	Total product cost—the sum of all product prices * quantity ordered.	
ORD_TAXCOST	Total cost of sales tax—computed by adding the taxes for each individual product ORDLINE table.	
PROM_ID	Promotion ID applied to order (optional).	
ORD_TOTCOST	Total cost of order: PRODCOST + SHIPCOST + TAXCOST – PRO_AMT (from promotion table).	
ORD_TRXNUM	Transaction confirmation number from credit card company.	
ORD_STATUS	Status of the order: Open, Shipped, or Paid.	

I-8e The ORDLINE Table

This table contains one or more products related to each order. Each ORDLINE row is related to one PRODUCT row and to one ORDER row. The ORDLINE contains the quantity and price for each product ordered. The ORDLINE table gets the PROD_ID and ORL_QTY from the SHOPCART table. (See Table I.9.)

TABLE I.9

ORDLINE TABLE

ATTRIBUTE NAME	DESCRIPTION	PK/FK
ORL_ID	Order line ID—automatically generated.	PK
ORD_ID	Order ID from ORDER table.	FK
PROD_ID	Product ID.	FK
ORL_QTY	Quantity ordered.	
ORL_PRICE	Product price—after all promotions and discounts.	
ORL_TAX	Percentage tax rate applied to this product. Some products or customers may be tax-exempt. If the product/customer is taxable, the tax rate is obtained according to the STATE in the shipping address.	
SHIP_ID	Shipping company and type used to ship this product—for cases in which partial shipment is required.	FK
ORL_SHIPDATE	Date this product shipped.	

I-8f The SHOPCART Table

The SHOPCART table is a special table used by the website to store the products temporarily during the customer's shopping activities. To understand how the SHOPCART table works, you need to understand the online ordering process.

- When a customer first visits the website, (s)he may or may not log in. If (s)he logs in, the web server keeps his/her customer ID (CUST_ID) in memory.
- The customer browses the product catalog. If the customer is registered, (s)he sees the member prices; otherwise, (s)he sees regular prices (PROD_PRICE minus the respective discount percentage, PROD_PRICE_D1 or PROD_PRICE_D2).
- A shopping cart is automatically assigned to the user the first time (s)he orders a product by clicking on the “Add to Shopping Cart” or the “Order Now” button. A unique shopping cart ID is negotiated between the client’s browser and the merchant’s web server through use of a secure session. This ID lasts only until the customer successfully checks out, cancels the order, exits the website, or closes the browser.
- The shopping cart stores the PROD_ID and the quantity for each product selected by the customer.
- When the customer clicks the “Check Out” button, an Order confirmation screen is shown. This screen shows all of the product details for the product(s) placed in the shopping cart.
- When the customer accepts the order, (s)he is shown another screen in which to enter the shipping and payment information. The customer then confirms the order.
- After the customer has confirmed the order, the web server requests a transaction confirmation from the credit card company. The completion of this process may take from 10 to 60 seconds.
- Once the confirmation has been received, the ORDER data and the ORDLINE data are saved.
- The SHOPCART data are deleted.

The SHOPCART structure is shown in Table I.10.

TABLE I.10		
SHOPCART TABLE		
ATTRIBUTE NAME	DESCRIPTION	PK/FK
CART_ID	Shopping cart unique ID—automatically generated.	PK
CART_PROD_ID	Product ID—a copy of the PROD_ID value. Because the SHOPCART table has the potential of becoming a high-traffic table with many add and delete operations, you do not want to relate it to the PRODUCT table for performance reasons. Remember, these values will be automatically copied to the ORDLINE table when the customer transaction is processed. However, a relationship could be established if so desired.	
CART_QTY	Quantity ordered.	

I-8g The PMTTYPE Table

This table contains one row for each payment method accepted by the merchant. (See Table I.11.)

TABLE I.11

PMTTYPE TABLE

ATTRIBUTE NAME	DESCRIPTION	PK/FK
PMT_ID	Payment type ID—automatically generated.	PK
PMT_NAME	Name: Visa, MasterCard, American Excess, Net30.	
PMT_MCHNT_ID	Merchant ID—used by payment processing systems; given to the merchant when it registers with a credit card company.	
PMT_NOTES	Additional notes.	

I-8h The SHIPTYPE Table

This table contains one row for each shipping method supported by the merchant. (See Table I.12.)

TABLE I.12

SHIPTYPE TABLE

ATTRIBUTE NAME	DESCRIPTION	PK/FK
SHIP_ID	Shipping type ID—automatically generated.	PK
SHIP_NAME	Name: UPS Next Day, UPS Three Days, FedEx Overnight, and so on.	
SHIP_COST	Shipping cost per weight unit—is dependent on the formula used by the shipping company. Most are based on the shipping zip code and the size and weight of the products being shipped. Therefore, it's very likely that you are going to need additional attributes in this table as you develop the design.	
SHIP_NOTES	Additional shipping notes.	

I-8i The TAXRATE Table

This table contains the different sales tax rates used in each state. Note that the sales tax is not applied to every order, only to orders in those states in which the merchant is required by the state to collect taxes on products sold in the state. How is the sales tax requirement determined? Federal and state regulations determine the requirement. Normally, the sales tax determination could be based on the shipping address. However, the customer's billing address or the credit card billing address may also be used for this purpose. This table is related to the STATE table. Also, tax-exempt institutions are not charged tax. (See Table I.13.)

TABLE I.13

TAXRATE TABLE

ATTRIBUTE NAME	DESCRIPTION	PK/FK
STATE_ID	State ID from the STATE table—required.	PK, FK
TAX_RATE	Percent sales tax rate applied—required.	
TAX_NOTES	Additional notes, such as reason for the tax charge.	

I-8j The STATE Table

This table contains one entry for each state or country. This table is related to the TAX-RATE table. The table may contain modified entries for countries that use postal regions or other identifiers. This table can also contain a COUNTRY field for businesses that have multinational offices. (See Table I.14.)

TABLE I.14		
STATE TABLE		
ATTRIBUTE NAME	DESCRIPTION	PK/FK
STATE_ID	State ID—automatically generated.	PK
STATE_NAME	Name of the state—required.	

I-8k The PROMOTION Table

This table is used to represent special sales promotions. It contains one entry for each sale or promotion offered by the merchant. All promotions have a start and end date. Some promotions apply to one product line or to a specific product. Some promotions offer a percent discount; other promotions, such as vouchers, have a specific face amount. You could combine the attributes shown in Table I.15 in many ways to represent different promotions.

TABLE I.15		
PROMOTION TABLE		
ATTRIBUTE NAME	DESCRIPTION	PK/FK
PROMO_ID	Promotion ID—automatically generated.	PK
PROMO_NAME	Name of the promotion: Summer sale, Christmas sale, Voucher.	
PROMO_DATE	Date the promotion was introduced.	
PROMO_BEGDATE	Date promotion begins.	
PROD_ENDDATE	Date promotion ends.	
PTYPE_ID	Product type ID—optional. Type of product(s) affected by the promotion.	FK
PROD_ID	Product(s) affected by the promotion—optional.	FK
PROMO_MINQTY	Minimum purchase quantity required for promotion—optional.	
PROMO_MAXQTY	Maximum purchase quantity to which the promotion applies.	
PROMO_MINPUR	Minimum total purchase cost required for promotion—optional.	
PROMO_PCTDISC	Percent discount of promotion—optional.	
PROMO_DOLLAR	Dollar amount of promotion—optional.	
PROMO_CEILING	Maximum value of promotion (if percentage)—optional.	

I-8l The PRICEWATCH Table

Many e-commerce websites offer a “pricewatch” service. This service sends an email to a customer when the price of a product is below or equal to a price that was preselected by the customer. The PRICEWATCH table implements this feature. (See Table I.16.) A variation of this table can also be used for reverse bids.

TABLE I.16

PRICEWATCH TABLE

ATTRIBUTE NAME	DESCRIPTION	PK/FK
PW_ID	Pricewatch ID—automatically generated.	PK
PW_DATE	Date and time when the row was inserted in the table.	
CUST_ID	Customer ID—optional; service is to be offered to all prospective visitors as well as registered customers.	FK
PW_CUST_NAME	Name of customer—required; manually entered or automatically copied from CUSTOMER data.	
PW_CUST_EMAIL	Customer email—required; email address to send email notification.	
PW_ENDDATE	Date the pricewatch stops—optional; customer has the option of entering this date.	
PROD_ID	The product for which the pricewatch is done—required.	FK
PW_LOWPRICE	The price at which the customer wants to be informed; if the product price is equal to or less than this value, the system will send an email to the customer.	

I-8m The PRODPRIICE Table

The PRODPRIICE table is used to manage multilevel pricing. Some e-commerce sites offer multiple prices depending on the order quantity. For example, if you purchase one to five (inclusive) pairs of shoes, the price per pair might be \$39.95. However, if you purchase six or more pairs, the price per pair might drop to \$35.95. This table is in a 1:M relationship with the PRODUCT table. (See Table I.17.) If multilevel pricing is used, the PROD_PRICE in the PRODUCT table is not used.

TABLE I.17

PRODPRIICE TABLE

ATTRIBUTE NAME	DESCRIPTION	PK/FK
PROD_ID	Product ID from the PRODUCT table.	PK, FK
PROD_QTYFROM	Product purchase quantity minimum value—required; for example, 1 or 6 or 11.	PK
PROD_QTYTO	Product purchase quantity maximum revalue—required; for example, 5 or 10 or 9999.	PK
PROD_PRICE	Price for the quantity range—required.	

Key Terms

asymmetric or public-key encryption, I-20	encryption, I-20	proxy server firewall, I-23
authentication, I-19	encryption key, I-20	public key, I-20
business to business (B2B), I-7	Extensible Markup Language (XML), I-4	resource security, I-22
business to consumer (B2C), I-7	firewall, I-23	Secure Electronic Transaction (SET), I-25
certification authority (CA), I-19	gateway server firewall, I-23	Secure Hypertext Transfer Protocol (S-HTTP), I-22
Data Encryption Standard (DES), I-20	government to business (G2B), I-7	Secure Sockets Layer (SSL), I-21
denial-of-service, I-23	government to consumer (G2C), I-7	security, I-17
digital cash, I-24	hacker, I-23	state inspection firewall, I-23
digital certificate, I-19	intrabusiness, I-7	symmetric or private-key encryption, I-20
digital signature, I-20	intranets, I-2	Transport Layer Security (TLS), I-21
Domain Name Service (DNS), I-13	packet filter firewall, I-23	value chain, I-8
electronic commerce (e-commerce), I-2	Pretty Good Privacy (PGP), I-20	virus, I-22
Electronic Data Interchange (EDI), I-3	privacy, I-17	
	private key, I-20	

Review Questions

1. What does e-commerce mean, and how did it evolve?
2. Identify and briefly explain five advantages and five disadvantages of e-commerce.
3. Define and contrast B2B and B2C e-commerce styles.
4. Describe and give an example of each of the two principal B2B forms.
5. Describe e-commerce architecture, then briefly describe each of its components.
6. What types of services are provided by the bottom layer of the e-commerce architecture?
7. Name and explain the operation of the main building blocks of the Internet and its basic services.
8. What does business enabling do? What services layer does it provide? Give six examples of business-enabling services.
9. What is the definition of *security*? Explain why security is so important for e-commerce transactions.
10. Give an example of an e-commerce transaction scenario. What three things should security be concerned with in this e-commerce transaction?
11. You are hired as a resource security officer for an e-commerce company. Briefly discuss what technical issues you must address in your security plan.

Problems

1. Use the Internet at your university computer lab or home to research the scenarios described in Problems 1–10. Then work through the following problems:
 - b. What websites did you visit?
 - c. Classify each site (B2B, B2C, and so on).
 - d. What information did you collect? Was the information useful? Why or why not?
 - e. What decision(s) did you make based on the information you collected?
2. Research—and document—the purchase of a new car. Based on your research, explain why you plan to buy this car.
3. Research—and document—the purchase of a new house.
4. You are in the market for a new job. Search the web for your ideal job. Document your job search and your job selection.
5. You need to do your taxes. Download IRS form 1040 and look for online tax processing help, documenting your search.
6. Research the purchase of a 20-year level term life insurance policy and report your findings.
7. Research—and document—the purchase of a new computer.
8. Vacation time is almost here! Research—and document—the destination(s) and activities of next summer's vacation.
9. You have some money to invest. Research—and document—mutual funds information for investment purposes. Report your investment decision(s) based on the research you conduct.

Appendix J

Web Database Development with ColdFusion

Preview

This appendix examines the basics of web database development with ColdFusion, an important web application server tool for creating web database front ends. This appendix also explores some of the reasons why and how web application development differs from more traditional database application development.

Data Files and Available Formats

MS Access **Oracle** **MS SQL** **My SQL**

MS Access **Oracle** **MS SQL** **My SQL**

Orderdb



Data Files Available on cengagebrain.com

J-1 Using a Web-to-Database Production Tool: ColdFusion

To understand how web-to-database interfaces work, you need to know how they are created and to see them in action. In this section, you have a chance to try Adobe ColdFusion, one of a new breed of products known as web application servers. A **web application server** is a middleware application that expands the functionality of web servers by linking them to a wide range of services, such as databases, directory systems, and search engines. The web application server also provides a consistent run-time environment for web applications.

ColdFusion application middleware can be used to:

- Connect to and query a database from a webpage.
- Present database data in a webpage using various formats.
- Create dynamic web search pages.
- Create webpages to insert, update, and delete database data.
- Define required and optional relationships.
- Define required and optional form fields.
- Enforce referential integrity in form fields.
- Use simple and nested queries and form select fields to represent business rules.



Note

Although ColdFusion has a wide range of features, the purpose of this section is to show you how to create and use a simple, yet useful web-to-database interface. You can learn additional ColdFusion features by tapping into its detailed and well-organized online documentation (www.adobe.com).

ColdFusion has several important characteristics:

- It is a powerful and stable software product that can be used to produce and support even the most complex web-to-database access solutions.
- In spite of its power, it is a developer- and user-friendly product. ColdFusion has a strong and growing corporate presence. Using ColdFusion, you can get some hands-on experience with the web-to-database environment, while improving the marketability of your knowledge.
- Adobe offers a free 30-day evaluation version of the latest ColdFusion software, which can be downloaded from www.adobe.com. Because ColdFusion includes a complete set of online documentation with full working demo applications that illustrate all of the functionality of the product, you will incur no documentation charges.

ColdFusion is, of course, not the only player in the web application server market, but products from different vendors tend to have many similar features.

web application server

A middleware application that expands the functionality of web servers by linking them to a wide range of services, such as databases, directory systems, and search engines.

Web application servers provide features such as:

- An integrated development environment with session management and support for persistent application variables.
- Security and authentication of users through user IDs and passwords.
- Computational languages to represent and store business logic in the application server.
- Automatic generation of HTML pages integrated with Java, JavaScript, VBScript, ASP, and so on.
- Performance and fault-tolerant features.
- Database access with transaction management capabilities.
- Access to multiple services, such as file transfers (FTP), database connectivity, email, and directory services.

All of these products offer the ability to connect web servers to multiple data sources and other services. These products vary in terms of the range of available features, robustness, scalability, ease of use, compatibility with other web and database tools, and extent of the development environment.

J-1a How ColdFusion Works

ColdFusion has been described as a full-fledged web application server that provides hooks to databases, email systems, directory systems, search engines, and so on. To do its job, ColdFusion provides a server-side markup language (HTML extensions, or **tags**) known as the **ColdFusion Markup Language (CFML)**, which is used to create ColdFusion application pages known as *scripts*. A **script** is a series of instructions executed in *interpreter* mode. The script is a plain-text file that is not compiled like COBOL, C++, or Java. ColdFusion scripts contain the code that is required to connect, query, and update a database from a web front end.

ColdFusion scripts contain a combination of HTML, ColdFusion tags, and when necessary, Java, JavaScript, or VBScript. ColdFusion tags start with <CF and may include an opening and closing component such as <CFQUERY> (begin a query) and </CFQUERY> (end a query). ColdFusion scripts are saved in files with .cfm extensions. When a client browser requests a .cfm page from the web server, the ColdFusion application server executes the .cfm script instructions and sends the resulting output—in HTML format—to the web server. The web server then sends the document to the client computer for display. Figure J.1 shows the application server components and actions.

tag

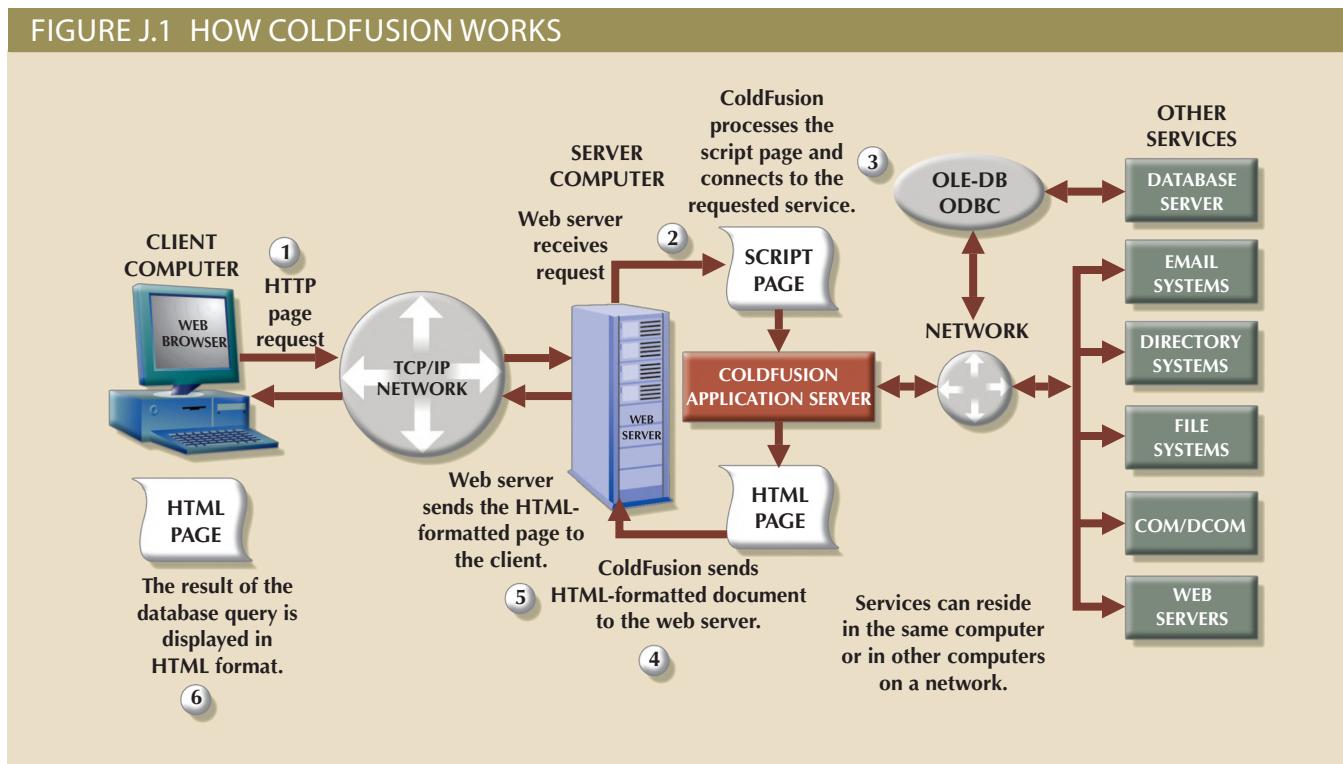
In markup languages such as HTML and XML, a command inserted in a document to specify how the document should be formatted. Tags are used in server-side markup languages and interpreted by a web browser for presenting data.

ColdFusion Markup Language (CFML)

A server-side markup language (HTML extensions or tags) that is used to create ColdFusion application pages known as *scripts*.

script

A programming language that is not compiled, but is interpreted and executed at run time.



J-1b The Orderdb Sample Database

To illustrate how ColdFusion can be used to provide the web-to-database interface, a small Microsoft Access database named Orderdb will be used. The following sections will guide you through the creation of several ColdFusion scripts designed to select, insert, update, and delete data from the Orderdb database.

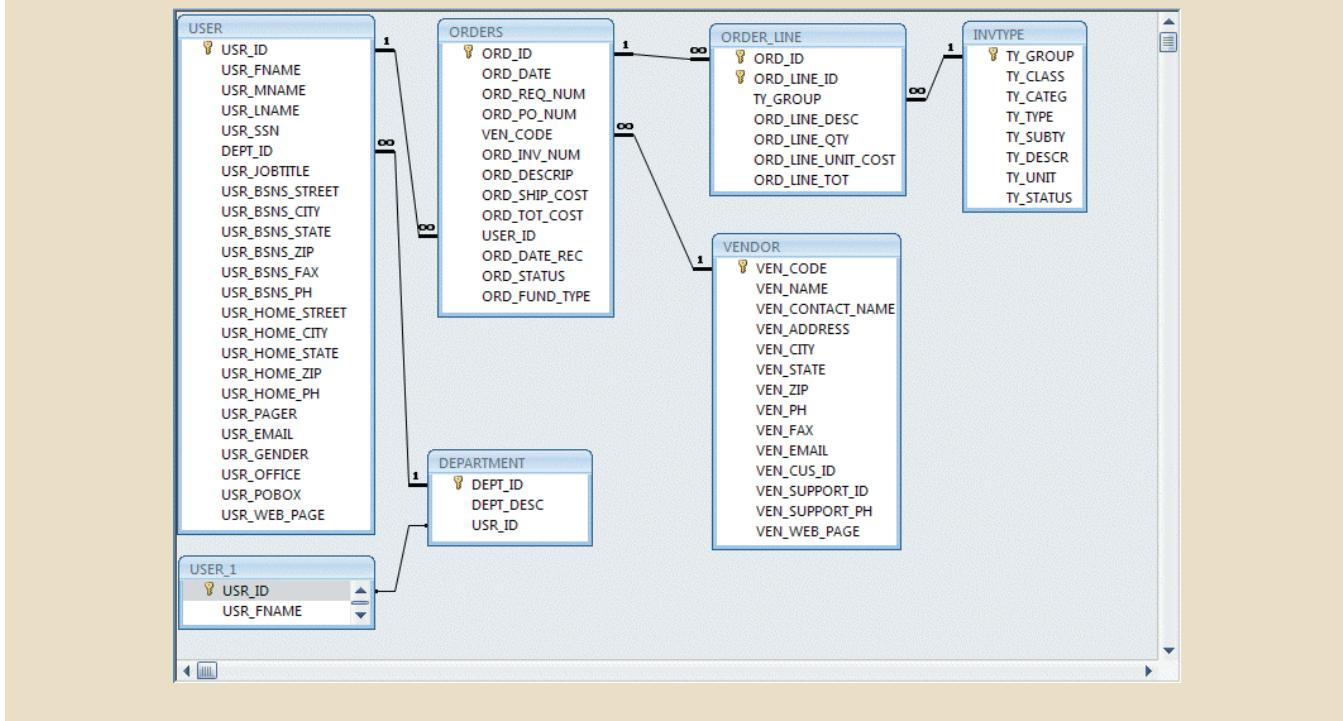


Note

To focus your efforts on the use of CFML to access databases, these exercises assume that you are familiar with basic HTML tags and the HTML editing process. The examples shown in this appendix can be created using any standard text editor such as Notepad.exe.

The Orderdb database, whose relational diagram is shown in Figure J.2, was designed to track the purchase orders placed by users in a multidepartment company.

FIGURE J.2 THE ORDERDB DATABASE'S RELATIONAL DIAGRAM



Note

The database and script files used in this appendix are located on cengagebrain.com. The database name is **orderdb.mdb**. Note that this database has been saved in MS Access 2000 format. **Please do not change the database format to a newer version.** Unless you have the latest MS Access ODBC/OLE drivers, changing the format could render the database inaccessible to the ColdFusion scripts. The orderdb.mdb is accessed using the ODBC data source name “**RobCor**”.

As you examine Figure J.2, note that the database contains the following tables: **USER**, **DEPARTMENT**, **VENDOR**, **INVTYPE**, **ORDERS**, and **ORDER_LINE**. The relationships between the tables are derived from the following business rules:

- A department employs many users.
- A department may be managed by one of those users.
- Each user belongs to one department, and each department can have many users.
- Each department may have a department manager. (That is, a department manager is optional.)
- Each order is placed to only one vendor, and each vendor can receive many orders.
- Each order contains one or more order lines.
- Each order line refers to one inventory type.

USER_1 is a virtual component created by MS Access when *multiple relationships* between USER and DEPARTMENT are set. MS Access created the USER_1 virtual table to represent the “USER *manages* DEPARTMENT” relationship. This is a one-to-one *optional* relationship, thus allowing the USR_ID field in the DEPARTMENT table to be null. (This relationship will be used to illustrate how you can manage optional relationships within a web interface.)

J-1c Creating a Simple Query with CFQUERY and CFOOUTPUT

Let’s begin by creating a simple script to produce a query that will list all of the vendors in the VENDOR table. This script will perform two tasks:

1. Query the database, using standard SQL to retrieve a data set that contains all records found in the VENDOR table.
2. Format all of the records generated in Step 1 in HTML so they are included in the page that is returned to the client browser.

Script J.1 contains the required code.



Note

If you install the ColdFusion demo, you can run this script (and all subsequent scripts) by pointing the browser to the web server address. For example, if your computer is the web server, you can use **<http://127.0.0.1/robcor/rc-1.cfm>** as your web address. If your web server is a different computer selected by your instructor, use the address supplied by your instructor. You can also access a menu for all ColdFusion scripts by going to **<http://127.0.0.1/robcor>**.

As you examine Script J-1 (rc-1.cfm), note that its ColdFusion tags are CFQUERY (to query a database) and CFOOUTPUT (to display the data returned by the query). Note that the CFML and HTML tags are shown in different colors. Let’s take a closer look at these two CFML tags.

- <CFQUERY> tag (lines 4–6). This tag sets the stage for the database connection and the execution of the enclosed SQL statement. You should include all query statements *before* or *within* the document’s HTML header (<HEAD>) section. Using that procedure, the page will display the output on the client side *after all queries have been executed*. If you do not use that procedure, the browser will be perceived as “slow” because the page will start to display output and then pause to wait for additional data to arrive from the Web server. The CFQUERY tag requires the following parameters:
 - NAME = “*queryname*”. This is a mandatory parameter, whose name uniquely identifies the record set returned by the database query—in this case, *venlist*. You can have multiple queries, each with a unique name, in a single script.
 - DATASOURCE = “*datasourcename*”. This, too, is a mandatory parameter that uses the database name as defined in ODBC. Keep in mind that the database name is case sensitive. Therefore, if the database name is “*RobCor*”, do not use “ROBCOR” or “robcor”. You use the ColdFusion Administrator interface to define all data sources. Data sources can use ODBC, a native driver such as Oracle SQL*Net, or Microsoft OLE-DB (object linking and embedding).

SCRIPT J.1 A SIMPLE QUERY USING CFQUERY AND CFOUTPUT (RC-1.CFM)

```

1  <HTML>
2   <HEAD>
3     <TITLE>Rob & Coronel - ColdFusion Examples</TITLE>
4     <CFQUERY NAME="venlist" DATASOURCE="RobCor">
5       SELECT * FROM VENDOR ORDER BY VEN_CODE
6     </CFQUERY>
7   </HEAD>
8   <BODY BGCOLOR="LIGHTBLUE">
9     <H1>
10    <CENTER><B>Simple Query using CFQUERY and CFOUTPUT</B></CENTER>
11    <CENTER><B>(Vertical Output)</B></CENTER>
12  </H1>
13  <BR>
14  <HR>
15  <CFOUTPUT>
16    Your query returned #venlist.RecordCount# records
17 </CFOUTPUT>
18 <CFOUTPUT QUERY="venlist">
19 <PRE><B>
20   VENDOR CODE:      #VEN_CODE#
21   VENDOR NAME:      #VEN_NAME#
22   CONTACT PERSON:   #VEN_CONTACT_NAME#
23   ADDRESS:          #VEN_ADDRESS#
24   CITY:              #VEN_CITY#
25   STATE:             #VEN_STATE#
26   ZIP:               #VEN_ZIP#
27   PHONE:             #VEN_PH#
28   FAX:               #VEN_FAX#
29   E-MAIL:            #VEN_EMAIL#
30   CUSTOMER ID:      #VEN_CUS_ID#
31   SUPPORT ID:       #VEN_SUPPORT_ID#
32   SUPPORT PHONE:    #VEN_SUPPORT_PH#
33   VENDOR WEB PAGE:  #VEN_WEB_PAGE#
34 <HR></B></PRE>
35 </CFOUTPUT>
36 <FORM ACTION="rc-0.cfm" METHOD="post">
37   <INPUT TYPE="submit" VALUE="Main Menu ">
38 </FORM>
39 </BODY>
40 </HTML>

```

- SQL statement (line 5) is another mandatory parameter. In this case, the parameter is defined by the following query, but you could use any ODBC SQL-compliant statement:

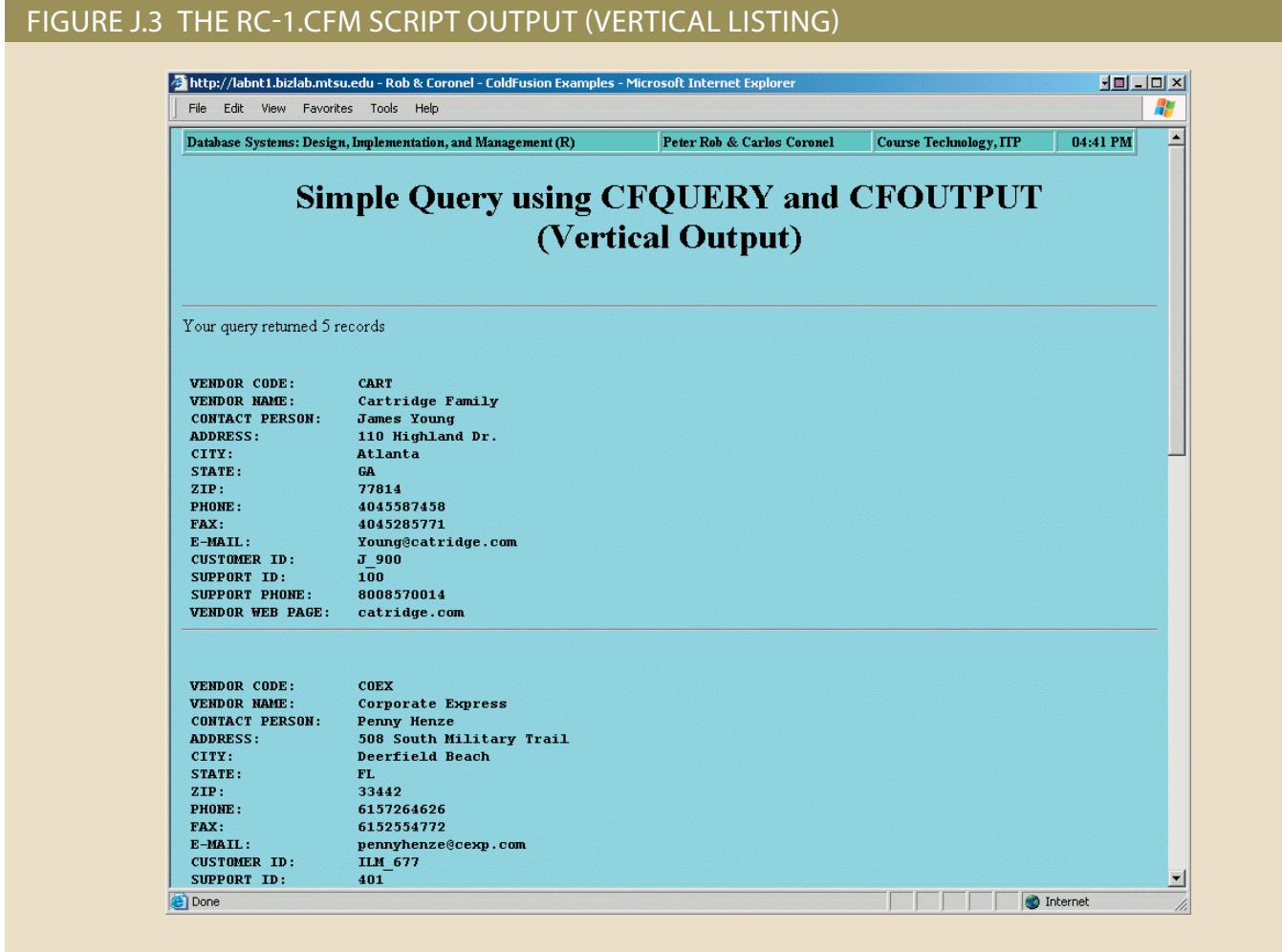

```
SELECT * FROM VENDOR ORDER BY VEN_CODE
```
- <CFOUTPUT> tag (lines 15–17 and 18–35). This tag is used to display the results from a CFQUERY or to call other ColdFusion variables or functions. Its parameters are:
 - QUERY = “*queryname*”. This is an optional parameter (see line 18). If you use a query name for a query that returns 10 rows, this tag will execute all commands between the opening and closing CFOUTPUT tags 10 times—one per row. In short, this tag works like a loop that is executed as many times as the number of rows in the named query set.
 - You can include any valid HTML tags or text within the opening and closing CFOUTPUT tags. ColdFusion uses pound signs (#) to reference query fields in the resulting query set or to call other ColdFusion functions or variables. When ColdFusion encounters a name enclosed within pound signs, it evaluates this named variable to verify that it is a field name of the named query, an internal variable, or a function. Following this evaluation, ColdFusion will replace the named variable with the value that corresponds to the query, the internal variable, or the function. In this case, line 16 is a call to a ColdFusion internal variable. When the query is

executed, this variable's value is the number of rows in the query output. The name of the query must precede the *RecordCount* keyword, and the two components must be separated by a period. For example, #*venlist.RecordCount*# is used to name the variable in line 16.

- Lines 19–34. These lines are repeated as a loop, one for each record returned in the named query. In this example, the loop is defined by *QUERY* = “*venlist*”. Note that in lines 20–33, the field names (enclosed in pound signs) will be replaced by the actual values of the fields that are returned by the query.

The output produced by *rc-1.cfm* is shown in Figure J.3.

FIGURE J.3 THE RC-1.CFM SCRIPT OUTPUT (VERTICAL LISTING)



A variation of the just-described approach is found in Script J.2 (*rc-2.cfm*), in which the output is preformatted (using the HTML *<PRE>* tag) one row per record, using the *CFOUTPUT* tag.

SCRIPT J.2 CFQUERY WITH TABULAR CFOUTPUT (RC-2.CFM)

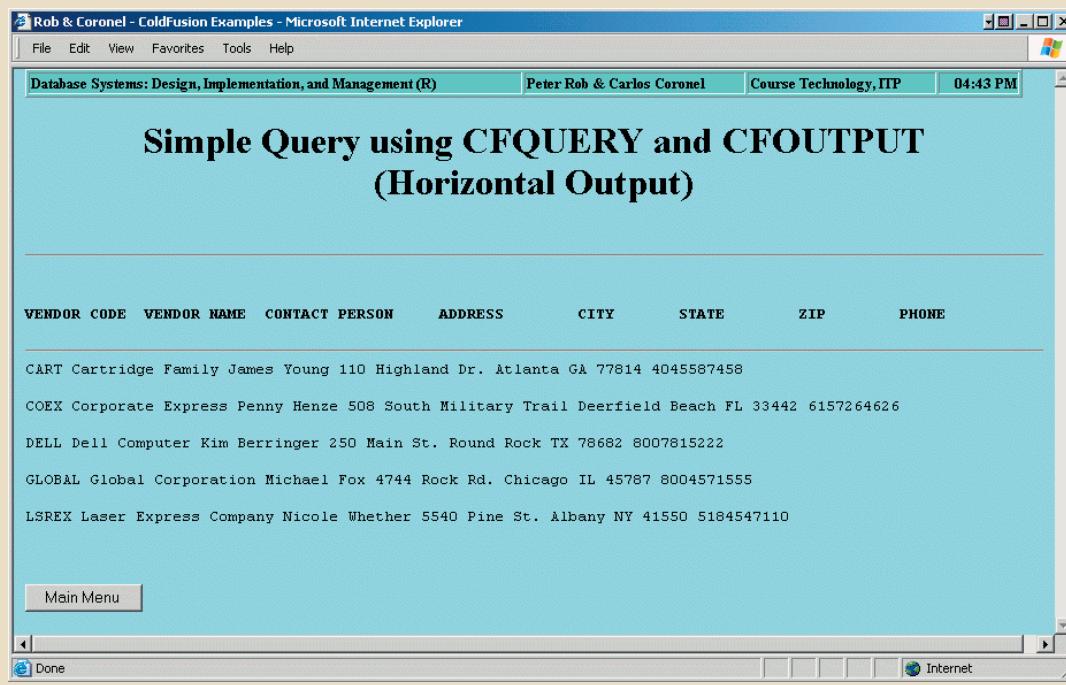
```

1  <HTML>
2   <CFQUERY NAME="VENDATA" DATASOURCE="RobCor">
3     SELECT * FROM VENDOR ORDER BY VEN_CODE
4   </CFQUERY>
5   <HEAD>
6     <TITLE>Rob & Coronel - ColdFusion Examples</TITLE>
7   </HEAD>
8   <BODY BGCOLOR="LIGHTBLUE">
9     <H1>
10    <B><CENTER>Simple Query using CFQUERY and CFOUTPUT</CENTER></B>
11    <B><CENTER>(Horizontal Output)</CENTER></B>
12  </H1>
13  <BR>
14  <HR>
15  <PRE>
16  <B>
17  VENDOR CODE  VENDOR NAME  CONTACT PERSON      ADDRESS          CITY        STATE        ZIP        PHONE
18 </B>
19 <HR>
20  <CFOUTPUT QUERY="VENDATA">
21  #VEN_CODE#  #VEN_NAME#  #VEN_CONTACT_NAME# #VEN_ADDRESS#  #VEN_CITY#  #VEN_STATE#  #VEN_ZIP#  #VEN_PH#<BR>
22  </CFOUTPUT>
23  </PRE>
24  <FORM ACTION="rc-0.cfm" METHOD="post">
25    <INPUT TYPE="submit" VALUE="Main Menu ">
26  </FORM>
27  </BODY>
28 </HTML>

```

The output of script rc-2.cfm is shown in Figure J.4.

FIGURE J.4 THE RC-2.CFM SCRIPT OUTPUT (HORIZONTAL LISTING)



J-1d Creating a Simple Query with CFQUERY and CFTABLE

As you can see in Figure J.4, the output produced by CFOUTPUT is not aligned. To give the output a more polished look, the vendor list can be displayed in tabular format, based on the CFTABLE tag. That tag will automatically create a tabular output in which each row in the data set is placed in a row in the table. The source code for this example is stored in Script J.3 (rc-3.cfm).

SCRIPT J.3 CFQUERY WITH CFTABLE (RC-3.CFM)

```

1  <HTML>
2  <HEAD>
3  <TITLE>Rob & Coronel - ColdFusion Examples</TITLE>
4  <CFQUERY NAME="VENDATA" DATASOURCE="RobCor">
5      SELECT * FROM VENDOR ORDER BY VEN_CODE
6  </CFQUERY>
7  </HEAD>
8  <BODY BGCOLOR="LIGHTBLUE">
9  <H1>
10 <B><CENTER>Simple Query Using CFQUERY and CFTABLE</CENTER></B>
11 </H1>
12 <BR>
13 <TABLE BGCOLOR="Silver" BORDERCOLOR="Fuchsia" FRAME="BORDER">
14 <TR><HR></TR>
15 <TR>
16 <CFTABLE QUERY="VENDATA" STARTROW="1" COLSPACING="1" COLHEADERS>
17 <CFCOL HEADER=<B>CODE</B> WIDTH="8" TEXT="#VEN_CODE#">
18 <CFCOL HEADER=<B>VENDOR_NAME</B> WIDTH="25" TEXT="#VEN_NAME#">
19 <CFCOL HEADER=<B>CONTACT_PERSON</B> WIDTH="14" TEXT="#VEN_CONTACT_NAME#">
20 <CFCOL HEADER=<B>ADDRESS</B> WIDTH="20" TEXT="#VEN_ADDRESS#">
21 <CFCOL HEADER=<B>CITY</B> WIDTH="10" TEXT="#VEN_CITY#">
22 <CFCOL HEADER=<B>STATE</B> WIDTH="2" TEXT="#VEN_STATE#">
23 <CFCOL HEADER=<B>ZIP</B> WIDTH="5" TEXT="#VEN_ZIP#">
24 <CFCOL HEADER=<B>PHONE</B> WIDTH="10" TEXT="#VEN_PH#">
25 <CFCOL HEADER=<B>FAX</B> WIDTH="10" TEXT="#VEN_FAX#">
26 <CFCOL HEADER=<B>E-MAIL</B> WIDTH="10" TEXT="#VEN_EMAIL#">
27 <CFCOL HEADER=<B>CUSTOMER_ID</B> WIDTH="8" TEXT="#VEN_CUS_ID#">
28 <CFCOL HEADER=<B>SUPPORT_PHONE</B> WIDTH="6" TEXT="#VEN_SUPPORT_ID#">
29 <CFCOL HEADER=<B>WEB PAGE</B> WIDTH="14" TEXT="#VEN_WEB_PAGE#">
30 </CFTABLE>
31 </TR>
32 <TR>
33 <FORM ACTION="rc-0.cfm" METHOD="post">
34     <INPUT TYPE="submit" VALUE="Main Menu ">
35 </FORM>
36 </TR>
37 </TABLE>
38 </BODY>
39 </HTML>

```

The output of script rc-3.cfm is shown in Figure J.5.

The rc-3.cfm script's CFTABLE ColdFusion tag contents are as follows:

- <CFTABLE> tag (line 16). This tag, used to display the results from a CFQUERY (lines 4–6) in a tabular format, requires the following parameters:
 - *QUERY* = “*queryname*”. A required parameter that uses the name of the query that generated the data set to be displayed in tabular format.
 - *STARTROW* = “1”. An optional parameter that is used to tell ColdFusion which query row will be the table’s first row. This parameter is particularly useful when your query returns many rows and you do not want to display them all in one long page. Instead, you can display them in multiple pages, each page displaying the number of rows defined by the parameter. For example, if you want to list 10 rows per page, the starting row will be 1 for the first page, 11 for the second page, 21 for the third, and so on.
 - *COLSPACING* = “1”. An optional parameter that is used to define the number of spaces to be placed between columns.
 - *COLHEADERS*. An optional parameter that will make the first row in the table a row header. This row header contains the name of each column, using the values defined in the CFCOL tags.
- The CFCOL tag (lines 17–29) is used to define each table column, using the following parameters:
 - *HEADER* = “*header text*”. This is the header text that will appear in the table’s header row for each of the displayed columns. The header text can include HTML tags.
 - *WIDTH* = “*x*”. This parameter defines the column width.
 - *TEXT* = “#*queryfieldname*#”. This is the actual value to be placed in the selected column. For example, line 17 will cause ColdFusion to replace #*VEN_CODE*# with the actual values retrieved by the query for this field.

FIGURE J.5 THE RC-3.CFM SCRIPT OUTPUT (FORMATTED HORIZONTAL LISTING)

CODE	VENDOR_NAME	CONTACT_PERSON	ADDRESS	CITY	ST	ZIP	PHONE	FAX
CART	Cartridge Family	James Young	110 Highland Dr.	Atlanta	GA	77814	4045587458	4045285771
COEX	Corporate Express	Penny Henze	508 South Military T	Deerfield	FL	33442	6157264626	6152554772
DELL	Dell Computer	Kim Berringer	250 Main St.	Round Rock	TX	78682	8007815222	8008425888
GLOBAL	Global Corporation	Michael Fox	4744 Rock Rd.	Chicago	IL	45787	8004571555	8007872421
LSREX	Laser Express Company	Nicole Whether	5540 Pine St.	Albany	NY	41550	5184547110	5185754570

J-1e Creating a Dynamic Search Page

At this point, you have seen how the CFQUERY, CFOUTPUT, and CFTABLE tags are used to send the SQL statement to the database to retrieve a data set. Given the script files used thus far, the query will always retrieve the same records from the VENDOR table. (Because the SQL statement is “hard-coded” inside the page, it cannot be changed unless the SQL code is manually edited each time it is to be used to generate a different query output.) Such a static output display clearly limits the query’s usefulness.

To create a practical dynamic query environment, you can create a dynamic search query in which the query search condition can be changed at the user’s option—without requiring script page editing. To demonstrate the process, two fields, vendor code and vendor state, will be used to search for user-specified vendor records. (Naturally, you can create a search form that uses as many fields as you think are necessary.)

To perform a dynamic query over the web, you must complete these two steps:

1. Create a script that will generate a form that will be used to enter the criteria used in the search. In other words, the form allows the user to enter the parameter values that are to be used in the query statement.
2. Create a script that will execute the query and display the results based on the parameters that are passed to it by the script created in Step 1.

The script required to complete Step 1 is listed in Script J.4A (rc-4a.cfm).

SCRIPT J.4A DYNAMIC SEARCH QUERY: CRITERIA-ENTRY FORM (RC-4A.CFM)

```

1  <HTML>
2   <CFQUERY NAME="STATELIST" DATASOURCE="RobCor">
3     Select VEN_STATE from VENDOR Order by VEN_STATE
4   </CFQUERY>
5   <HEAD>
6     <TITLE>Rob & Coronel - ColdFusion Examples</TITLE>
7   </HEAD>
8   <BODY BGCOLOR="LIGHTBLUE">
9     <H1>
10    <CENTER><B>Dynamic Search Query: Criteria Entry Form</B></CENTER>
11  </H1>
12  <FORM ACTION="rc-4b.cfm" METHOD=POST>
13    <TABLE ALIGN="CENTER" BGCOLOR="Silver">
14      <TR>
15        <TD ALIGN="right">VEN_CODE</TD>
16        <TD>
17          <INPUT TYPE = "text" NAME="VEN_CODE" SIZE="10" MAXLENGTH="10">
18        </TD>
19        <TR>
20          <TD ALIGN="right">VEN_STATE</TD>
21          <TD><SELECT NAME="VEN_STATE" SIZE=1>
22            <OPTION SELECTED VALUE="ANY">ANY
23            <CFOUTPUT QUERY="STATELIST">
24              <OPTION VALUE="#STATELIST.VEN_STATE#">#VEN_STATE#
25            </CFOUTPUT>
26          </SELECT>
27        </TD>
28      </TR>
29      <TR>
30        <TD ALIGN="right" VALIGN="middle">
31          <INPUT TYPE="submit" VALUE="Search">
32        </TD>
33      </TR>
34      <TD ALIGN="right" VALIGN="middle">
35        <FORM ACTION="rc-0.cfm" METHOD="post">
36          <INPUT TYPE="submit" VALUE="Main Menu ">
37        </FORM>
38      </TD>
39    </TR>
40  </TABLE>
41  </BODY>
42  </HTML>
```

The rc-4a.cfm script output is shown in Figure J.6.

FIGURE J.6 THE RC-4A.CFM SCRIPT OUTPUT (STATE SEARCH CRITERIA ENTRY FORM)

The screenshot shows a Microsoft Internet Explorer window with the title bar "Rob & Coronel - ColdFusion Examples - Microsoft Internet Explorer". The menu bar includes File, Edit, View, Favorites, Tools, and Help. The toolbar has icons for Back, Forward, Stop, Home, and Favorites. The status bar shows "Done" and "Internet". The main content area is titled "Dynamic Search Query: Criteria Entry Form". It contains two input fields: "VEN_CODE" (text box) and "VEN_STATE" (drop-down menu set to "ANY"). Below the fields are two buttons: "Search" and "Main Menu". The background of the form is light blue.

As you examine Figure J.6, note that the rc-4a.cfm script generates a form in which the user enters the search parameters. Let's follow the rc-4a.cfm script to see how it works.

- In lines 2–4, all existing values are retrieved from the *VEN_STATE* field in the VENDOR table. This query effectively tells you what states are represented in the vendor table. If no vendor is listed for a given state, that state will not be shown in the returned data set. This query is named “*STATELIST*”, and it will be used later in your form.
- In lines 12–32, the form is defined. Note that when the user clicks the “submit” button, the rc-4b.cfm script (shown later) will be called to receive the form’s variables, *VEN_CODE* and *VEN_STATE*.
- Line 17 presents the first input text box to let the user enter a value for the *VEN_CODE* form variable. This value will be used in the SQL statement to search for all records with matching *VEN_CODE* values.
- Lines 21–26 create a drop-down SELECT box to let the user pick the state to be used in the vendor table query. This selection will later be passed to the rc-4b.cfm script.
- In lines 23–25, the CFOUTPUT tag is used to build the selection options, using the states that occur in the VENDOR table. The default (“SELECTED”) option shown in line 22 gives the user the ability to search without selecting any particular state. If line 22 is not included, there will be no way to limit the search to only a vendor code, nor will the user be able to display all vendors from all states. To limit the search, the *VEN_STATE* form field must have the option to contain a null “value.” In short, line 22’s default condition provides crucial flexibility.

When the user clicks the Search button, the rc-4b.cfm script is executed, and the two form variables (VEN_CODE and VEN_STATE) are passed to the rc-4b.cfm script. The rc-4b.cfm script is shown in Script J.4B.

SCRIPT J.4B THE VENDOR SEARCH RESULTS (RC-4B.CFM)

```

1  <HTML>
2   <CFQUERY NAME="SearchVendor" DATASOURCE="RobCor">
3     SELECT VEN_CODE, VEN_NAME, VEN_CONTACT_NAME, VEN_ADDRESS, VEN_CITY, VEN_STATE, VEN_PH
4     FROM VENDOR
5     WHERE 0=0
6     <CFIF #FORM.VEN_CODE# IS NOT "">
7       AND VENDOR.VEN_CODE LIKE '%#FORM.VEN_CODE#%'
8     </CFIF>
9     <CFIF #FORM.VEN_STATE# IS NOT "ANY">
10      AND VENDOR.VEN_STATE LIKE '%#FORM.VEN_STATE#%'
11    </CFIF>
12    ORDER BY VEN_CODE
13  </CFQUERY>
14  <HEAD>
15  <TITLE>Rob & Coronel - ColdFusion Examples</TITLE>
16  </HEAD>
17  <BODY BGCOLOR="LIGHTBLUE">
18  <H1>
19  <CENTER><B>Vendor Search Results</B></CENTER>
20  </H1>
21  <BR>
22  <CFTABLE QUERY="SearchVendor" STARTROW="1" COLSPACING="1" COLHEADERS>
23  <CFCOL HEADER=<B>VENDOR CODE</B>" WIDTH="14" TEXT="#VEN_CODE#">
24  <CFCOL HEADER=<B>VENDOR NAME</B>" WIDTH="20" TEXT="#VEN_NAME#">
25  <CFCOL HEADER=<B>CONTACT PERSON</B>" WIDTH="20" TEXT="#VEN_CONTACT_NAME#">
26  <CFCOL HEADER=<B>ADDRESS</B>" WIDTH="20" TEXT="#VEN_ADDRESS#">
27  <CFCOL HEADER=<B>CITY</B>" WIDTH="20" TEXT="#VEN_CITY#">
28  <CFCOL HEADER=<B>STATE</B>" WIDTH="2" TEXT="#VEN_STATE#">
29  <CFCOL HEADER=<B>PHONE</B>" WIDTH="20" TEXT="#VEN_PH#">
30  </CFTABLE>
31  <CFIF #SEARCHVENDOR.RECORDCOUNT# IS 0>
32    <H2>No records were found matching your criteria </H2>
33  <CFELSE>
34    <CFOUTPUT>Your search returned #SearchVendor.RecordCount# record(s).</CFOUTPUT>
35  </CFIF>
36  <FORM ACTION="rc-0.cfm" METHOD="post">
37    <INPUT TYPE="submit" VALUE="Main Menu ">
38  </FORM>
39  </BODY>
40  </HTML>

```

The rc-4b.cfm script output is shown in Figure J.7.

FIGURE J.7 THE RC-4B.CFM SCRIPT OUTPUT (VENDOR SEARCH RESULTS: ALL STATES)

VENDOR_CODE	VENDOR_NAME	CONTACT_PERSON	ADDRESS	CITY	ST_PHONE
CART	Cartridge Family	James Young	110 Highland Dr.	Atlanta	GA 4045587458
COEX	Corporate Express	Penny Henze	508 South Military T	Deerfield Beach	FL 6157264626
DELL	Dell Computer	Kim Berringer	250 Main St.	Round Rock	TX 8007815222
GLOBAL	Global Corporation	Michael Fox	4744 Rock Rd.	Chicago	IL 8004571555
LSREX	Laser Express Compan	Nicole Whether	5540 Pine St.	Albany	NY 5184547110

Your search returned 5 record(s.).

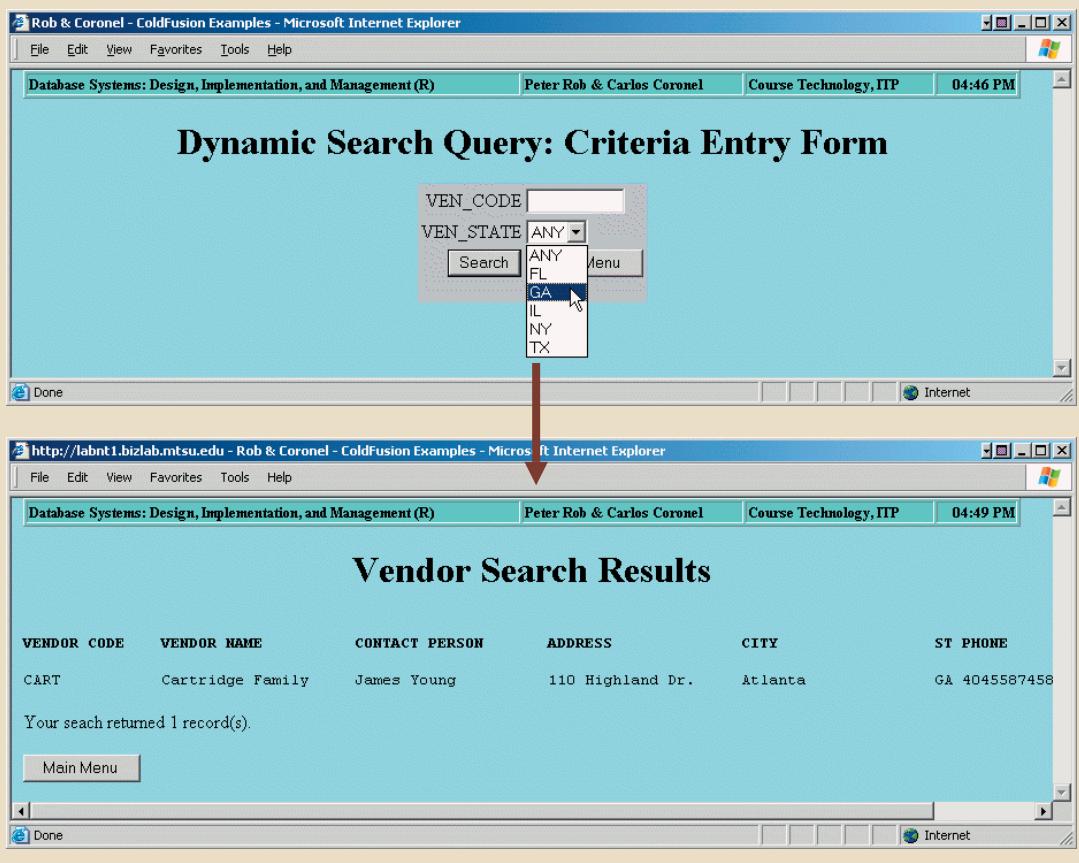
Main Menu

As you examine Script J.4B (rc-4b.cfm,) note how it references the form variables received from the calling script (rc-4a.cfm) and how it uses the CFIF tag to dynamically build SQL statements. Let's take a closer look at this script.

- Lines 2–13 are used to create the SQL statement and to query the database.
- Line 5 creates a dummy WHERE clause that will be used to anchor the query's conditional criteria. The “0=0” conditional criterion is the default condition that will list all records. This line is required to form the basis for additional conditions, using SQL's logical operators. Naturally, if no additional conditions are specified, the default value ensures that the query will list all records.
- Lines 6–8 use a CFIF tag to evaluate the VEN_CODE form field passed from the calling page. (In this case, the rc-4a.cfm script created the calling page.) If the field specified in line 6 is not null (“”), the condition specified in line 7 is added to the query.
- Lines 9–11 use the CFIF tag to evaluate the VEN_STATE form field passed from the calling page (rc-4a.cfm). If this field is not “ANY”, line 10 adds the “state” condition to the query. (Remember that “ANY” is the default selection for the rc-4a.cfm script's VEN_STATE field.) This default selection ensures that it is not necessary to search for any specific state. Therefore, the user can select a state to limit the query output to that state, or by not selecting a state, the user can generate the query output for all states.
- Lines 22–30 indicate that the CFTABLE and CFCOL tags are used to display the query result set in a tabular format.
- Finally, a CFIF statement is used to evaluate the number of records in the resulting record set and to display the appropriate message. For example, line 31 ensures that the “*No records were found matching your search criteria*” message is displayed in line 32 if the record set returns zero (0) rows. If the record count is not zero, line 34 generates a message that indicates the number of records returned in the query result set.

Figure J.8 shows the output for a dynamic search based on Vendors for the condition VEN_STATE = “GA”.

FIGURE J.8 THE VENDOR LIST FOR THE CONDITION VEN_STATE = “GA”



The dynamic search process clearly makes the web a viable end-user information-generation tool. However, to conduct *transactions*, you must be able to modify the database’s table contents. (For example, if you make a withdrawal from inventory, you must be able to update the inventory table; if you make a purchase, you must be able to generate an invoice record; and so on.) Therefore, the focus now will be on the basic procedures that may be used to insert, update, and delete data through web interfaces. Before you can develop web-based transaction applications, you need to know why and how the shortcomings of HTML and browsers affect data manipulation activities. Those shortcomings are a function of the web’s basic structure, which is often described as a so-called *stateless system*.

stateless system

A system in which a web server does not know the status of the clients communicating with it. The web does not reserve memory to maintain an open communications state between the client and the server.

J-1f The Web as a Stateless System

The web is said to be a stateless system. Simply put, the label **stateless system** indicates that at any given time, a web server does not know the status of any of the clients communicating with it. That is, there is no open communication line between the server and each client accessing it, which, of course, is impractical in a *worldwide web*! Instead, client and server computers interact in very short “conversations” that follow the request-reply model. For example, the browser is concerned only with the *current* page, so there is no way for the second page to know what was done in the first page. The only time the

client and server computers communicate is when the client requests a page—when the user clicks a link—and the server sends the requested page to the client. Once the client receives the page and its components, the client/server communication is ended. Therefore, although you may be browsing a page and *think* that the communication is open, you are actually just browsing the HTML document stored in the local cache (temporary directory) of the client browser. The server does not have any idea what the end user is doing with the document, what data is entered in a form, what option is selected, and so on. On the web, if you want to act on a client's selection, you need to jump to a new page (go back to the web server), thus losing track of whatever was done before!

Not knowing what was done before (or what a client selected before getting to this page) makes adding business logic to the web cumbersome. For example, suppose that you need to write a program that performs the following steps: display a data-entry screen, capture data, validate data, and save data. That entire sequence can be completed in a single COBOL program because COBOL uses a working storage section that holds in memory all variables used in the program. Now imagine the same COBOL program—but *each* section (PERFORM statement) is now a separate program! That is precisely how the web works. In short, the web's stateless nature means that extensive processing required by a program's execution cannot be done directly through a single webpage; the client browser's processing ability is limited by the lack of processing ability and the lack of a working storage area to hold variables used by all pages on a website.

Keep in mind that a web browser's function is to display a page on the client computer. The browser—through its use of HTML—does not have computational abilities beyond formatting output text and accepting form field inputs. Even when the browser accepts form field data, there is no way to perform immediate data entry validation. Therefore, to perform such crucial processing in the client, the web defers to other web programming languages such as Java, JavaScript, and VBScript. The browser most resembles a dumb terminal that displays only data and can perform only rudimentary processing such as accepting form data inputs. Most of the processing takes place at the server end—in this case, the web application server.

To circumvent the above-mentioned shortcomings of the web environment, most web application servers have session management capabilities that allow a web server to maintain status information for each client session in the server's memory. Each web server vendor has its own way to maintain that information. In the case of ColdFusion, session variables are declared using the <CFSET session.variablename=value> command syntax. For example, to declare a user-type session variable, you would say, <CFSET session.usertype = "ADMIN">. Then the variable would be available to all pages in the same client session. The client session starts the first time the client's browser connects to the web server; it ends when the client accesses a page outside the website, closes his/her browser, or stays idle for a given time-out period.

J-1g Inserting Data

In this section, you will create a data entry form to insert data in the DEPARTMENT table. In the following example, the DEPARTMENT table contains three fields: department ID and department description, which are required, and an optional manager user ID that references the USER table. Of course, *if the user enters a user ID, that ID must match a user ID in the USER table*. Given that basic scenario, let's see how ColdFusion can be used to establish basic server-side data validation for the required fields and how ColdFusion can implement and manage data entry for an optional relationship.

At least two pages are needed to accomplish the just-described tasks. The first page, generated by a script named rc-5a.cfm, creates a form to get the data. The second page, generated by a script named rc-5b.cfm, inserts the data in the table. Data validation will

take place at the server side. Let's first look at script J.5A (rc-5a.cfm). The script is followed by its output in Figure J.9.

SCRIPT J.5A INSERT QUERY—DATA ENTRY (RC-5A.CFM)

```

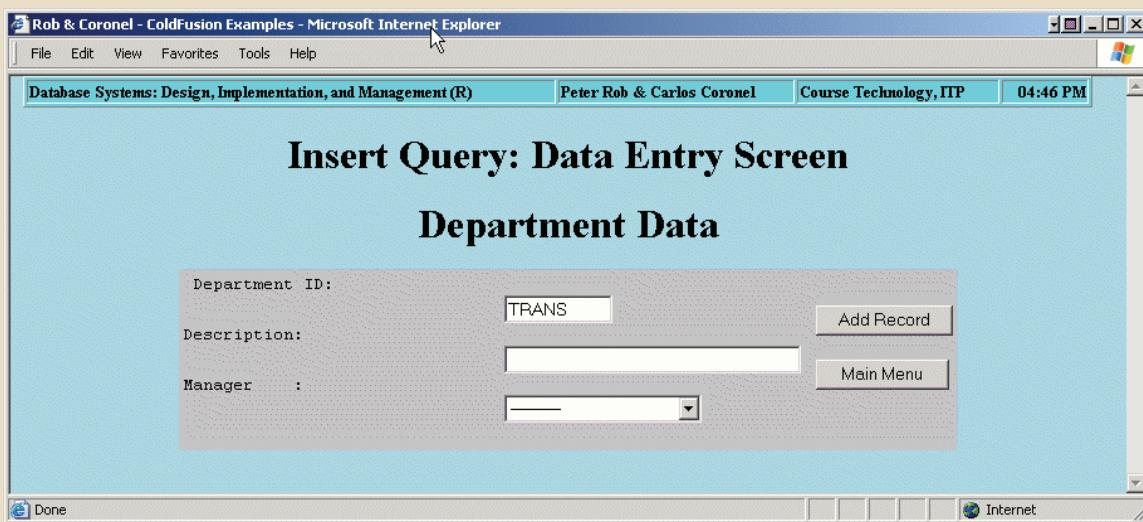
1  <HTML>
2  <HEAD>
3  <TITLE>Rob & Coronel - ColdFusion Examples</TITLE>
4  <CFQUERY NAME="USRLIST" DATASOURCE="RobCor">
5  SELECT USR_ID, USR_LNAME, USR_FNAME, USR_MNAME
6   FROM USER
7   WHERE USR_ID NOT IN (SELECT USR_ID FROM DEPARTMENT WHERE USR_ID > 0)
8   ORDER BY USR_LNAME, USR_FNAME, USR_MNAME
9  </CFQUERY>
10 </HEAD>
11 <BODY BGCOLOR="LIGHTBLUE">
12 <H1>
13 <CENTER><B>Insert Query: Data Entry Screen</B></CENTER></H1>
14 <FORM ACTION="rc-5b.cfm" METHOD="post">
15 <CENTER><H1>Department Data</H1></CENTER><!-- the following code defines required fields -->
16   <INPUT TYPE="hidden" NAME="DEPT_ID_required" VALUE="You must enter a DEPT_ID">
17   <INPUT TYPE="hidden" NAME="DEPT_DESC_required" VALUE="You must enter a description">
18   <TABLE ALIGN="CENTER" BGCOLOR="Silver">
19     <TR>
20       <TD>
21         <PRE>
22 Department ID:
23           <INPUT TYPE="text" NAME="DEPT_ID" SIZE="10" MAXLENGTH="10"><BR>Description:
24           <INPUT TYPE="text" NAME="DEPT_DESC" SIZE="35" MAXLENGTH="35"><BR>Manager    :
25           <SELECT NAME="USR_ID" SIZE="1"><!-- select user from list -->
26             <OPTION VALUE="-->
27             <CFOUTPUT QUERY="USRLIST">
28               <OPTION VALUE="#USR_LIST.USR_ID#">#USR_LNAME#, #USR_FNAME#, #USR_MNAME#
29             </CFOUTPUT>
30           </SELECT></PRE>
31         </TD>
32         <TD>
33           <INPUT TYPE="submit" VALUE="Add Record">
34     </FORM>
35   <FORM ACTION="rc-0.cfm" METHOD="post">
36     <INPUT TYPE="submit" VALUE=" Main Menu ">
37   </FORM>
38   </TD>
39   </TR>
40 </TABLE>
41 </BODY>
42 </HTML>

```

The rc-5a.cfm script produces a data entry form to enable the end user to enter the new department data. To show you how the form works, let's add a new Transportation department (TRANS) and assign a manager to run the department. The rc-5a.cfm script is designed to perform a data validation check in the USR_ID field, using a query and a select box. To see how the script accomplishes those tasks, let's look at some key lines, as follows:

- Lines 4–9 execute a nested query to find all user IDs for employees who are not already department managers. Performing this portion of the data validation procedure ensures that there are no duplicate user IDs in the DEPARTMENT table. Therefore, it will not be possible to have a manager manage more than one department. Also, because a department might not yet have a manager assigned to it, the USR_ID might not have a value in it. To perform the requisite checks, start with a nested query, using the *USR_ID > 0* condition. This condition will be true only for those records in which a manager (USR_ID) exists.

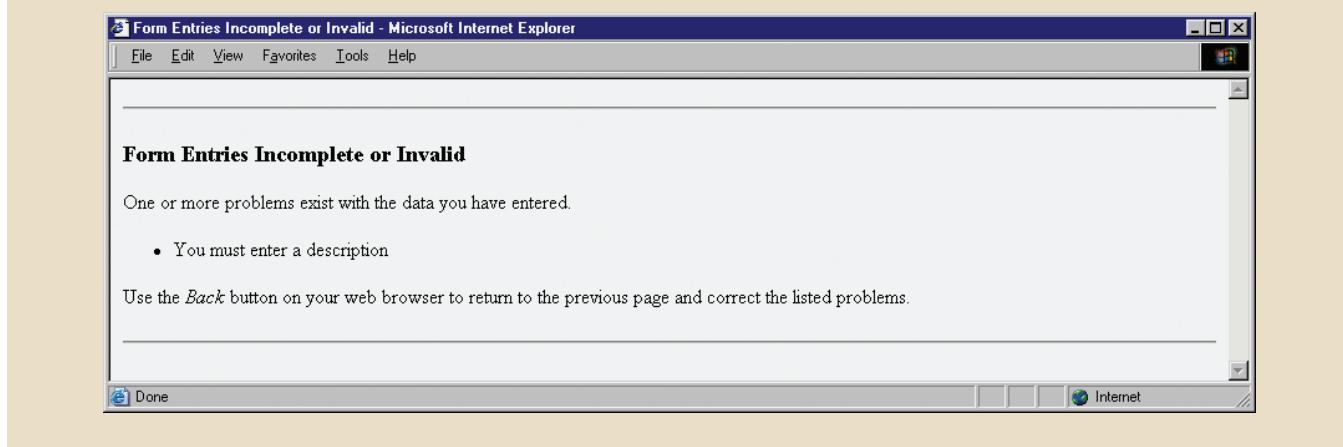
FIGURE J.9 INSERT QUERY—DATA ENTRY SCREEN (RC-5A.CFM)



- Lines 14–34 define the form that will be used to enter the data. Note that the rc-5b.cfm script will be called when the user clicks the Add Record button.
- Lines 16 and 17 are special form tags ColdFusion uses to perform data validation at the server side. In this case, the entries will be validated for the two required fields, the department identification code (DEPT_ID) and the department description (DEPT_DESC). This task is performed by using an INPUT form tag with the following parameters:
 - *TYPE = "hidden"*. This parameter ensures that the field will not be displayed on the screen.
 - *NAME = "fieldname_required"*. This parameter specifies the field to be checked, followed by the word *_required*. Other parameter options are:
 - *_integer* to check for integer values only.
 - *_date* to check for valid dates only in the format mm/dd/yy.
 - *_range* to check for a value within a range. The range is given in the *value =* parameter; for example, *value = "MIN=10 MAX=20"*.
 - *VALUE = "error message"*. This parameter contains the error message to be displayed when the constraint is violated (in this case, when the field is empty). When the parameter *type = _range*, this field contains the maximum and minimum values used in the validation.
- Lines 25–30 create a drop-down select box to show all of the *available* users who can be selected as department manager. Note in particular the following lines:
 - Line 26 defines a null option, represented on the screen by a dotted line to indicate that the department does not have a manager assigned. This line implements the optionality of the USR_ID field. If this line is not included, there will be no way to assign a null to the USR_ID field.
 - Lines 27–29 generate a list of all users who are eligible to manage the new department. Remember that the USRLIST query returns only the USR_ID of users who are not already in the DEPARTMENT table. In other words, the USRLIST query lists only those users *who are still available to become department managers*. (Remember that the business rule specifies that a manager can manage only one department.)

When the user clicks the Add Record button, the form is sent to the web server for processing. There, ColdFusion will evaluate the required fields, sending an error message if one of the required fields is empty. (See Figure J.10.)

FIGURE J.10 SERVER-SIDE VALIDATION ERROR MESSAGE



If the server-side data validation yields the conclusion that the data entry is valid, the second script, rc-5b.cfm, is executed. This script receives the form fields from the rc-5a.cfm script and uses the CFININSERT tag to add the record to the database. Once the record has been added, the rc-5b.cfm script presents a confirmation screen to the end user.

Script rc-5b.cfm (Script J.5B) is shown next. The execution of the rc-5b.cfm script is shown in Figure J.11.

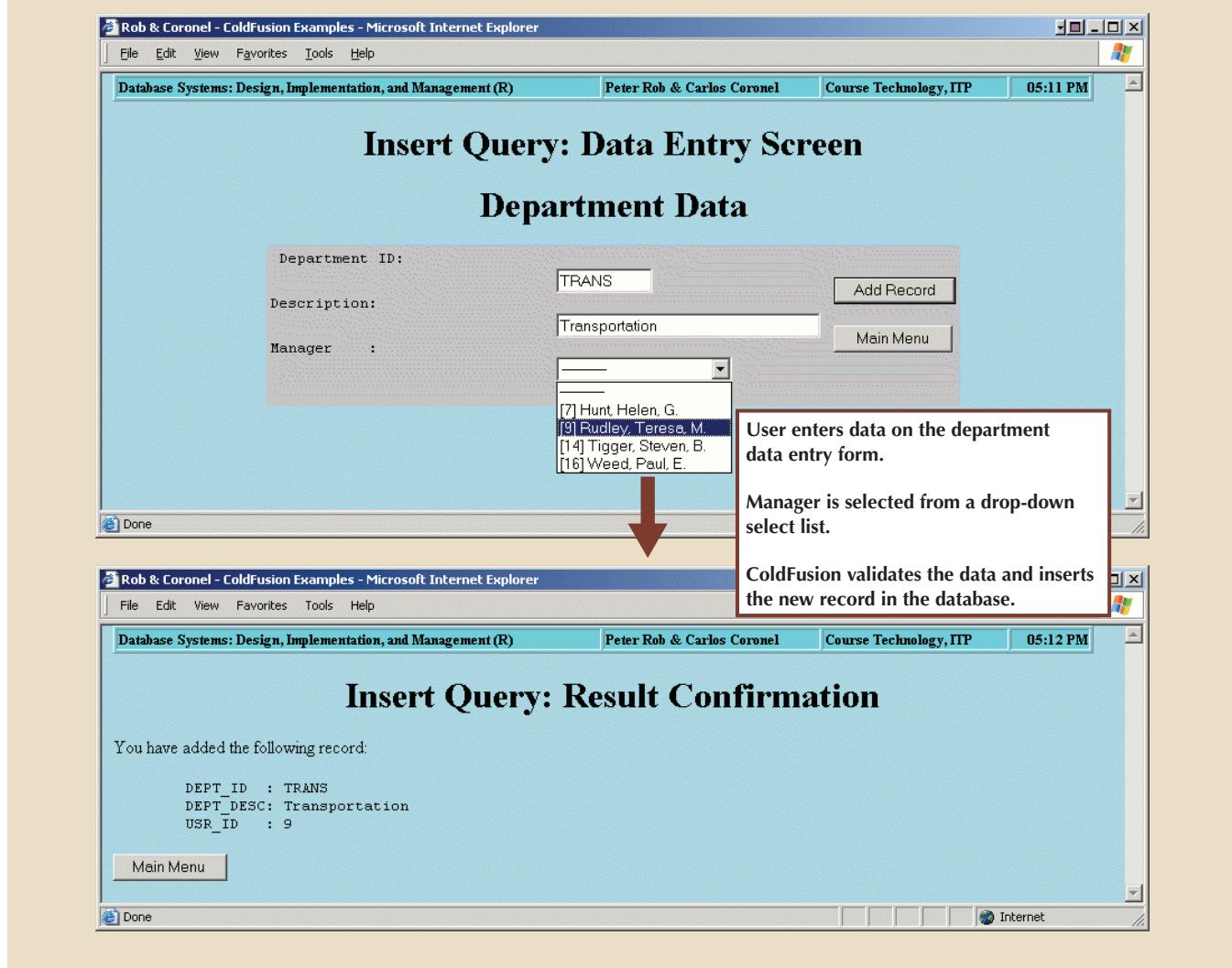
SCRIPT J.5B INSERT QUERY—RESULT CONFIRMATION (RC-5B.CFM)

```

1  <HTML>
2  <HEAD>
3  <TITLE>Rob & Coronel - ColdFusion Examples</TITLE>
4  <!-- inserting record in table --->
5  <CFINSERT Datasource="RobCor" TableName="DEPARTMENT">
6  </HEAD>
7  <BODY BGCOLOR="LIGHTBLUE">
8  <H1>
9  <CENTER><B>Insert Query: Result Confirmation</B></CENTER></H1>
10 <CFOUTPUT>
11   You have added the following record:
12   <PRE>
13     DEPT_ID : #DEPT_ID#
14     DEPT_DESC: #DEPT_DESC#
15     USR_ID : #USR_ID#
16   </PRE>
17   </CFOUTPUT>
18   <FORM ACTION="rc-0.cfm" METHOD="post">
19     <INPUT TYPE="submit" VALUE="Main Menu ">
20   </FORM>
21   </BODY>
22 </HTML>

```

FIGURE J.11 INSERT QUERY—RESULT CONRMATION SCREEN



To see how the rc-5b.cfm script uses the CFINSERT tag to add the record to the database, check line 5. Note that this tag uses the following two parameters:

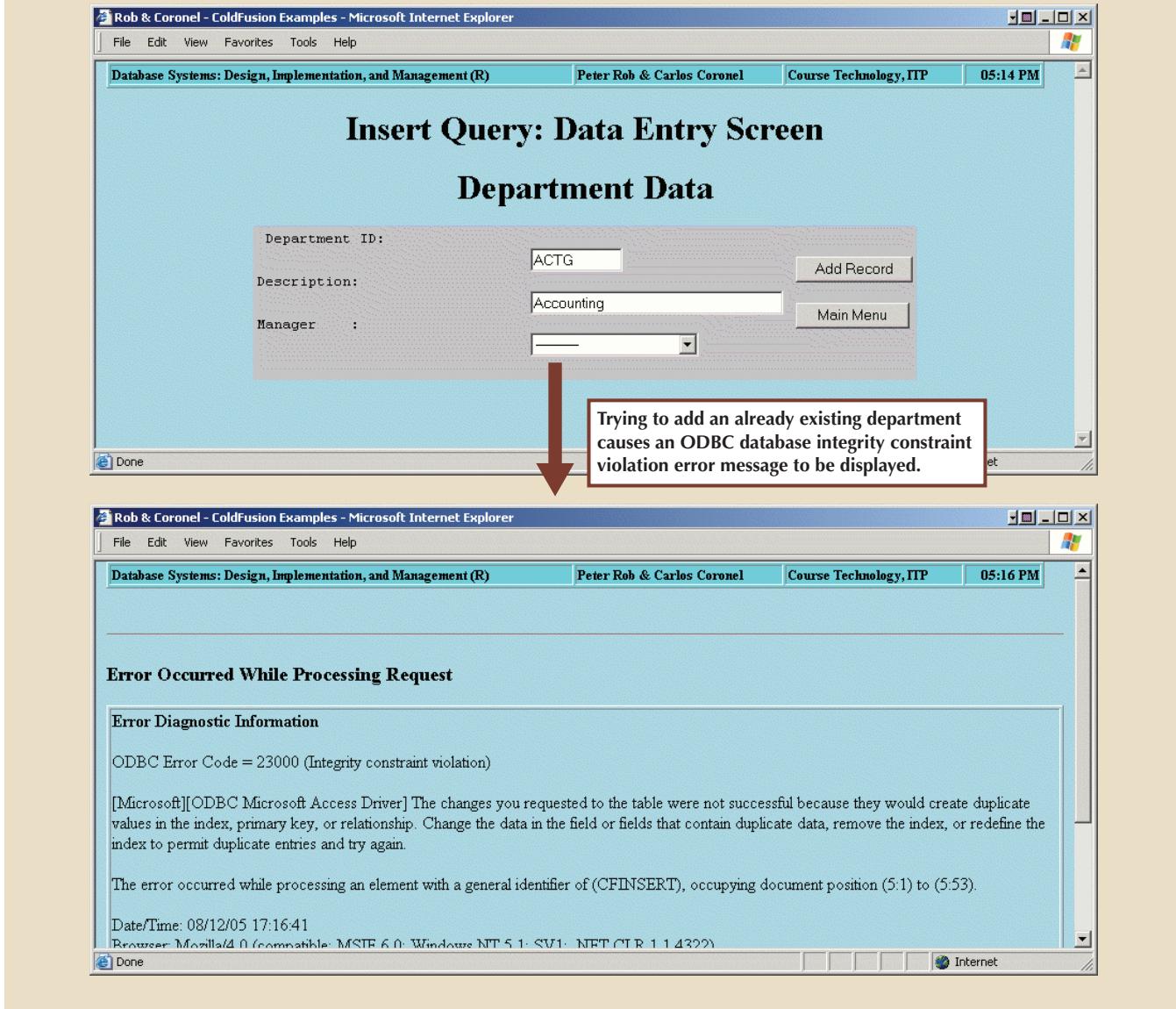
- **DATASOURCE** = “*datasource_name*” (the name of the ODBC database connection).
- **TABLENAME** = “*table_name*” (the name of the table to be updated).

How does the CFINSERT tag know what fields to insert in the table? And where does it get the values to insert into the table columns? The answer to both questions is found by examining the rc-5b.cfm script CFINSERT tag. That tag uses the field names that were defined on the form generated by the rc-5a.cfm script. Recall that the rc-5a.cfm script produced a form containing the DEPT_ID, DEPT_DESC, and USR_ID fields. The CFINSERT tag compares those form field names with the names of the table columns in order to do the insert. To avoid an error condition, the form that was created in the calling page must have form field names that match the table column names.

The MS Access database enforces entity integrity for the Orderdb database’s DEPARTMENT table. Therefore, entering an *existing* department ID on the input form automatically triggers an ODBC database integrity violation error, as shown in Figure J.12.

(Remember that entity integrity is violated when a primary key value is duplicated; your error message may look slightly different depending on your version of ColdFusion.)

FIGURE J.12 ODBC INTEGRITY VIOLATION ERROR



J-1h Updating Data

Data updates require multiple pages. For example, if you want to produce a simple update in the DEPARTMENT table's records, the update process requires *three* different pages.

- The first page, produced by the rc-6a.cfm script, lets the end user select the record to be updated. When the user clicks this page's Edit button, the second page, produced by the rc-6b.cfm script, is called and the first page's search field value is passed to the second page. (To keep the process as simple as possible, the primary key, DEPT_ID, will be used to ensure a unique match. If you use secondary search fields, you may

find more than one record. You would need to use an *additional* page to select one of the multiple records to generate a unique match.)

- The second page (rc-6b.cfm) reads the selected record, then displays a data entry form to let the end user modify the data. When the end user clicks the Update button, this page calls the third script and passes the second page's form fields to the third page.
- The third and last page, generated by the rc-6c.cfm script, updates the data in the database and presents a confirmation message to the end user.

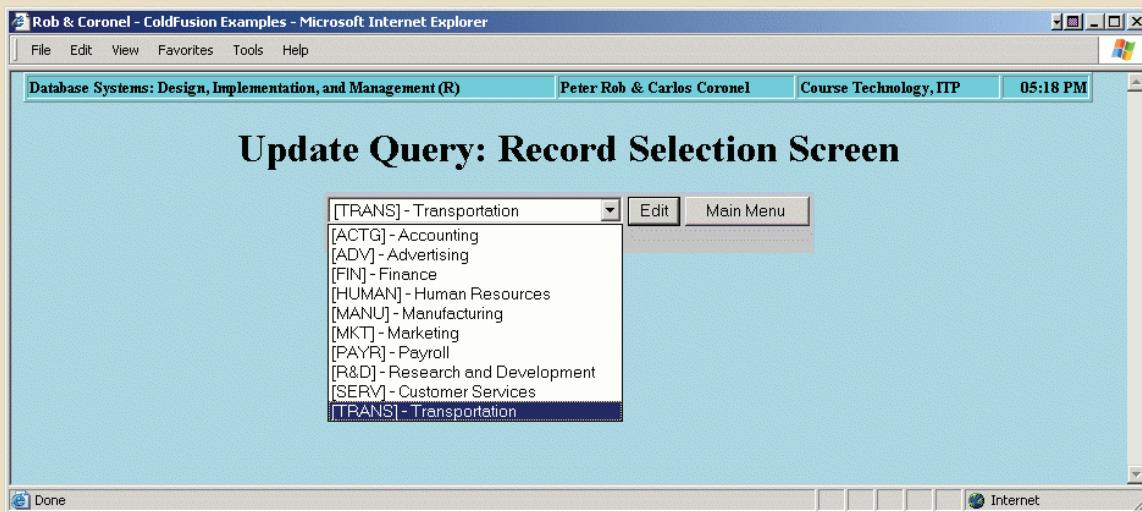
Although this process seems simple enough, several issues must be addressed, as you will see next. Let's begin by taking a close look at Script J.6A (rc-6a.cfm). The rc-6a.cfm script output is shown in Figure J.13.

SCRIPT J.6A UPDATE QUERY—RECORD SELECTION (RC-6A.CFM)

```

1  <HTML>
2  <HEAD>
3  <TITLE>Rob & Coronel - ColdFusion Examples</TITLE>
4  <CFQUERY NAME="Deptlist" DATASOURCE="RobCor">
5      SELECT * FROM DEPARTMENT ORDER BY DEPT_ID
6  </CFQUERY>
7  </HEAD>
8  <BODY BGCOLOR="LIGHTBLUE">
9  <H1>
10 <CENTER><B>Update Query: Record Selection Screen</B></CENTER>
11 </H1>
12 <TABLE ALIGN="CENTER" BGCOLOR="Silver">
13 <TR VALIGN="TOP">
14 <TD>
15 <FORM ACTION="rc-6b.cfm" METHOD="post">
16     <SELECT NAME="DEPT_ID" SIZE=1>
17     <CFOUTPUT QUERY="Deptlist">
18         <OPTION VALUE="#DEPT_ID#" value="#DEPT_ID# - #DEPT_DESC#"
19     </CFOUTPUT>
20     </SELECT>
21     <INPUT TYPE=HIDDEN NAME="DEPT_ID_required" VALUE="DEPT_ID is required">
22 </TD>
23 <TD>
24     <INPUT TYPE="submit" VALUE=" Edit ">
25 </FORM>
26 </TD>
27 <TD>
28     <FORM ACTION="rc-0.cfm" METHOD="post">
29         <INPUT TYPE="submit" VALUE="Main Menu ">
30     </FORM>
31 </TD>
32 </TR>
33 </TABLE>
34 </BODY>
35 </HTML>
```

FIGURE J.13 UPDATE QUERY—RECORD SELECTION SCREEN



The rc-6a.cfm script produces the form that lets the end user select the record to be updated. This record selection process requires the completion of the following steps:

- Lines 4–6 execute a query (“Deptlist”) to retrieve all DEPARTMENT table records. This record set will be used later to create a drop-down selection box.
- Lines 17–19 use a CFOUTPUT tag to produce a list of available options. In this example, ColdFusion uses an OPTION tag for each department in the “Deptlist” query.
- Lines 15–25 produce the record selection form. This form uses an HTML form tag to pick the department to be updated.
- Line 21 defines the “DEPT_ID” form field to be a required field. This definition ensures that the end user will not generate “variable not defined” errors when the next page is called.

When the end user clicks the form’s Edit button, the rc-6b.cfm script is called. The rc-6b.cfm script will receive the DEPT_ID form field as a parameter, using it to find the matching department table record. It will then prepare and display the data edit form.

The script rc-6b.cfm is shown next in Script J.6B. The rc-6b.cfm script output is shown in Figure J.14.

SCRIPT J.6B UPDATE QUERY—EDIT RECORD (RC-6B.CFM)

```

1  <HTML>
2  <HEAD>
3  <TITLE>Rob & Coronel - ColdFusion Examples</TITLE>
4  <CFQUERY NAME="DeptData" DATASOURCE="RobCor">
5      SELECT * FROM DEPARTMENT WHERE (DEPARTMENT.DEPT_ID='#form.DEPT_ID#')
6  </CFQUERY>
7  <CFQUERY NAME="USRLIST" DATASOURCE="RobCor">
8      SELECT USR_ID, USR_LNAME, USR_FNAME, USR_MNAME
9          FROM USER
10         WHERE USR_ID NOT IN (SELECT USR_ID FROM DEPARTMENT
11                                WHERE USR_ID > 0 AND DEPT_ID <> '#form.DEPT_ID#')
12        ORDER BY USR_LNAME, USR_FNAME, USR_MNAME
13  </CFQUERY>
14  </HEAD>
15  <BODY BGCOLOR="LIGHTBLUE">
16  <H1>
17  <CENTER><B>Update Query: Edit Record Screen</B></CENTER>
18  </H1>
19  <FORM ACTION="rc-6c.cfm" METHOD="post">
20  <TABLE ALIGN="CENTER" BGCOLOR="Silver" BORDERCOLOR="Blue">
21  <TR>
22  <TD>
23  <CFOUTPUT QUERY="DeptData">
24  <PRE>
25  <INPUT TYPE="hidden" NAME="DEPT_ID" VALUE="#DEPTDATA.DEPT_ID#">
26  Department ID: <B>#DEPT_ID#</B><BR>
27  Description : <INPUT TYPE="text" NAME="DEPT_DESC" VALUE="#DEPT_DESC#" SIZE=35 MAXLENGTH=35><BR>
28  Manager : <SELECT NAME="USR_ID" SIZE=1><!-- select user from list -->
29          <OPTION <CFIF #DEPTDATA.USR_ID# EQ "">SELECTED</CFIF> VALUE="">-----</OPTION>
30  </CFOUTPUT>
31  <CFOUTPUT QUERY="USRLIST">
32          <OPTION <CFIF #DEPTDATA.USR_ID# EQ #USRLIST.USR_ID#>SELECTED</CFIF> VALUE="#USRLIST.USR_ID#">
33          [#USR_ID#] #USR_LNAME#, #USR_FNAME#, #USR_MNAME#
34  </CFOUTPUT>
35          </SELECT>
36  </PRE>
37  </TD>
38  <TD VALIGN="TOP">
39  <INPUT TYPE="submit" VALUE=" Update ">
40  </FORM>
41  <FORM ACTION="rc-0.cfm" METHOD="post">
42      <INPUT TYPE="submit" VALUE="Main Menu">
43  </FORM>
44  </TD>
45  </TR>
46  </TABLE>
47  </BODY>
48  </HTML>

```

FIGURE J.14 UPDATE QUERY—EDIT RECORD SCREEN

Update Query: Edit Record Screen

Department ID: TRANS

Description : Transportation

Manager : [9] Rudley, Teresa, M.

[7] Hunt Helen, G.
 [9] Rudley, Teresa, M.
 [14] Tigger, Steven, B.
 [16] Weed, Paul, E.

Update Main Menu

This form enables the end user to assign a new manager to the transportation department.

Update Query: Edit Record Screen

Department ID: TRANS

Description : Transportation

Manager : [9] Rudley, Teresa, M.

[7] Hunt Helen, G.
 [9] Rudley, Teresa, M.
 [14] Tigger, Steven, B.
 [16] Weed, Paul, E.

Update Main Menu

Note that the existing manager appears as the default selection.

As you compare the rc-6b.cfm script and the sequence shown in Figure J.14, note that the script generates the following actions:

- Lines 4–6 read the Department data, using the “#form.DEPT_ID#” parameter received from the rc-6a.cfm script.
- Lines 7–13 specify and execute a key query. When a new department is to be inserted, the “USRLIST” query lists all users who are not already managers of a department. This query applies only to *new* record inserts; it cannot be used to edit an *existing* record.

To see why it is necessary to modify the original USRLIST query, let’s suppose that `USR_ID = 13` is the current ACTG department’s manager. If you try to *edit* the ACTG department record using the *original* USRLIST query, the list of “available users” will yield all users who are *not* already in the DEPARTMENT table. Therefore, because the ACTG department’s current manager *is* listed in the DEPARTMENT table’s `USR_ID` column, the current manager will not appear on the list of users who are available as ACTG department manager. In short, if you try to edit (update) a record using the original USRLIST query, you will be forced to select a manager other than the current one. That is clearly not what’s intended!

To avoid the just-described dilemma, the nested query criteria of the USRLIST query must be modified to exclude the “to be edited” department’s DEPT_ID from the subquery. Therefore, line 11 specifies the nested query criteria to be WHERE USR_ID > 0 AND DEPT_ID <> ‘#form.DEPT_ID#’.

Given that modification, the ACTG department’s current manager (USR_ID = 13) will appear on the list of available user IDs.

- Lines 19–40 produce the user edit form. This form allows the end user to modify only two DEPARTMENT table fields: department description (DEPT_DESC) and department manager (USR_ID).
 - *Because the DEPT_ID is the DEPARTMENT table’s primary key, the end user cannot be allowed to modify its value.* The reason for this restriction is simple. Suppose that the end user edits the ACTG department, that is, the DEPT_ID = ‘ACTG’. If the end user is permitted to change the DEPT_ID from ‘ACTG’ to ‘ACTNG’, the department data update will pass the DEPT_ID = ‘ACTNG’ form field to the update script. Unfortunately, the DEPT_ID = ‘ACTNG’ does not exist in the DEPARTMENT table. Therefore, the database will return an error to indicate that the end user is trying to update a record that does not exist—which is true: ‘ACTG’ exists, but ‘ACTNG’ does not.
- Line 25 creates an input form variable named DEPT_ID, to which the “#DEPTDATA.DEPT_ID#” value is assigned. In other words, the script assigns the current record value to the edit mode. Note that this variable is hidden, so it will not be shown on the screen. This value assignment ensures that the DEPT_ID is passed to the rc-6c.cfm script. (Remember that all form variables that are defined with an INPUT or SELECT form tag are passed to the called program.)
- Line 26 ensures that the current department ID is shown on the screen. Note that the end user cannot edit this value.
- Line 27 allows the end user to modify the department’s description. Note that the INPUT tag sets the default value for this field to “#DEPT_DESC#”, thus ensuring that the previous field’s contents are displayed. The end user can then modify the displayed values.
- Lines 28–35 allow the end user to choose a manager for the selected department. These lines create a select box that lists all valid options for the manager field. The valid options are:
 - All users who are not managers.
 - If the department has a manager, the existing manager.
 - A null manager option to indicate that no manager has yet been assigned to the selected department.
 Given those options, the end user can set the manager to null, leave the current manager unchanged, or choose another manager. If the department being edited already has a manager, that manager must appear as the default (SELECTED) option.
- Line 29 allows the manager ID to be set to null. (Note that VALUE = ““.) This line also contains a CFIF tag to evaluate the current value of the department’s USR_ID field. If the USR_ID is ““ (that is, the selected department does not have a manager), this null will be the SELECTED option. Otherwise, this option will appear on the list of options but will not be preselected. This null option must be available to ensure that a department’s manager can be removed.
- Lines 31–34 use the CFOUTPUT tag to create OPTION entries for each of the user IDs in the “USRLIST” query. (Remember that the CFOUTPUT tag will loop through

each record in the named query.) For each added option, a CFIF tag compares the existing USR_ID in the department table (#DEPTDATA.USR_ID#) with the USR_ID being added (#USRLIST.USR_ID#). If the two are equal, the “SELECTED” keyword is added to the OPTION tag.

When the user clicks the form’s Update button, Script J.6B (rc-6b.cfm) calls Script J.6C (rc-6c.cfm), passing its variable values.

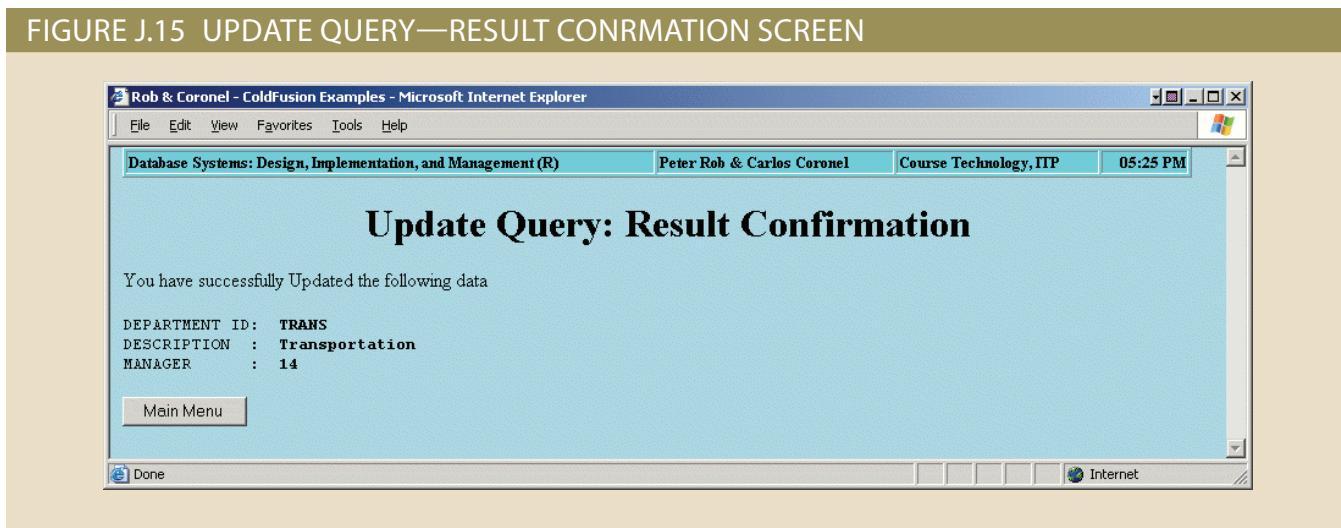
The rc-6c.cfm script’s output is shown in Figure J.15.

SCRIPT J.6C UPDATE QUERY—RESULT CONRMATION (RC-6C.CFM)

```

1  <HTML>
2  <HEAD>
3  <TITLE>Rob & Coronel - ColdFusion Examples</TITLE>
4  <CFUPDATE DATASOURCE="RobCor" TABLENAME="Department">
5  </HEAD>
6  <BODY BGCOLOR="LIGHTBLUE">
7  <H1>
8  <CENTER><B>Update Query: Result Confirmation</B></CENTER>
9  </H1>
10 <CFOUTPUT>
11 You have successfully Updated the following data
12 <PRE>
13 DEPARTMENT ID: <B>#DEPT_ID#</B>
14 DESCRIPTION : <B>#DEPT_DESC#</B>
15 MANAGER : <B>#USR_ID#</B>
16 </PRE>
17 </CFOUTPUT>
18 <FORM ACTION="rc-0.cfm" METHOD="post">
19   <INPUT TYPE="submit" VALUE="Main Menu ">
20 </FORM>
21 </BODY>
22 </HTML>

```



The rc-6c.cfm script uses the CFUPDATE tag to update the DEPARTMENT table. The parameters for this tag are:

- *DATASOURCE = “datasource_name”* (the name of the ODBC database connection).
- *TABLENAME = “table_name”* (the name of the table to be updated).

The CFUPDATE tag uses the form fields passed from the calling page (DEPT_ID, DEPT_DESC, and USR_ID) to update the named table. As was true with the other .cfm

pages, the form created in the calling page must name its form fields to match the table column names. Failure to adhere to that naming requirement will generate a “variable not found” error.

The most basic web-based data management process requires at least three actions: create a new record, modify an existing record, and delete a record. The first two have been discussed, so it’s time to examine the “delete” action.

J-1i Deleting Data

The “delete” query examined in this section enables the end user to delete a department record. As was true for the update query, the delete query’s implementation requires three pages.

- The first page (the rc-7a.cfm script shown in Script J.7A) allows the end user to select the record to be deleted. When the user clicks the form’s Delete button, the rc-7b.cfm script is invoked and the DEPT_ID form field value is passed to it.
- The second page (the rc-7b.cfm script shown in Script J.7B) reads the selected record and displays its data on the screen. This query also performs a referential integrity check to ensure that the end user cannot delete a department that still contains users. When the user clicks the Delete button, this page calls the third page, passing the DEPT_ID form’s field value to that page.
- The last page (the rc-7c.cfm script) deletes the department row from the database table, using the DEPT_ID form field value passed from its calling program, rc-7b.cfm.

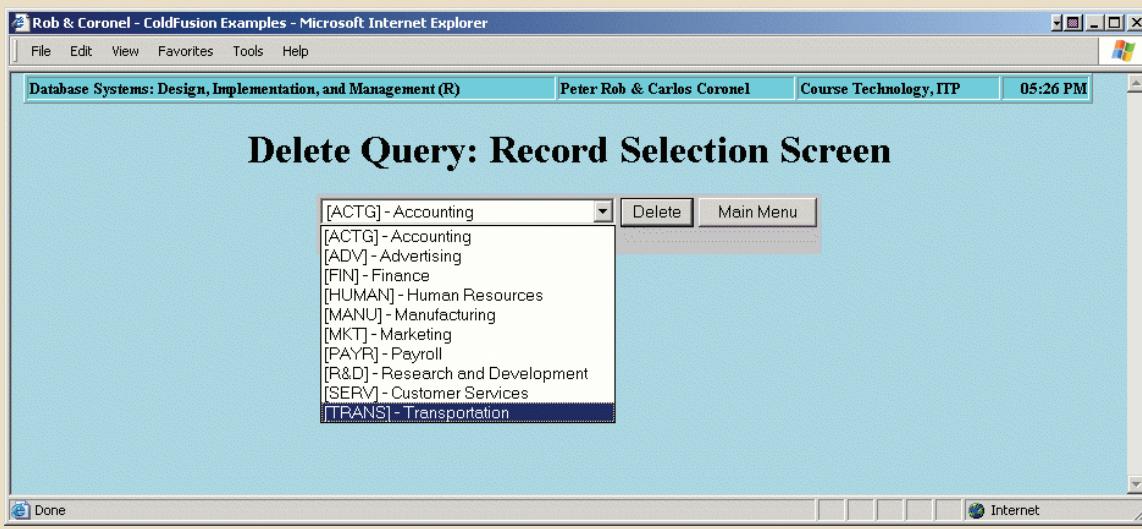
Let’s look at the first of these three scripts. The rc-7a.cfm script output is shown in Figure J.16.

SCRIPT J.7A DELETE QUERY—RECORD SELECTION (RC-7A.CFM)

```

1  <HTML>
2  <HEAD>
3  <TITLE>Rob & Coronel - ColdFusion Examples</TITLE>
4  <CFQUERY NAME="Deptlist" DATASOURCE="RobCor">
5      SELECT * FROM DEPARTMENT ORDER BY DEPT_ID
6  </CFQUERY>
7  </HEAD>
8  <BODY BGCOLOR="LIGHTBLUE">
9  <H1>
10 <CENTER><B>Delete Query: Record Selection Screen</B></CENTER>
11 </H1>
12 <TABLE ALIGN="CENTER" BGCOLOR="Silver">
13 <TR VALIGN="TOP">
14 <TD>
15 <FORM ACTION="rc-7b.cfm" METHOD="post">
16     <SELECT NAME="DEPT_ID" SIZE=1>
17         <CFOUTPUT QUERY="Deptlist">
18             <OPTION VALUE="#DEPT_ID#">[#DEPT_ID#] - #DEPT_DESC#
19         </CFOUTPUT>
20     </SELECT>
21     <INPUT TYPE=HIDDEN NAME="DEPT_ID_required" VALUE="DEPT_ID is required">
22 </TD>
23 <TD>
24     <INPUT TYPE="submit" VALUE="Delete">
25 </FORM>
26 </TD>
27 <TD>
28 <FORM ACTION="rc-7c.cfm" METHOD="post">
29     <INPUT TYPE="submit" VALUE="Main Menu">
30 </FORM>
31 </TD>
32 </TR>
33 </TABLE>
34 </BODY>
35 </HTML>
```

FIGURE J.16 DELETE QUERY—RECORD SELECTION SCREEN



The rc-7a.cfm script allows the end user to select a record to be deleted. To trace its operations, let's examine the following lines:

- Lines 4–6 perform a query (“Deptlist”) to retrieve all records from the DEPARTMENT table. This query result set will be used to display a record selection form.
- Lines 15–25 define the record selection form. When the user clicks the form’s Delete button, the rc-7a.cfm script calls script rc-7b.cfm. As before, the rc-7a.cfm script form passes its DEPT_ID form field to the rc-7b.cfm script.
- Lines 16–20 create the SELECT form control.
- Lines 17–19 use a CFOUTPUT tag to dynamically create the OPTION list.
- Line 21 uses the INPUT tag to define the DEPT_ID field as a required field. ColdFusion uses this command to perform server-side validation on this field. If this field is left blank, ColdFusion will return the error message text entered in the “VALUE” parameter. Therefore, line 21 ensures that the end user selects a record before trying to delete it. (You can’t delete a record without first specifying which one it is.) If the end user fails to select a record before clicking the Delete button, an ODBC database error message will result, indicating the attempted deletion of a nonexistent record—or, even worse, the deletion of all table rows!

The second script (rc-7b.cfm) performs the following two important functions:

- It reads the record to be deleted and presents its data on the screen to let the end user confirm that this is the record (s)he wants to delete.
- It performs the referential integrity validation. Remember that the DEPARTMENT and USER tables maintain a 1:M relationship expressed by “each department may have one or more users.” Therefore, the end user cannot be allowed to delete a department if it still contains users.

Script rc-7b.cfm is shown in Script J7.B. The rc-7b.cfm script output is shown in Figure J.17.

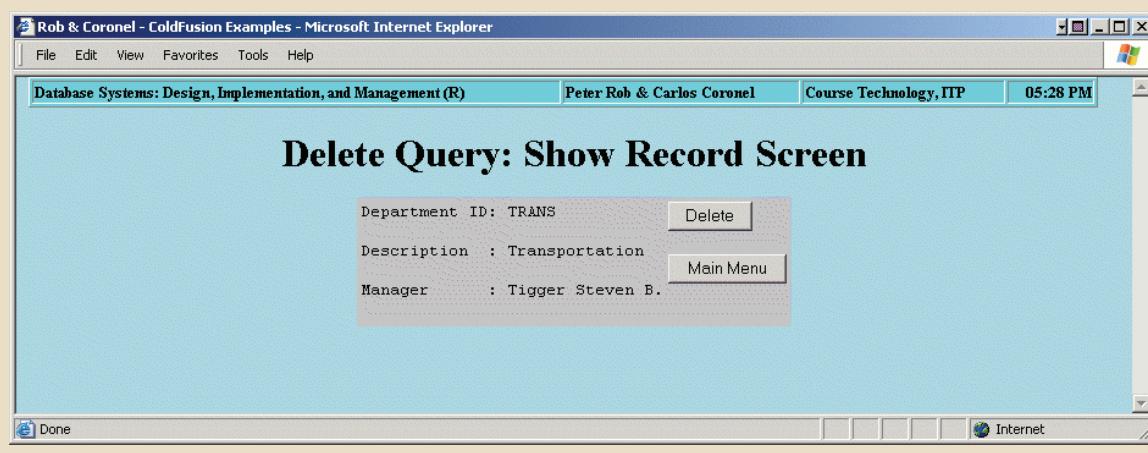
SCRIPT J.7B DELETE QUERY—SHOW RECORD (RC-7B.CFM)

```

1  <HTML>
2  <HEAD>
3  <TITLE>Rob & Coronel - ColdFusion Examples</TITLE>
4  </HEAD>
5  <CFQUERY NAME="DeptData" DATASOURCE="RobCor">
6      SELECT * FROM DEPARTMENT WHERE (DEPARTMENT.DEPT_ID='#form.DEPT_ID')
7  </CFQUERY>
8  <CFIF #DEPTDATA.USR_ID# IS NOT "">
9      <CFQUERY NAME="Usrdata" DATASOURCE="RobCor">
10         SELECT USR_ID, USR_LNAME, USR_FNAME, USR_MNAME FROM USER
11         WHERE (USER.USR_ID = #deptdata.usr_id#)
12     </CFQUERY>
13 </CFIF>
14 <CFQUERY NAME="UsrTot" DATASOURCE="RobCor">
15     SELECT COUNT(*) AS T1 FROM USER
16     WHERE (USER.DEPT_ID = '#form.DEPT_ID')
17 </CFQUERY>
18 </HEAD>
19 <BODY BGCOLOR="LIGHTBLUE">
20 <H1><CENTER><B>Delete Query: Show Record Screen</B></CENTER></H1>
21 <FORM ACTION="rc-7c.cfm" METHOD="post">
22 <CFOUTPUT QUERY="DeptData">
23 <INPUT TYPE="hidden" NAME="DEPT_ID" VALUE="#deptdata.DEPT_ID#">
24 <INPUT TYPE="hidden" NAME="DEPT_DESC" VALUE="#deptdata.DEPT_DESC#">
25 <INPUT TYPE="hidden" NAME="USR_ID" VALUE="#DEPTDATA.USR_ID#">
26 <TABLE ALIGN="CENTER" BGCOLOR="Silver" BORDERCOLOR="Blue">
27 <TR>
28 <TD>
29 <PRE>
30 Department ID: #DEPT_ID#<BR>
31 Description : #DEPT_DESC#<BR>
32 Manager : <CFIF #DEPTDATA.USR_ID# IS NOT "">#Usrdata.USR_LNAME# #Usrdata.USR_FNAME# #Usrdata.USR_MNAME#</CFIF>
33 </CFOUTPUT>
34 </PRE>
35 </TD>
36 <TD VALIGN="TOP">
37 <CFIF #USRTOT.T1# EQ 0>
38     <INPUT TYPE="submit" VALUE=" Delete ">
39 <CFELSE>
40     <SMALL><B>We cannot delete this record <BR>because there are dependent <BR>users assigned to this department</B></SMALL>
41 </CFIF>
42 </FORM>
43 <FORM ACTION="rc-0.cfm" METHOD="post"><INPUT TYPE="submit" VALUE="Main Menu"></FORM>
44 </TD>
45 </TR>
46 </TABLE>
47 </BODY>
48 </HTML>

```

FIGURE J.17 DELETE QUERY—SHOW RECORD SCREEN



Let's examine the rc-7b.cfm script to understand how it works.

- Lines 5–7 use the CFQUERY tag to read the selected department data. The query uses the “#form.DEPT_ID#” form field passed from the rc-7a.cfm script.
- Lines 8–13 retrieve the department manager data from the USER table. Because this is an optional field, the user ID is checked first to see whether it is not null. If this non-null condition is met, the user data is read from the USER table, using the “#deptdata.usr_id#” value. If the user ID is null, there is no need to read the user data.
- Lines 14–17 perform referential integrity validation checks. The process starts by executing a query to see if users are still assigned to the department to be deleted. Note that the SQL query in lines 15 and 16 counts the number of users assigned to the department. (If this count yields a value greater than zero, the department contains at least one user.) Note that the count is stored in variable T1.
- Lines 21–42 define a form to display the department data and to confirm the record deletion. When the user clicks the Delete button, the rc-7c.cfm script (shown in Script J.7C) is called and the three variables (DEPT_ID, DEPT_DESC, and USR_ID) are passed to it.
- Line 23 defines the form’s DEPT_ID field as “hidden,” and the to-be-deleted DEPARTMENT table’s DEPT_ID value is assigned to this hidden field. (Although this hidden input field does not show on the screen, it is passed to the next script.)
- Lines 24 and 25 perform the same function as line 23, defining the remaining form fields (DEPT_DESC and USR_ID) as “hidden” and assigning the corresponding department field values to the hidden form fields. These hidden form field values also are passed to the next script.
- Lines 30–32 display the department data for the record to be deleted. This action enables the end user to see the record to confirm that this is, in fact, the department record (s)he wants to delete. Note that the fields specified in lines 30 and 31 use the “DeptData” query source as specified in line 22’s CFOUTPUT tag. In contrast, note that line 32’s field name prefix indicates that it uses fields from the “Usrdata” query.
- Line 32 uses a CFIF tag to check whether there are user data. If the “#deptdata.usr_id#” is not null, the user data are displayed.
- Lines 37–41 check to see if any users are assigned to the department. If user records do not exist (#usrtot.t1# EQ 0), the Delete button is shown. (Check line 15 again and note that its count is now the basis for the condition check.) If the count is anything other than zero, the form displays a message to indicate that users are still assigned to this department and the Delete button is not shown.

If the record can be deleted and the user clicks the Delete button, script rc-7c.cfm (Script J.7C) is called and the (hidden) form fields are passed to it. The rc-7c.cfm script output is shown in Figure J.18.

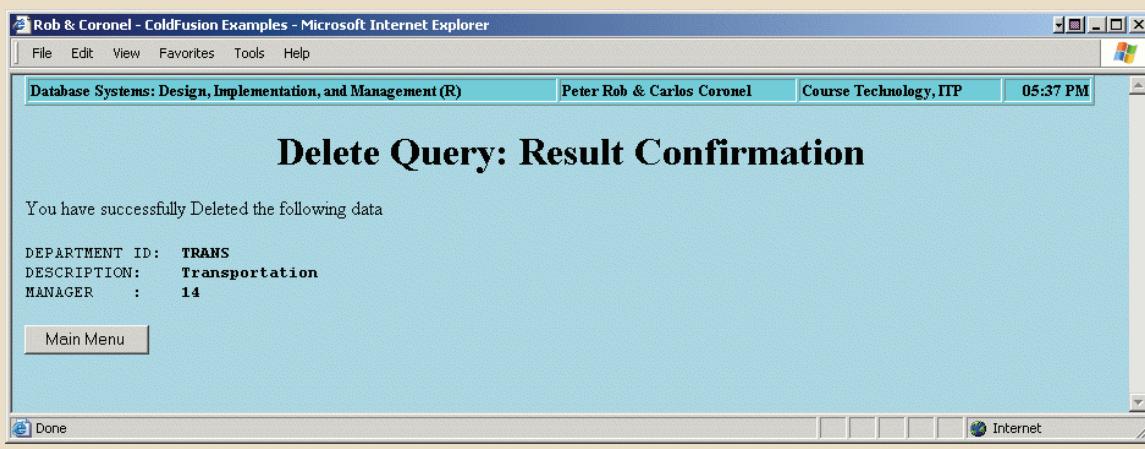
SCRIPT J.7C DELETE QUERY—RESULT CONRMATION (RC-7C.CFM)

```

1 <HTML>
2 <HEAD>
3 <TITLE>Rob & Coronel - ColdFusion Examples</TITLE>
4 <CFQUERY NAME="DeleteDept" DATASOURCE="RobCor">
5     DELETE FROM DEPARTMENT WHERE (DEPT_ID = '#FORM.DEPT_ID#')
6 </CFQUERY>
7 </HEAD>
8 <BODY BGCOLOR="LIGHTBLUE">
9 <H1>
10 <CENTER><B>Delete Query: Result Confirmation</B></CENTER>
11 </H1>
12 <CFOUTPUT>
13 You have successfully Deleted the following data
14 <PRE>
15 DEPARTMENT ID: <B>#DEPT_ID#</B>
16 DESCRIPTION: <B>#DEPT_DESC#</B>
17 MANAGER : <B>#USR_ID#</B>
18 </PRE>
19 </CFOUTPUT>
20 <FORM ACTION="rc-0.cfm" METHOD="post">
21     <INPUT TYPE="submit" VALUE="Main Menu ">
22 </FORM>
23 </BODY>
24 </HTML>

```

FIGURE J.18 DELETE QUERY—RESULT CONRMATION SCREEN

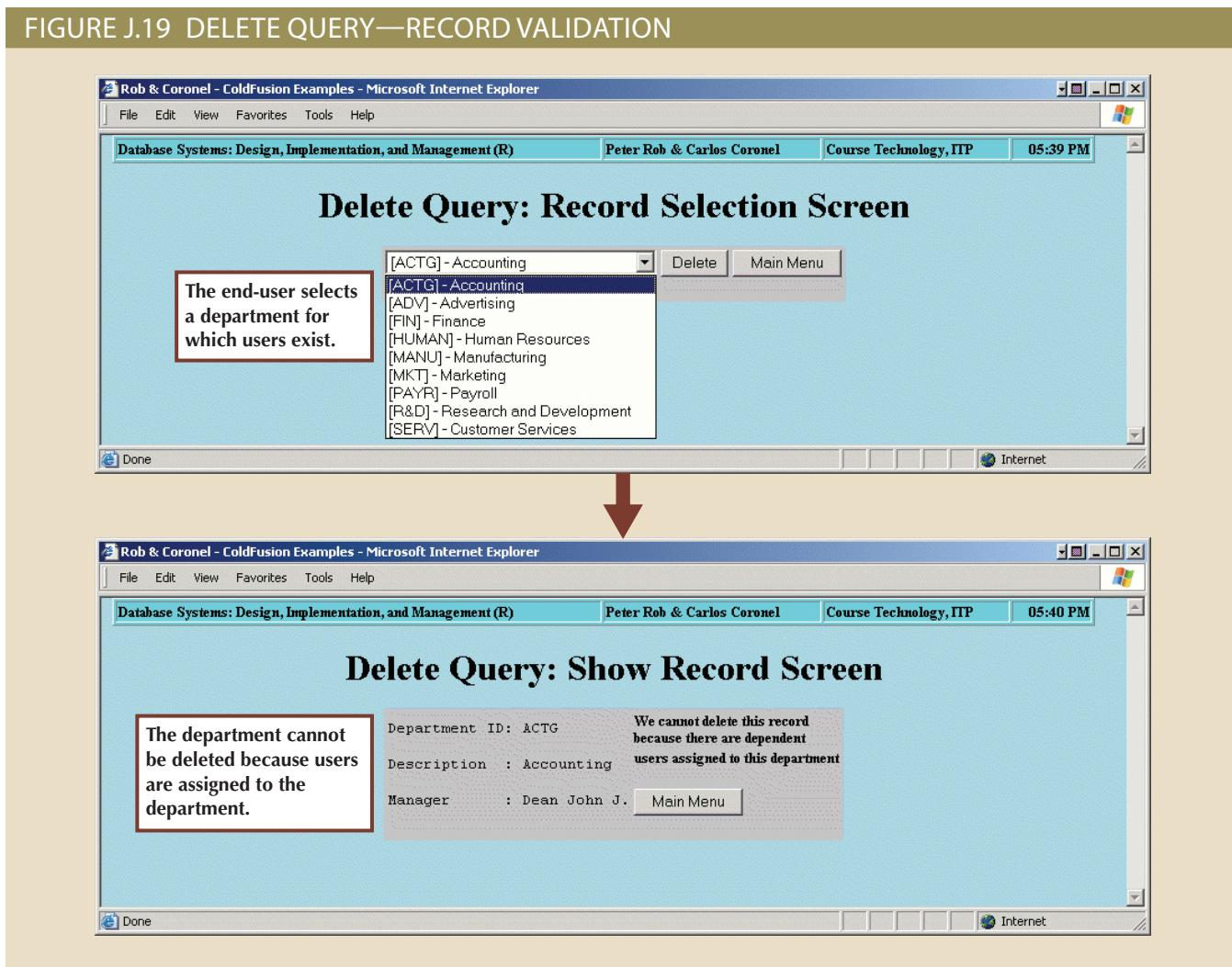


The rc-7c.cfm script deletes the department record from the database and displays a confirmation screen. To see how the script accomplishes those tasks, examine the following lines:

- Lines 4–6 perform a query used to delete the department record from the database. This query executes the SQL “Delete” statement, using the form’s DEPT_ID field received from the rc-7b.cfm script.
- Lines 10–17 confirm that the record has been deleted, and they display the deleted data.

Figure J.19 shows how the delete sequences work. As you examine the screens, note that the attempt failed to delete a department that still contains users. Also note that the Delete button is not shown in the second screen. There is, after all, no reason to display a Delete button if the selected department record cannot be deleted.

FIGURE J.19 DELETE QUERY—RECORD VALIDATION



Note

The ColdFusion techniques presented in this appendix represent just the tip of the proverbial iceberg in the development of database-enabled web applications. At the time of this writing, ColdFusion provides hundreds of additional tags and functions to help you develop professional web applications properly. Although the preceding examples are far from exhaustive, they do provide a compelling illustration of the web interfaces power and flexibility.

In today's increasingly web-driven business environment, there is little doubt that you will work with databases that are web-enabled. Given the clear competitive advantages

provided by database web access, it is tempting to focus on the web side of the web-database equation. Yet it is important to realize that a web interface to a badly designed database is a recipe for database disasters. On the other hand, good database design and implementation, coupled with sound web development techniques, yield countless business tactical and strategic advantages, virtually boundless professional opportunities, and personal satisfaction.

J-2 Internet Database Systems: Special Considerations

Internet database systems involve more than just the development of database-enabled web applications. In addition, certain issues must be addressed when web interfaces are used as the gateway to corporate and institutional databases. For example, data security, transaction management, client-side data validation, and many other operational and management challenges must be met. Although many of those issues were discussed in detail in Chapter 15, Database Connectivity and Web Technologies, they are particularly relevant to web database development. Therefore, some of them are revisited here.

Whether you are talking about databases in a conventional client/server environment or in the latest Internet arena, database systems development requires sound database design and implementation. The database system must exist within a secure environment that is well suited to maintaining well-monitored and -protected data access, robust transaction management with a focus on data integrity maintenance, and solid data recovery. Finally, from the end user's and business manager's points of view, the database is not particularly useful unless its front end is characterized by user-friendly, information-capable, and transaction-supportive end-user applications.

Production database and data warehouse designs are not affected—at the conceptual level—by the change from the conventional client/server environment to the Internet's client/server environment. Therefore, the basic design processes and procedures need not be revisited. However, Internet database application development is quite different from that found in the traditional client/server environment. And given the vastness of the Internet, development issues such as security, backup, and transaction volume are even more critical than they were in the traditional environment.

Clearly, concurrent database access by multiple heterogeneous clients affects how transactions are defined and managed. Support for multiple data sources and types, the advent of increasing platform independence and portability, process distribution and scalability, and open standards have a major effect on how applications are developed, installed, and managed.

No doubt, database application development is most affected by the Internet. Characteristics of the Internet—particularly its web service—fundamentally change the way that applications work. The stateless nature of the web has a major impact on how database queries are presented and executed. Just think of the web's request-reply model and how different it is from the conventional programmer's view.

If database systems are to be developed and managed intelligently, today's database administrator must understand the Internet-based business environment in order to cope successfully with the issues that drive the development, use, and management of web-to-database interfaces. Since it all begins and ends with data, you will begin by looking at the incredibly diverse data types that are supported in the Internet environment.

J-2a What Data Types Are Supported?

Web development requires the concurrent management of many and quite different data types. Typically, conventional databases support data types such as Julian dates, various types of numbers (integer, floating point, and currency), and text (fixed and variable length). Most leading RDBMS vendors support extended data types such as binary and OLE (Object Linking and Embedding) objects in the conventional database environment.

Because interactive websites tend to integrate data from multiple sources—word-processed documents, spreadsheets, presentations, pictures, movies, sounds, and even holograms—some web and database designers use extended data types to store page components (in binary format) for later incorporation into the webpages. Although the page arrangement may provide better data organization from the database point of view, several issues must be addressed.

- How does someone store and extract data objects such as documents, pictures, and movies through a web browser? Remember that the web client expects every page component to be a file stored in the web server's directory. Therefore, the DBMS or the web-to-database middleware must provide special functions or subroutines that allow objects to be extracted dynamically from a database field to the web server's directory, and vice versa.
- How much overhead will be created by the storage of binary objects in the database? How robust must the DBMS be to handle binary object transactions? What are the limitations for extended or OLE data types? How many extended or OLE data type fields can tables have?
- Does the client browser support the data type of the object being accessed? Are the necessary plug-ins available? Is there a way to automatically translate documents from their native format to HTML? For example, a PowerPoint presentation can be viewed within an Internet Explorer browser, but a plug-in might be required to open it in a different browser.
- Finally, storing pictures or multimedia presentations in the database can very quickly increase the size of the database. Does the DBMS support very large databases? What about transaction speed? The concurrent insertion and extraction of binary objects in database fields can take quite a toll on database transaction performance. How many users are going to access the database? How frequently?

Web-to-database interface design must juggle all of those issues and find the right balance to ensure that the database does not become the web-based system's bottleneck.

J-2b Data Security

Security is a key issue when databases are accessible through the Internet. Most DBMS vendors provide interfaces to manage database security. When you create a database web interface, security can be implemented in the web server, in the database, and in the networking infrastructure. In many ways, building multiple firewalls is the essence of Internet database security.

At the web server level, most web clients and servers can perform secure transactions by using encryption routines at the TCP/IP protocol level. Clients and servers can exchange security certificates to ensure that the clients and servers are who they say they

are. Therefore, you must ensure that the clients and servers are properly registered and that they have compatible encryption protocols. Also, the web administrator can use TCP/IP addresses and firewalls to restrict access to the site. The firewalls ensure that only authorized data travel outside the company.

All RDBMS vendors provide security mechanisms at the database end, providing some form of login authentication for users who are trying to access the database. At the SQL level, administrators can use the GRANT and REVOKE commands to assign access restrictions to tables and/or to specific SQL commands.

Web-to-database middleware vendors usually have several security mechanisms available for interfacing with databases. For example, when using ODBC data sources, the administrator can restrict end-user access to certain SQL statements (such as SELECT, UPDATE, INSERT, or DELETE) or to some combination of those commands. And although the webpages operate in the request-reply model, the use of web interfaces does not preclude the creation of algorithms to guarantee data entity and referential integrity requirements. Data security measures must also include logs to relate data manipulation activities to specific end users. Those logs ensure that each database update is directly associated with an authorized user.

Security must also be extended to support electronic commerce, or e-commerce. That support is key to the website's ability to execute secure business transactions over the Internet. If a vendor wants to be able to take credit card orders over the Internet, the order processing, rooted in a production database environment, must have strict security mechanisms in place to safeguard the transactions. In addition, the order transaction must be able to interact *securely* with multiple sites—such as distributors and banks—making sure that the transaction information cannot be modified and that the information cannot be stolen.

J-2c Transaction Management

Although the preceding comments focus on transaction-management issues that must be addressed for e-commerce to be conducted successfully, the concept of database *transactions* is foreign to the web. Remember that the web's request-reply model means that the web client and the web server interact by using very short messages. Those messages are limited to the request for and delivery of pages and their components. (Page components may include pictures, multimedia files, and so on.) The dilemma created by the web's request-reply model is that:

- The web cannot maintain an open line between the client and the database server.
- The mechanics of a recovery from incomplete or corrupted database transactions require that the client must maintain an open communications line with the database server.

Clearly, the creation of mission-critical web applications mandates support for database transaction management capabilities. Given the just-described dilemma, *designers must ensure proper transaction management support at the database server level.*

Many web-to-middleware products provide transaction management support. For example, ColdFusion provides this support through the use of its CFTRANSACTION tag. If the transaction load is very high, this function can be assigned to an independent computer. By using that approach, the web application and database servers are free to perform other tasks and the overall transaction load is distributed among multiple processors.

J-2d Denormalization of Database Tables

When the web is used to interact with databases, the application design must take into account the fact that the web forms cannot use the multiple data entry lines that are typical of parent-child (1:M) relationships. Yet those 1:M relationships are crucial in e-commerce. For example, think of order and order line, or invoice and invoice line. Most end users are familiar with the conventional GUI entry forms that support multitable (parent-child) data entry through a multiple-component structure composed of a main form and a subform. Using such main-form/subform forms, the end user can enter multiple purchases associated with a single invoice. All data entry is done on a single screen.

Unfortunately, the web environment does not support this very common type of data entry screen. As illustrated in the ColdFusion script examples, the web can easily handle single-table data entry. However, when multitable data entries or updates are needed—such as order with order lines, invoice with invoice lines, and reservation with reservation lines—the web falls short. Although implementing the parent/child data entry is not impossible in a web environment, its final outcome is less than optimum, usually counterintuitive, less user-friendly, and prone to errors.

To see how the web developer might deal with the parent/child data entry, let's briefly examine how you might deal with the ORDER and ORDER_LINE relationship used to store customer orders. Using an applications middleware server such as ColdFusion to create a web front end to update orders, one or more of the following techniques might be used:

- Design HTML frames to separate the screen into order header and detail lines. An additional frame would be used to provide status information or menu navigation.
- Use recursive calls to pages to refresh and display the latest items added to an order.
- Create temporary tables or server-side arrays to hold the child table data while in the data entry mode. This technique is usually based on the bottom-up approach in which the end user first selects the products to order. When the ordering sequence is completed, the order-specific data, such as customer ID, shipping information, and credit card details, are entered. Using this technique, the order detail data are stored in the temporary tables or arrays.
- Use stored procedures or triggers to move the data from the temporary table or array to the master tables.

Although the web itself does not support the parent/child data entry directly, it is possible to resort to web programming languages such as Java, JavaScript, or VBScript to create the required web interfaces. The price of that approach is a steeper application development learning curve and a need to hone programming skills. And while that augmentation works, it also means that complete programs are stored outside the HTML code that is used in a website.

Key Terms

ColdFusion Markup
Language (CFML), J-3

script, J-3
stateless system, J-16

tag, J-3
web application server, J-2

Review Questions

1. What are scripts, and how are they created in ColdFusion?
2. Describe the basic services provided by the ColdFusion web application server.
3. Discuss the following assertion: The web is not capable of performing transaction management.
4. Transaction management is critical to the e-commerce environment. Given the assertion made in Question 3, how is transaction management supported?
5. Describe the webpage development problems related to database parent/child relationships.

Problems

In the following exercises, you are required to create ColdFusion scripts. When you create these scripts, include one main script to show the records and the main options, for a total of five scripts for each table (show, search, add, edit, and delete). Consider and document foreign key and business rules when creating your scripts.

1. Create ColdFusion scripts to search, add, edit, and delete records for the USER table in the RobCor data source.
2. Create ColdFusion scripts to search, add, edit, and delete records for the INVTYPE table in the RobCor data source.
3. Create ColdFusion scripts to search, add, edit, and delete records for the VENDOR table in the RobCor data source.
4. Modify the insert scripts (rc-5a.cfm and rc-5b.cfm) for the DEPARTMENT table so that the users who can be manager of a department are only those who belong to that department.
5. Create an Order data-entry screen, using the ORDERS and ORDER_LINE tables in the RobCor data source. To do this, you can use frames and other advanced ColdFusion tags. Consult the online manual and review the demo applications.

Appendix K

The Hierarchical Database Model

Preview

Chapter 2, Data Models, briefly introduced the hierarchical model's history and basic structure. The focus in this appendix is on implementation issues. IBM's Information Management System (IMS) is used to show you how the hierarchical model's basic structures are implemented. Although the hierarchical model now labors on as a mere legacy system, its structure and the implementation issues examined in this appendix remain a valuable part of your database knowledge base. In fact, many hierarchical concepts survive within the modern database environment, thus illustrating that "the more things change, the more they stay the same."

You will discover that, when properly implemented, the hierarchical model creates an environment in which some very important data integrity rules are maintained automatically. If the database design conforms to the hierarchical structure, the hierarchical model yields fast access and is capable of handling large amounts of data.

Data Files and Available Formats

[MS Access](#) [Oracle](#) [MS SQL](#) [My SQL](#)

[MS Access](#) [Oracle](#) [MS SQL](#) [My SQL](#)

There are no data files for this appendix.

Data Files Available on cengagebrain.com

database description (DBD) statement

In the hierarchical model, the series of commands that define the hierarchical tree structure and how the segments are stored in the database.

program communication block (PCB)

In a hierarchical database, after the physical database has been defined through the DBD, a way through which application programs are given a subset of the physical database.

Recall from Chapter 2 that the hierarchical database model is based on a *tree* structure. You will see how each tree structure is stored in its own (physical) database and how each is defined by a detailed **database description (DBD) statement**. After the physical database has been defined through the DBD, you will see how application programs are given a subset of the physical database through a **program communication block (PCB)**. Although the hierarchical model's application programs tend to be less complex than those written for file systems, the complexity of the database-definition process makes the hierarchical model's implementation more difficult.

Before reading about how to implement a hierarchical database model, you should understand its basic concepts and components. To review, augment, and illustrate the hierarchical database discussion presented in Chapter 2, you will examine some of the details of the hierarchical database structure in the next two sections.

K-1 A Simple Billing System

One of the billing system's components is the invoice. A typical invoice form, shown in Figure K.1, shows that a customer named Mary D. Allen purchased three items on 12-Feb-2018. Note that the invoice in Figure K.1 contains:

- Basic customer data, such as the customer number, name, and address. The label CUSTOMER will be used to refer to such data.
- Specific invoice data, such as the invoice number and date. The label INVOICE will be used when referring to such data.
- A variable number of invoice detail lines, one for each product bought. The label INVLINE will be used when referring to the invoice detail-line data.
- Computed (derived) data such as subtotals, taxes, and totals.

FIGURE K.1 AN INVOICE FORM

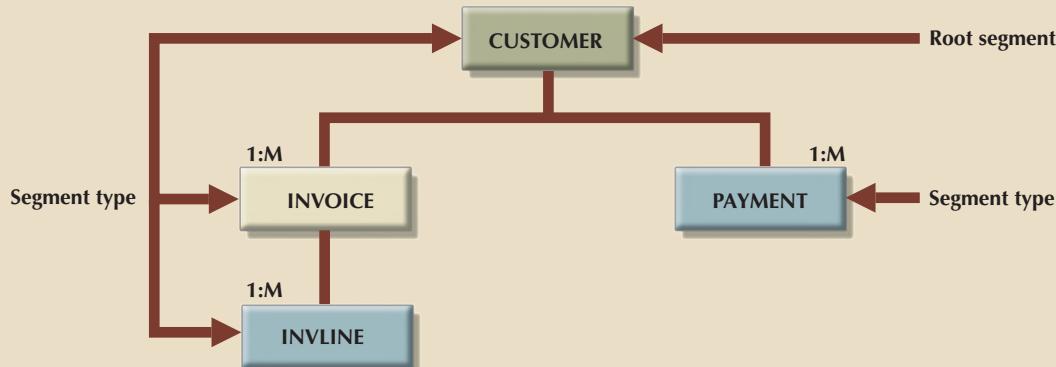
The diagram illustrates the hierarchical structure of an invoice form. It is divided into three main sections: Invoice header, Invoice detail, and Invoice footer. The Invoice header (Fixed number of lines) contains customer information: Customer number: 1276, Customer name: Mary D. Allen, and Customer Address: 23 Main, Anytown, TN 37121. The Invoice detail (Variable number of lines) contains a table of products purchased: Glue gun (1 unit, \$12.95), Drill bit (5 units, \$0.49), and Chisel (2 units, \$8.99). The Invoice footer (Fixed number of lines) contains computed data: Subtotal (\$33.38), Discount (\$1.00), Tax (\$2.45), and Total due (\$34.83).

Product Purchased	Number Purchased	Unit Price	Total
Glue gun	1	\$12.95	\$12.95
Drill bit	5	\$0.49	\$2.45
Chisel	2	\$8.99	\$17.98

Naturally, a billing system contains additional components. Because customers may make purchases on credit, the payments made by customers must be tracked. The label PAYMENT will be used to refer to the customer payment data. To keep the billing system simple, there will be no need to track customer balances at this point.

From a hierarchical point of view, the purchase and payment information introduced thus far can be represented by a hierarchy based on four segment types, CUSTOMER, INVOICE, PAYMENT, and INVLINE, as shown in Figure K.2.

FIGURE K.2 HIERARCHICAL STRUCTURE OF A SAMPLE DATABASE



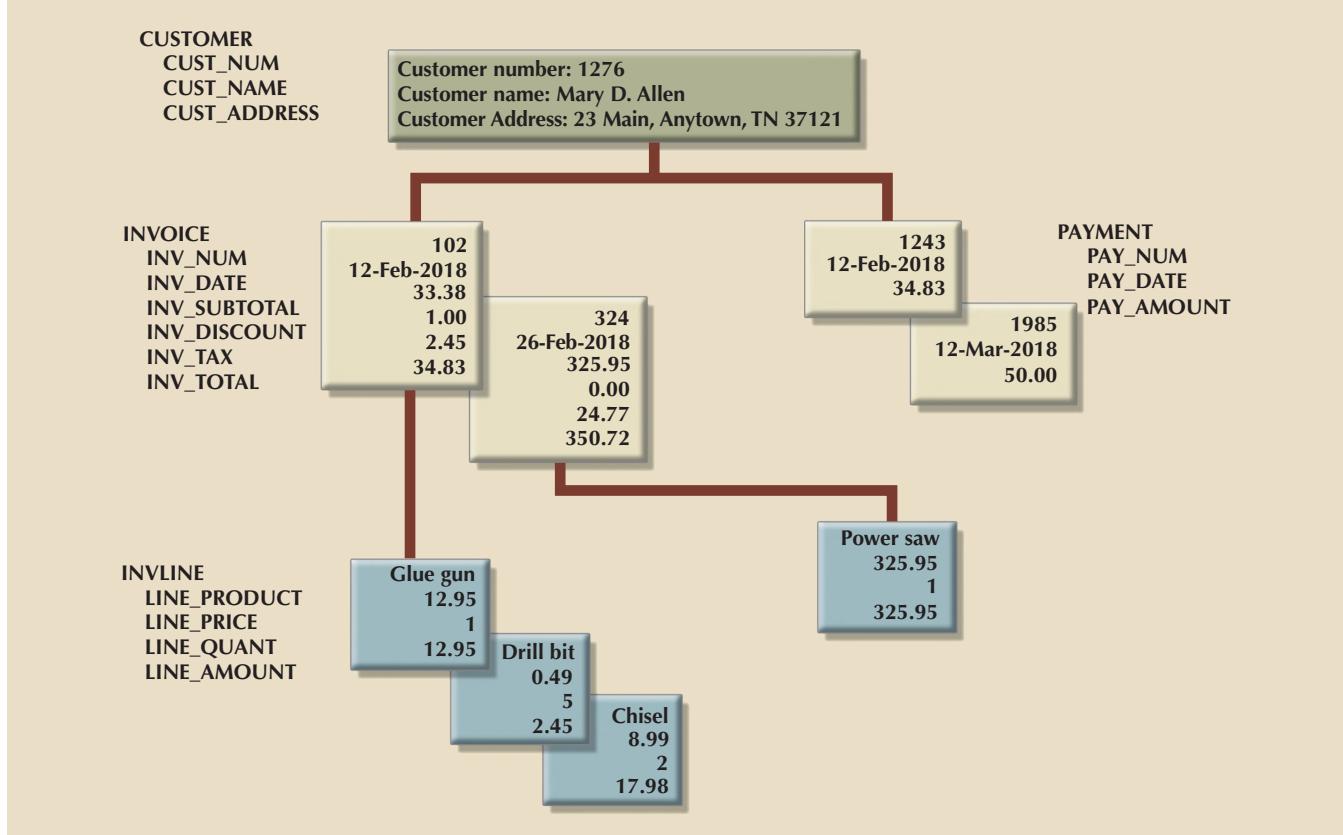
The four segment types or segments you see in Figure K.2 exist within a database named CUSREC. Each *segment type* represents a specific entity set and contains several *segment occurrences*. For example, the CUSTOMER segment type may contain segment occurrences such as Mary D. Allen, John P. Marsutto, and Jean M. Valverde. Given the structures shown in Figure K.2, you can now describe the following relationships:

1. A customer can have one or more invoices and can make one or more payments. However, each payment is made by only one customer, and each invoice belongs to only one customer. In other words, the model depicts a 1:M relationship between CUSTOMER and the two segments INVOICE and PAYMENT. The CUSTOMER segment is the *parent* of the INVOICE and PAYMENT segments.
2. Each INVOICE and each PAYMENT segment occurrence is related to only one CUSTOMER segment occurrence.
3. The INVOICE segment is the parent of the INVLINE segment.
4. Each INVLINE segment occurrence is related to only one INVOICE segment occurrence. Given the conditions in Numbers 3 and 4, you may conclude that a 1:M relationship exists between INVOICE and INVLINE. (Remember, a single INVOICE may contain many items that are entered as *detail lines*.)

Each match of a root segment occurrence with its child segment occurrences represents a hierarchical database *record occurrence*. Figure K.3 illustrates several relationships produced by the root segment occurrence of the customer named Mary D. Allen. As you examine Figure K.3, note that Mary D. Allen's record consists of two INVOICE segment occurrences and two PAYMENT segment occurrences. Invoice number 102 contains the detail lines for the items "Glue gun," "Drill bit," and "Chisel" and is related to Mary D. Allen. The item "Power saw," located in invoice number 324, is also related to Mary D. Allen.

Figure K.3 illustrates that the hierarchical database record is formed by the segment types CUSTOMER, INVOICE, INVLINE, and PAYMENT. The segment components are the equivalent of file fields. In other words, the CUSTOMER segment components CUST_NUMBER, CUST_NAME, and CUST_ADDRESS are equivalent to a file system's CUSTOMER file fields.

FIGURE K.3 A SINGLE OCCURRENCE OF A CUSREC RECORD



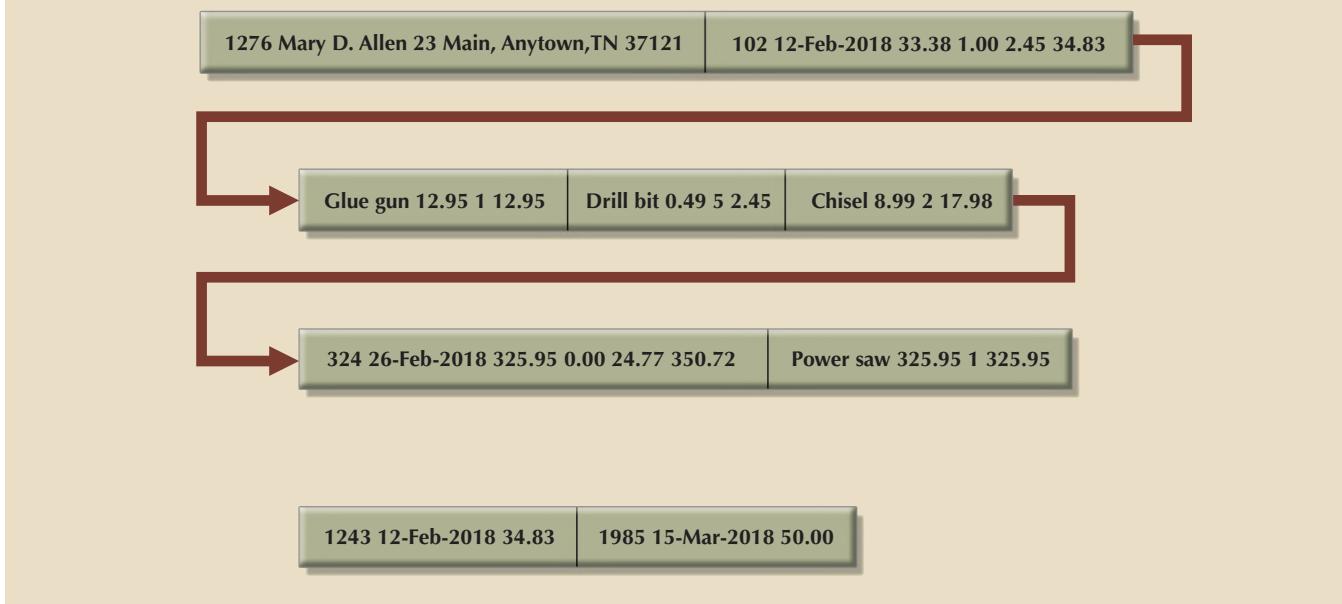
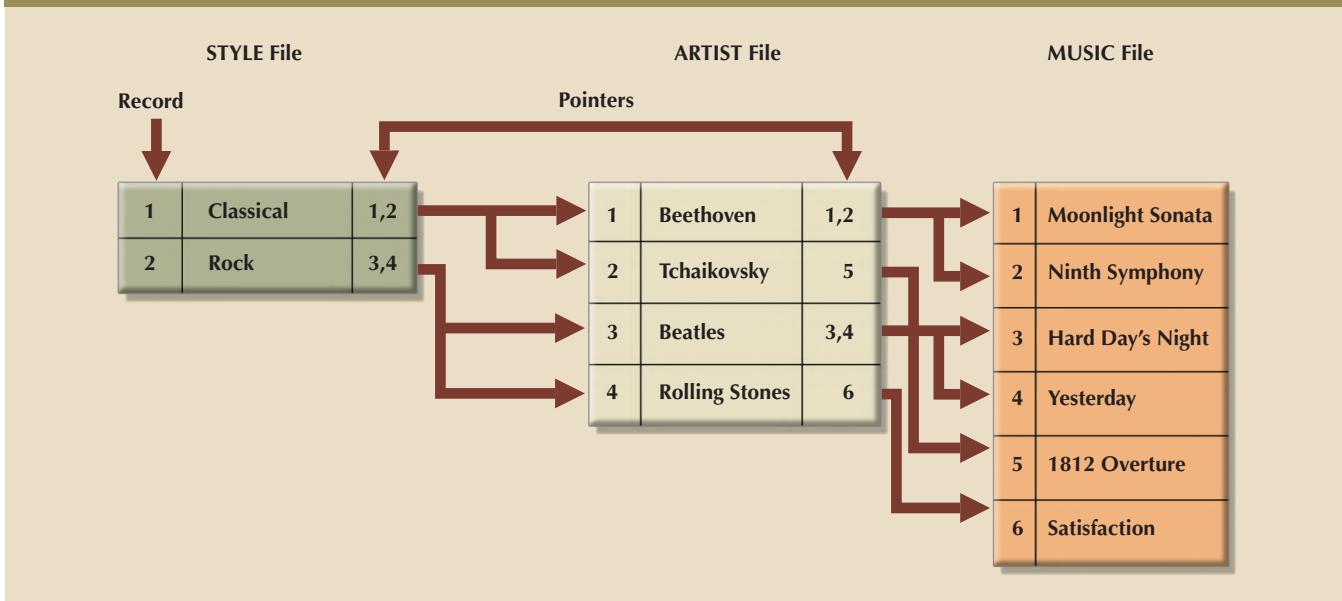
The tree structure depicted in Figure K.3 cannot be duplicated (as shown) on the computer's storage media. Instead, the tree is defined by the path that traces the parents to their children, beginning from the left. This ordered sequencing of segments to represent the hierarchical structure is known as the *hierarchical path*.

Given the structure depicted in Figure K.3, the hierarchical path for the record composed of the segments CUSTOMER, INVOICE, INVLINE, and PAYMENT can be traced as shown in Figure K.4. Note that the path followed in Figure K.4 traces all segments from the root, starting at the leftmost segment. The *left-list path* is known as the *preorder traversal* or the *hierarchic sequence*. Given such a path, designers must make sure that the most frequently accessed segments and their components are located closest to the tree's leftmost branches.

K-2 Contrasting File Systems with the Hierarchical Model

To help you better understand the segment concept, it might be useful to examine the relationship between file structures and the hierarchical database. For example, consider the small file system depicted in Figure K.5. Note that the three physical files are connected through the use of pointers. Thus, the *Classical* pointer in the STYLE file points to Beethoven and Tchaikovsky in the ARTIST file. In turn, the ARTIST file's pointers lead to specific music in the MUSIC file.

The file system depicted in Figure K.5 is composed of three distinct physical files: STYLE, ARTIST, and MUSIC. The records in each of the three files are physically isolated

FIGURE K.4 TRACING THE PATH OF A SINGLE HIERARCHICAL RECORD**FIGURE K.5 COMPOSITION OF A SMALL FILE SYSTEM**

from the records in the other files. Therefore, the first record in each of the three files may be depicted as shown in Table K.1.

In sharp contrast to the file system, the hierarchical model merges the separate physical files into a single structure known as a database. *Therefore, there is no equivalent of a file in the hierarchical model. The fields encountered in the file system are simply segment components in the hierarchical database.*

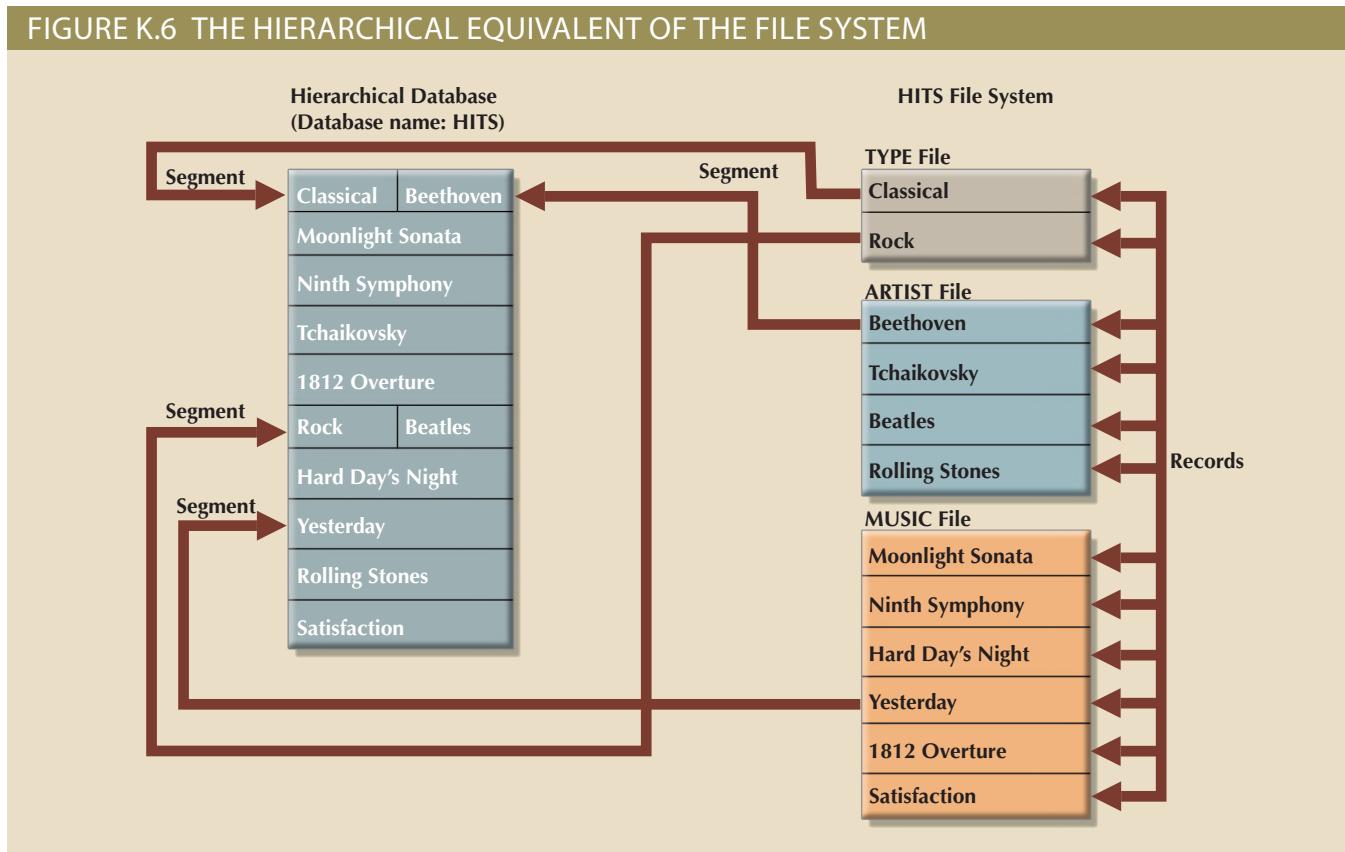
TABLE K.1

SAMPLE M_STORE FILE COMPONENTS

FILE	RECORD
STYLE	Classical
ARTIST	Beethoven
MUSIC	Moonlight Sonata

Translating the small file system into a hierarchical database (named HITS) yields a structure in which each file record becomes a database segment. Thus, the HITS database structure will be composed of three different segment types: STYLE, ARTIST, and MUSIC. Figure K.6 shows you how the file system's records are arranged within a hierarchical database.

FIGURE K.6 THE HIERARCHICAL EQUIVALENT OF THE FILE SYSTEM

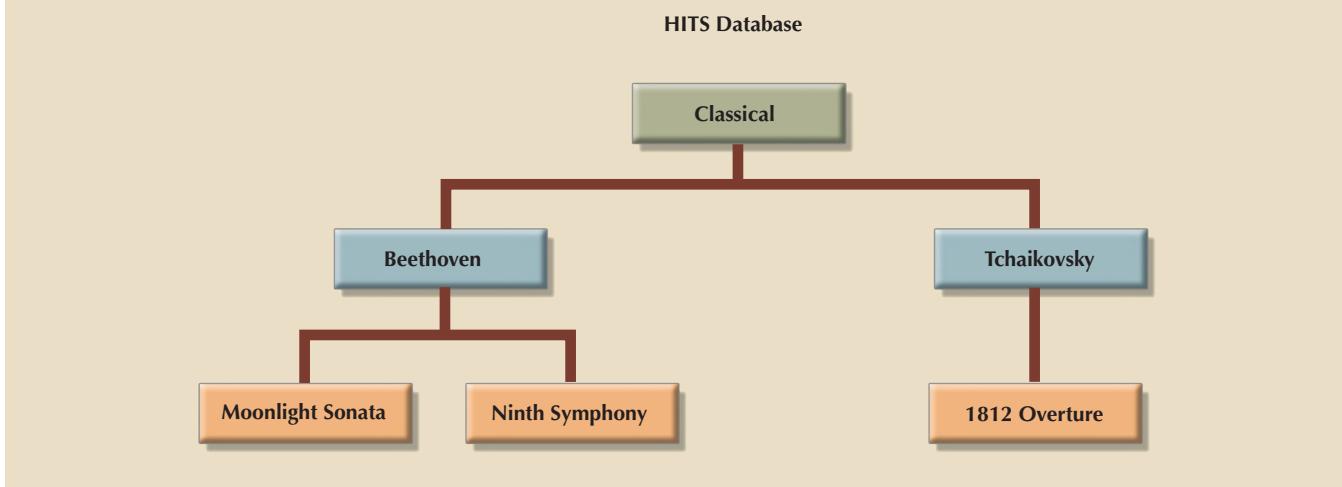


Given the contrasting structures shown in Figure K.6, keep in mind that the file system's user must maintain physical control of the indexes and pointers that validate data integrity. However, in the hierarchical model, the DBMS takes care of those complex chores, and the pointer movement is *transparent* to the user. (The word **transparent** indicates that the user is unaware of the system's operation.) Figure K.7 shows the hierarchical representation of the first database record for the HITS database.

transparent

Indicating that the user is unaware of the system's operations.

FIGURE K.7 THE FIRST HIERARCHICAL DATABASE RECORD



K-3 Defining a Hierarchical Database

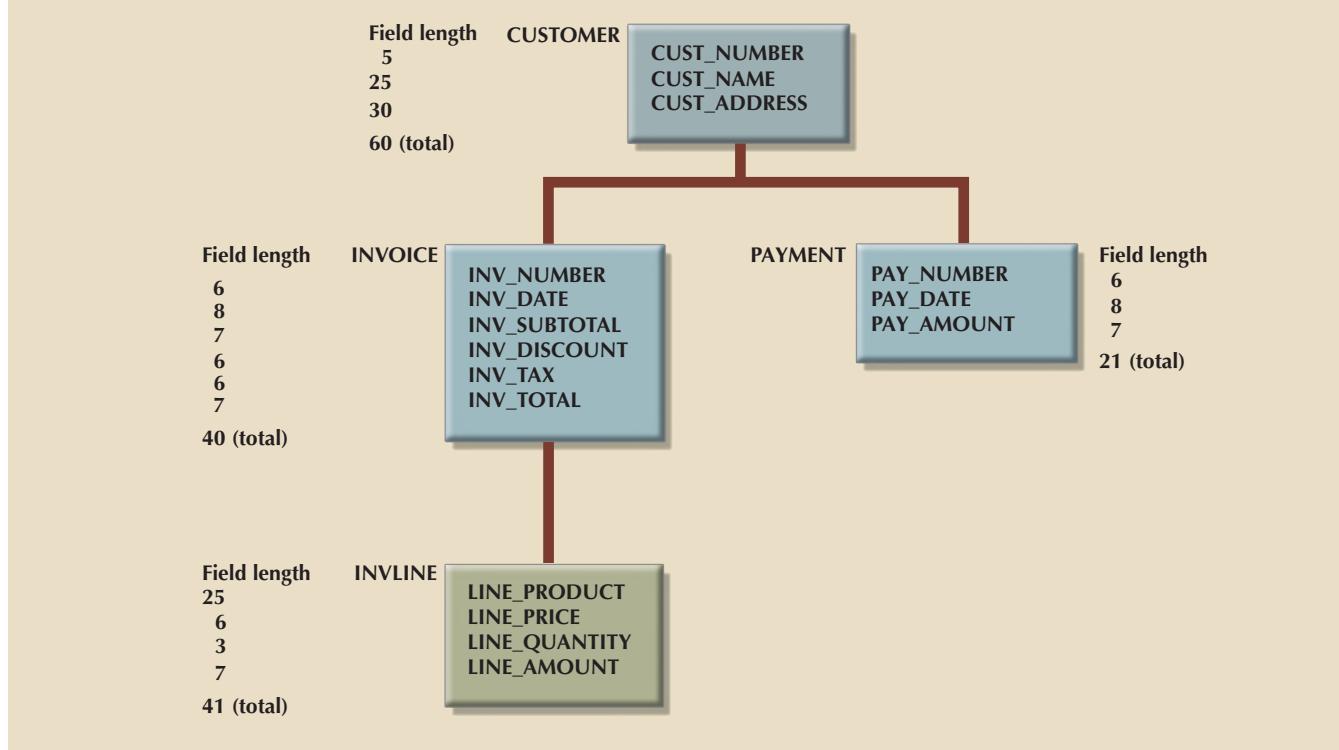
You will now learn how Figure K.2's simple billing system can be implemented through IBM's Information Management System (IMS). IMS uses a language named Data Language One (DL/1). At the conceptual level, IMS may control several databases. Each database is composed of a collection of physical records (segments) that are occurrences of a single tree structure. Therefore, each tree requires its own database. For example, the tree structure depicted in Figure K.8 may be stored in a database named CUSREC to reflect its CUStomer RECord orientation. (Incidentally, you could name the database GEORGE or SALLY; however, it is helpful to give the database a name that describes its contents.) Each of the physical databases is defined by a database description (DBD) statement when the database is created.

The hierarchical model's segment relationships are determined explicitly by the user when the database is defined, using the data definition language (DDL). Those segment relationships do *not* depend on the contents of a field in the child record (as was true in the relational model). Therefore, the relationships between the segments cannot be derived *via* each segment's components or fields. For an illustration of the use of the DDL, refer to the field names as shown in Figure K.8.

As you examine Figure K.8, note that the field lengths, measured in bytes, are shown next to each field. For example, the three fields describing the segment named CUSTOMER are CUST_NUMBER, CUST_NAME, and CUST_ADDRESS and their field lengths are 5, 25, and 30, respectively. Therefore, the segment named CUSTOMER is $5 + 25 + 30 = 60$ bytes long. The INVOICE segment is 40 bytes long, the INVLINE segment is 41 bytes long, and the PAYMENT segment is 21 bytes long. Therefore, the total hierarchical database record length is $60 + 40 + 41 + 21 = 162$ bytes.

To define the CUSREC database, a simplified syntax of DL/1, the data access-and-manipulation language of IMS, will be used. DL/1 is used to describe the conceptual and logical views of the database. The *conceptual view* encompasses the entire database as seen by the database administrator; the *logical view* describes the programmer's and user's perceptions of the database. Thus, the logical view is more restrictive, limiting the programmer/user to the portion of the database that is currently in use. The existence of logical views constitutes a security measure that helps avoid the unauthorized use of the database. Both the conceptual and logical views are necessary when the database administrator is working with a hierarchical database.

FIGURE K.8 SEGMENT FIELDS AND FIELD LENGTHS (BYTES) IN THE CUSREC DATABASE



K-3a The Conceptual View Definition

Remember from the discussion in Chapter 2 that the tree structure is defined starting from the left. Therefore, the sequence shown in the DDL conforms to the path:

CUSTOMER → INVOICE → INVLINE → PAYMENT

Based on that structure definition, Table K.2 shows the DL/1 statements used to define the conceptual view of the CUSREC database as seen by the database administrator.

TABLE K.2

DL/1 STATEMENTS THAT DEFINE THE CONCEPTUAL VIEW OF THE CUSREC DATABASE

STATEMENT #	CODE	STATEMENT
1	DBD	NAME=CUSREC, ACCESS=HISAM
2	SEGM	NAME=CUSTOMER, BYTES=60
3	FIELD	NAME=(CUST_NUMBER,SEQ,U), BYTES=5, START=1
4	FIELD	NAME=CUST_NAME, BYTES=25, START=6
5	FIELD	NAME=CUST_ADDRESS, BYTES=30, START=31
6	SEGM	NAME= INVOICE, PARENT=CUSTOMER, BYTES=40
7	FIELD	NAME=(INV_NUMBER,SEQ,U), BYTES= 6, START=1
8	FIELD	NAME=INV_DATE, BYTES=8, START=7
9	FIELD	NAME=INV_SUBTOTAL, BYTES=7, START=15

TABLE K.2

DL/1 STATEMENTS THAT DEFINE THE CONCEPTUAL VIEW OF THE CUSREC DATABASE (CONTINUED)

STATEMENT #	CODE	STATEMENT
10	FIELD	NAME=INV_DISCOUNT,BYTES=6,START=22
11	FIELD	NAME=INV_TAX,BYTES=6,START=28
12	FIELD	NAME=INV_TOTAL,BYTES=7,START=34
13	SEGM	NAME=INVLINE,PARENT= INVOICE,BYTES= 42
14	FIELD	NAME=LINE_PRODUCT,SEQ,M),BYTES=25,START=1
15	FIELD	NAME=LINE_PRICE,BYTES=7,START=26
16	FIELD	NAME=LINE_QUANTITY,BYTES=3,START=33
	FIELD	NAME=LINE_AMOUNT,BYTES=7,START=36
17	SEGM	NAME=PAYOUT,PARENT=CUSTOMER,BYTES=21
18	FIELD	NAME=(PAY_NUMBER,SEQ,U),BYTES= 6,START=1
19	FIELD	NAME=PAY_DATE,BYTES=8,START=7
20	FIELD	NAME=PAY_AMOUNT,BYTES=7,START=15
21	DBGEND	
22	FINISH	
23	END	

Table K.2's DL/1 lines describe the database and its contents this way:

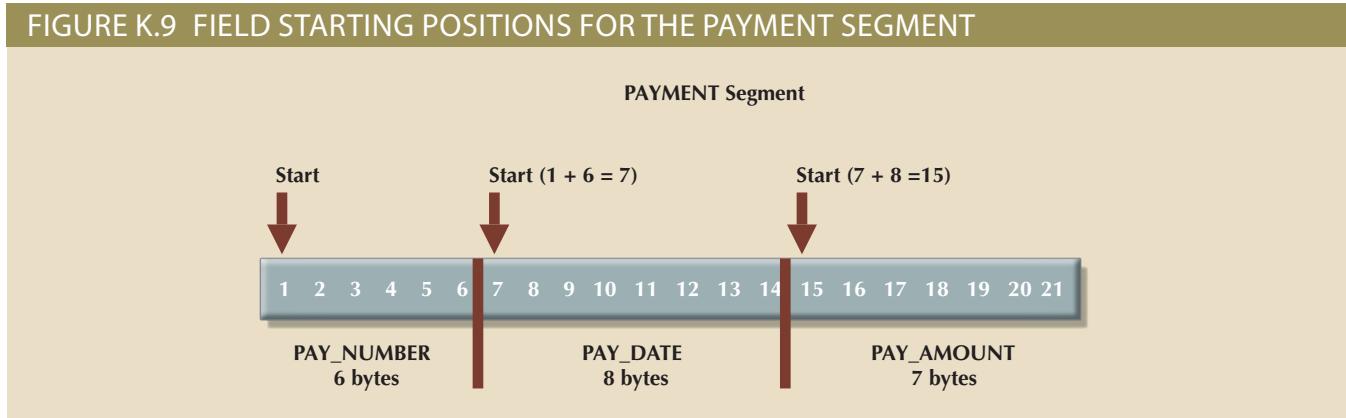
1. The first line tells IMS that a database named CUSREC is being defined. (The acronym *DBD* stands for *database description*.) The selected access mode is HISAM, or Hierarchical Indexed Sequential Access Method.
2. Line 2 defines the root segment; IMS uses the term **segment (SEGM)** to serve as a reference to the logical records of a database. This example defines the root segment to be the CUSTOMER segment, composed of the fields CUS_NUMBER, CUS_NAME, and CUS_ADDRESS. The CUSTOMER segment is 60 bytes long.
3. Lines 3–5 define the fields that are contained in the CUSTOMER segment. The FIELD specification defines the name, the size, and the starting position for each field making up a segment.
4. Line 3 defines CUST_NUMBER as the Sequence (SEQ) field for the CUSTOMER segment. By definition, the hierarchical database contains ordered collections of records. To avoid having two customers with the same customer number, the ID values are unique (U) for this field.
5. Line 6 defines the INVOICE segment; the parameter PARENT is used to indicate the parent of a segment. The parent of INVOICE is CUSTOMER in this example.
6. Note (in line 14) that there can be multiple (M) occurrences of the product's value.
7. Similar definitions are used for the remaining segments (INVLINE and PAYMENT).

segment (SEGM)

The equivalent of a file record type in the hierarchical database model."

8. **DBGEN** generates the physical database with all of its necessary structures. (The hierarchical database creation is *not* an interactive process.)
9. Note that the hierarchical model's implementation requires that you keep track of physical details such as the number of bytes and the starting position for each field. Figure K.9 illustrates how the starting position for each of the PAYMENT fields is determined.

FIGURE K.9 FIELD STARTING POSITIONS FOR THE PAYMENT SEGMENT



As you can see, the hierarchical model's database definition must conform to its physical characteristics. Even given the simplified DL/1 syntax, the details make the hierarchical model sufficiently complex to be described as a system designed by programmers for programmers. For instance, the physical storage details may require the definition of complex storage schemes such as:

- HSAM (Hierarchical Sequential Access Method).
- SHSAM (Simple Hierarchical Sequential Access Method).
- HISAM (Hierarchical Indexed Sequential Access Method).
- SHISAM (Simple Hierarchical Indexed Sequential Access Method).
- HDAM (Hierarchical Direct Access Method).
- HIDAM (Hierarchical Indexed Direct Access Method).
- MSDB (Main Storage DataBase).
- DEDB (Data Entry DataBase).
- GSAM (Generalized Sequential Access Method).

Specific access methods are best suited to particular kinds of applications. HSAM, SHSAM, HISAM, and SHISAM are particularly well suited for storing and retrieving data in hierarchic sequence, putting parent and children records in contiguous disk locations. (GSAM is a special case of the sequential access method.) On the other hand, if direct-access pointers are required to keep track of the hierarchy of segments, HDAM, HIDAM, MSDB, and DEBD are preferred. That series of access methods is generally more valuable when many (and frequent) changes are made to the database. Generally, the IMS manuals suggest that you:

1. Use HSAM when relatively small databases with relatively few access requirements are used.
2. Use HISAM with databases that require direct segment access, especially when:
 - a. Fixed record lengths are used.

DBGEN

In the hierarchical model, the component that generates the physical database with all its necessary structures.

- b. All segments are the same size.
 - c. Few root segments and many child segments exist.
 - d. Few deletions are made.
3. Use HDAM with databases designed for fast direct access.
 4. Use HIDAM with databases having users who require both random (direct) and sequential access.
 5. Use MSDB with databases that use fixed-length segments and that require very fast processing. MSDB will reside in virtual storage during execution.
 6. Use DEBD with databases that are characterized by high data volume.
 7. Use SHSAM, SHISAM, and GSAM when you frequently import and export data between database and nondatabase applications.

Table K.2's database definition requires each segment to be identified by a so-called **sequence field**. The identifier is also known as a **key**. Working with sequence fields requires that you recognize these features and conditions:

- Sequence fields allow direct access to segments when you are working with HISAM, HDAM, or HIDAM access methods. Those access methods make it possible to address segments directly, without having to search the entire database. Direct access increases performance substantially.
- Sequence fields do not have to be defined for every segment.
- Sequence fields may be either unique (U) or duplicate (M).

Keep in mind that an IMS database is rather limited structurally:

- Each database can have a maximum of 255 different segment types.
- Each segment can have a maximum of 255 segment fields.
- Each database can have a maximum of 1,000 different fields.

Having defined the conceptual view of the database, now let's define the logical views for each application program that will access the database.

K-3b The Logical View Definition

The logical view depicts the application program's view. Application programs use embedded DL/1 statements to manipulate the data in the database. Each application that accesses an IMS database requires the creation of a **program specification block (PSB)**. The PSB defines the database(s), segments, and types of operations that can be performed by the application. The PSB represents a logical view of a selected portion of the database. The use of PSBs yields better data security as well as improved program efficiency by allowing access to only the portion of the database that is required to perform a given function.

The application program and the database system communicate through a common storage area in primary memory known as the program communication block (PCB). The PSB contains one or more PCBs, one for each database that is accessed by the application program.

To illustrate the use of the PCB, let's create one for an application that displays customer payments. Since the program access requirements can be defined, you need only be in the database portion defined by the CUSTOMER and the PAYMENT segments shown in Figure K.8. You may then use DL/1 to define the type of access or **processing option (PROCOPT)** granted to the program. The access types are (G)et, (I)nsert,

sequence field

An attribute that contains values that are unique and sequential (ascending or descending). Some DBMS allows the explicit definition of sequence or autonumber attributes that are generally used to uniquely identify each row.

key

An entity identifier based on the concept of functional dependence; keys may be classified in several ways. See also *superkey*, *candidate key*, *primary key (PK)*, *secondary key*, and *foreign key*.

program

specification block (PSB)

In a hierarchical database, this represents a logical view of a selected portion of the database and also defines the database(s), segments, and types of operations that can be performed by the application. Using PSBs yields better data security as well as improved program efficiency by allowing access only to the portion of the database that is required to perform a given function.

processing option (PROCOPT)

A type of access granted to a program.

(R)eplace, and (D)elete. Table K.3 shows the appropriate DL/1 statements used to create the PSB for the application.

TABLE K.3

THE DL/1 STATEMENTS USED TO CREATE THE PSB

BLOCK	DL/1 STATEMENT	DEFINITION
1	PCB	DBNAME = CUSREC
2	SENSEG	NAME = CUSTOMER, PROCOPT = G
3	SENSEG	NAME = PAYMENT, PARENT = CUSTOMER, PROCOFF = G
4	SENFLD	NAME = PAY_DATE, START 8
5	SENFLD	NAME = PAY_AMOUNT, START 15
6	PSBCEN	LANG = COBOL, PSBNAME ROBPROG

The **SENSEG (SENsitive SEGment)** declares the segments that will be available, starting with the root segment. The SENFLD indicates which fields are available to the program. In Table K.3's example, all of the CUSTOMER fields will be available, but only the PAY_DATE and PAY_AMOUNT will be available in the PAYMENT segment because the PAY_NUMBER field was omitted. (*Note: The logical views may be limited to only a portion of a physical database or to parts of several different physical databases.*)

The creation of the database structure and the PSBs is not based on interactive operations. Instead, independent utility programs that run from the operating-system prompt must be used. Therefore, the database definitions must be re-created (recompiled) and reloaded, *and all of the user views must be re-created and validated if any changes are made to the database.*

The order of the SEGM statements indicates the physical order of the records in the database. In other words, the physical order represents the hierarchical path that must be followed to access any segment. In this case, the order of the segments is shown in Table K.4.

TABLE K.4

THE HIERARCHICAL PATH FOR THE CUSREC DATABASE

HIERARCHICAL PATH	SAMPLE DATA
CUSTOMER 1	Mary D. Allen
INVOICE 1	102
INVLINE 1	Glue gun
INVLINE 2	Drill bit
INVLINE 3	Chisel
INVOICE 2	324
INVLINE 1	Power saw
PAYMENT 1	1243
PAYMENT 2	1985
CUSTOMER 2	John G. Washington

SENSEG (SENsitive SEGment)

In the IMS hierarchical database, this keyword declares the segments that will be available, starting with the root segment.

TABLE K.4

THE HIERARCHICAL PATH FOR THE CUSREC DATABASE (CONTINUED)

HIERARCHICAL PATH	SAMPLE DATA
INVOICE 1	410
INVLINE 1	Grease pencils
INVLINE 2	Masking tape
INVOICE 2	306
INVLINE 1	Computer paper
INVLINE 2	Ink-jet cartridge
...	...
...	...

Remember that IMS provides support for several different data-access methods. Some are very efficient at sequential file processing; others work well in an indexed file environment; yet others work best in a direct-access environment. The example shown in Table K.3 assumes the use of the HSAM storage structure in which the database is represented as an ordered sequence of segments and all dependent segments are located close to their parent segments for fast sequential access.

K-4 Loading IMS Databases

An IMS database must be loaded before any program can access it. You cannot load a database from an interactive application program. Instead, a batch program must be used to perform the loading, and this batch program must be run in “load” mode (PRO-COPT=L in the PCB).

The database must be loaded in the proper hierarchic sequence; *the segment order is crucial*. (Load the parent segments before loading the child segments!) If you have defined sequence fields, the segment order must conform to the sequence field order. You must maintain the proper segment order, or the subsequent applications programs will fail.

K-5 Accessing the Database

Hierarchical databases are so-called *record-at-a-time* databases. The term **record at a time** indicates that the database commands affect a single record at a time. You may remember that other database types, such as the relational database, allow a command to affect several (many) records at a time.

The record-at-a-time structure implies that each record is accessed independently when database operations are performed. Therefore, to access a specific record, you must follow the tree’s hierarchical path, starting at the root and following the appropriate branches of the tree, using preorder traversal. For example, if you want to access the payments of CUSTOMER Mary D. Allen, you must first access the parent segment, after which you can access the first PAYMENT child, then the next PAYMENT child, and so on, until you have accessed all PAYMENT segments in the subtree. (Remember that PAYMENT segments are ordered by the PAY_NUMBER field.) Similarly, if you want to access the INVOICE segment occurrences, you must first access the parent CUSTOMER segment; then you must access the INVOICE segment occurrences, starting with the first

record at a time
This term indicates that the database commands affect a single record at a time.

one. For each INVOICE, you can access the subtree of INVLINE segment occurrences for that INVOICE. (Remember that the segments are ordered according to the field specified as the sequence field when the database is defined.)

After the database and its characteristics have been defined, you can navigate through the database by using the data manipulation language (DML) invoked from some host language such as COBOL, PL/1, or assembler. Keep in mind that some lines of code must be written by an experienced programmer before you can access the database. Given the complexity of the hierarchical database environment, end users are not likely to have the technical expertise to generate even the simplest query output, thus putting “spur-of-the-moment” queries out of reach. For example, a query such as “list all customers who reside in the 12345 zip code” requires detailed knowledge of the hierarchical database’s physical file structure and the physical storage details. (In contrast, that query is easy to generate in a relational database environment, merely requiring the execution of the brief SQL command `SELECT * FROM CUSTOMER WHERE CUST_ZIP = “12345”`.)

IMS requires the use of a (3GL) host language such as COBOL to access the database. To communicate with the application program correctly, IMS assumes the use of certain parameters. Therefore, each application must declare:

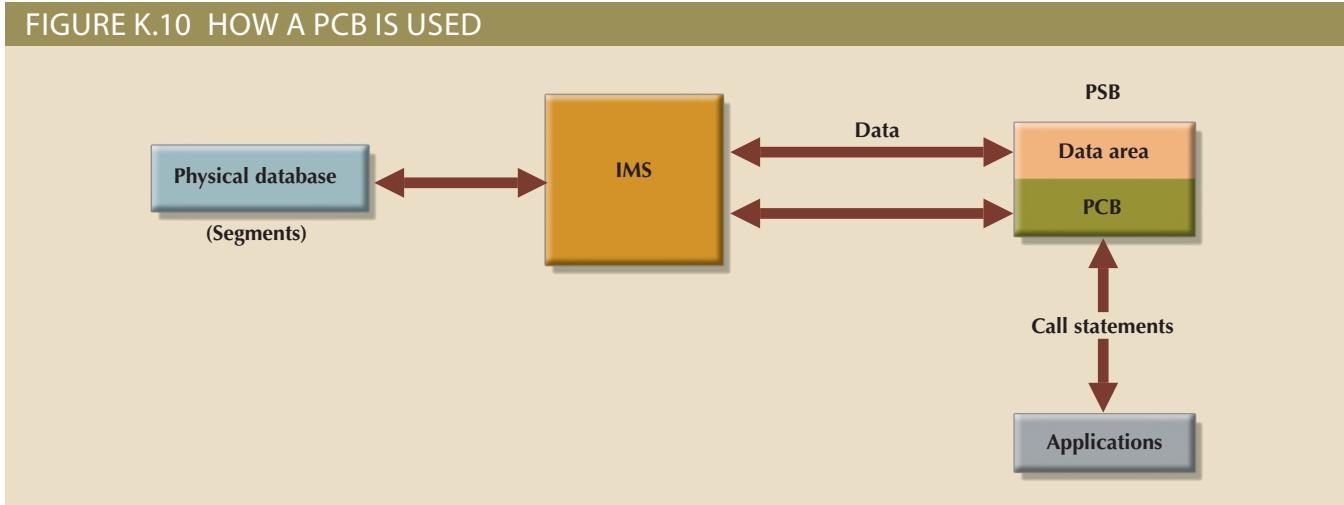
1. An *input area* (program record area) reserved to:
 - a. Receive the data retrieved from the database.
 - b. Temporarily store data to be written into the database.
2. A PCB to store a return code for every operation that is executed. (The program must check this area to see if the requested operation was completed successfully.)

A COBOL application communicates with the IMS DBMS through *call* statements in its procedure division. Figure K.10 illustrates the use of the PCB. When the application program calls IMS, the following *flow parameters* are needed:

- The function code, that is, the operation to be executed on the database.
- The PCB name to be used.
- The input area address.
- The (optional) segment search argument (SSA). The SSA parameter identifies the key of the segment being retrieved.

After the completion of a call to the database, the program must check the status of the return code in the PCB to ensure that the operation was executed correctly.

FIGURE K.10 HOW A PCB IS USED



K-5a Data Retrieval: Get Unique

The IMS statement **Get Unique (GU)** is used to retrieve a database segment into the application program input area or record area. The syntax for the Get Unique statement is:

CU (segment) (SSA)

Using the data shown earlier in Figure K.3, the GU statement required to retrieve the customer Mary D. Allen must read:

GU CUSTOMER (CUST_NUMBER=1276)

Similarly, the GU statement:

GU INVOICE (INV_NUMBER=102)

will retrieve the INVOICE segment whose number is 102. If HSAM is being used, the DBMS will search the database sequentially until it finds the INVOICE segment whose field value INV_NUMBER is 102. If this INV_NUMBER is the last segment in the database, the DBMS will have searched the entire database, thereby producing significant performance degradation. It is strongly recommended that the hierarchical path be specified to maximize the DBMS performance!

Retrieval by a nonkey field is also possible; for example:

GU CUSTOMER (CUST_NAME 'Mary D. Allen')

will achieve its intended purpose. If two customers have the same name, that command will retrieve the *first* segment that satisfies the condition.

Logical operators may be used to search for several customer records that meet a specified condition. For instance, the GU statements:

GU CUSTOMER (CUST_NUMBER > 1034)

and

GU CUSTOMER (CUST_NUMBER <= 1167)

are both valid. If the user fails to specify the SSA, the database search automatically locates the first segment of the database. Therefore, the GU statement:

GU CUSTOMER

will yield the first CUSTOMER segment.

IMS can retrieve more than one segment at a time. For example, if you want to access the INVOICE segment with an INV_NUMBER = 102 as well as its parent CUSTOMER segment, the command:

GU CUSTOMER *D

INVOICE (IN_NUMBER=102)

will retrieve both the parent CUSTOMER segment and the *specified* INVOICE child segment; the *D indicates that the user wants to retrieve both. In contrast:

GU CUSTOMER

INVOICE

will retrieve only the first INVOICE segment found. IMS always retrieves the *last* referenced segment unless the *D is used.

Get Unique (GU)

In an IMS hierarchical DBMS, a statement that is used to retrieve a database segment into the application program input area or record area.

K-5b Sequential Retrieval: Get Next

The **Get Next (GN)** statement is used to retrieve segments sequentially. (Naturally, the retrieval sequence is based on the preorder traversal requirements.) The GN syntax conforms to the format:

GN (segment) SSA

For example, the statements in Table K.5 will retrieve all payments. (Note that *Pseudocode* has been used to indicate the use of some programming language to complete the request.)

TABLE K.5

RETRIEVE ALL PAYMENTS

PSEUDOCODE	COMMENTS
GU PAYMENT	Retrieve 1st PAYMENT segment.
DO WHILE PCB-CODE IS OKAY	Check the PCB return code.
PRINT (PAY_NUMBER, PAY_DATE)	Process the segment.
ENDDO	

Similarly, if you want to retrieve all payments over \$1,000, you would write the pseudocode shown in Table K.6.

TABLE K.6

RETRIEVE ALL PAYMENTS OVER 1000

PSEUDOCODE	COMMENTS
GU PAYMENT (PAY_AMOUNT > 1000)	Retrieve the first PAYMENT segment.
DO WHILE PCB-CODE IS OKAY	Check the PCB return code.
PRINT (PAY_NUMBER, PAY_DATE)	Print the requested data.
GNPAYMENT (PAY_AMOUNT > 1000)	Retrieve the next PAYMENT segment.
ENDDO	



Note

The use of OKAY indicates that the return code is correct. The return code is part of the PCB.

Get Next (GN)

In hierarchical databases, a statement to retrieve sequential segments.

Get Next within Parent (GNP)

In hierarchical databases, a statement to return all segments within the current parent.

K-5c Get Next within Parent

Get Next within Parent (GNP) will return all of the segments within the current parent. The following command sequence will retrieve all INVOICE segments for the CUSTOMER whose CUST_NUMBER=1276 in the preorder traversal sequence shown in Table K.7.

TABLE K.7

RETRIEVE INVOICES FOR SPECIFIED CUSTOMER

PSEUDOCODE	COMMENTS
GU CUSTOMER (CUSTOMER=1276)	Retrieve the first INVOICE segment for customer 1276.
INVOICE	
DO WHILE PCB-CODE IS OKAY	
.....	
.....(process segment)	
.....	
GNP INVOICE	Retrieve the next INVOICE segment for customer 1276.
ENDDO	

K-5d Data Deletion and Replacement

The **Get Hold (GH)** statement is used to hold a segment for delete or replace operations. There are three different Get Hold statements, as shown in Table K.8.

TABLE K.8

GET HOLD STATEMENTS

STATEMENT	MEANING
GHU	Get Hold Unique
GHN	Get Hold Next
GHNP	Get Hold Next Within Parent

Used in combination with the GH statement, DLET deletes a segment occurrence from the database. For example, to delete the PAYMENT segment numbered 1985 in Table K.3, you would use:

```
GHU CUSTOMER (CUSTOMER=1276)
    PAYMENT (PAY_NUMBER=1985)
DLET
```

If a root segment is deleted, all dependent segments are deleted. Therefore, the command sequence:

```
CHU CUSTOMER (CUST_NUMBER=1276)
DLET
```

will delete the occurrence Mary D. Allen in the CUSTOMER segment and all dependent segments (INVOICE, INVLINE, and PAYMENT).

The REPL statement allows you to change (update) the contents of a field within a segment. REPL also requires the GH operation before it can be invoked. Keep in mind that the REPL function cannot be used to update a key field. Instead, first delete the record, then insert the updated version. The application program should use the input

Get Hold (GH)

In an IMS hierarchical DBMS, this statement is used to hold a segment for delete or replace operations. There are three different Get Hold statements: Get Hold Next (GHN), Get Hold Next within Parent (GHNP), and Get Hold Unique (GHU).

area to store the necessary fields that are to be updated and the new values. The operation sequence thus becomes:

1. Retrieve the data and put it in the input area.
2. Make the changes in the input area.
3. Invoke REPL to move the changed values into the physical database.

For example, to change Mary Allen's address, you can use the pseudocode shown in Table K.9.

TABLE K.9

UPDATE FIELD CONTENTS FOR A SPECIFIED CUSTOMER

PSEUDOCODE	COMMENTS
GU CUSTOMER (CUST_NUMBER = 1276)	Find the CUSTOMER segment.
STORE '103 E. Main St. D-44' TO CUST_ADDRESS	Move the data to the input area.
REPL	Save the data to the disk.

K-5e Adding a New Segment to the Database

The **Insert (ISRT)** statement is used to add a segment to the database. The parent segment must already exist if a child segment is to be inserted. The segment will be inserted in the database in the sequence field order specified for the segment.

The input area in the applications program must contain the data to be stored in the segment. Therefore, if you want to insert the segment PAYMENT for customer number 1276, you write the pseudocode shown in Table K.10.

TABLE K.10

ADDING A NEW SEGMENT

PSEUDOCODE	COMMENTS
STORE 1632 TO PAY_NUMBER	Move the data into the input area.
STORE '20180315' TO PAY_DATE	
STORE 345.66 TO PAY_AMOUNT	
ISRT CUSTOMER (CUS_NUMBER= 1276)	Insert the field values. (Naturally, customer 1276 must exist in the database.)
PAYMENT	

K-6 Logical Relationships

Suppose you want to keep product information in the database system. Further suppose that the product information is to be stored in an INVENTORY database and that you want this database to be related to the CUSREC database. Because the invoice lines contain product information, the PRODUCT segment in the INVENTORY database must be related to the INVLINE segment in the CUSREC database.

Given the preceding scenario, you face the problem of having a segment with two parents, a condition that cannot be easily supported by the hierarchical model. The

Insert (ISRT)

In hierarchical databases, a statement used to add a segment to the database.

multiple-parent problem can be solved by creating a logical relationship between INVLIN and PRODUCT in which INVLIN becomes the logical child of PRODUCT and PRODUCT becomes the logical parent of INVLIN. Unfortunately, this solution has some drawbacks.

- Implementing such a solution yields an even more complex applications environment.
- Creating logical parent/child relationships is very complex and requires the services of an experienced programmer. To accomplish the task, referential rules must be defined for each of the operations (Insert, Replace, and Delete) for each logical segment involved in the two physical databases. The rules may be unidirectional or bidirectional depending on which way the database is to be accessed.

Nonetheless, using logical relationships, you can link two independent physical databases and treat them as though they were one. Thus, logical relationships allow you to reduce data redundancy. In addition, IMS can manage all of the data required to link the databases in logical relationships; it is always better to have the DBMS software do the delicate work of keeping track of such data rather than trust the applications software to do those chores.

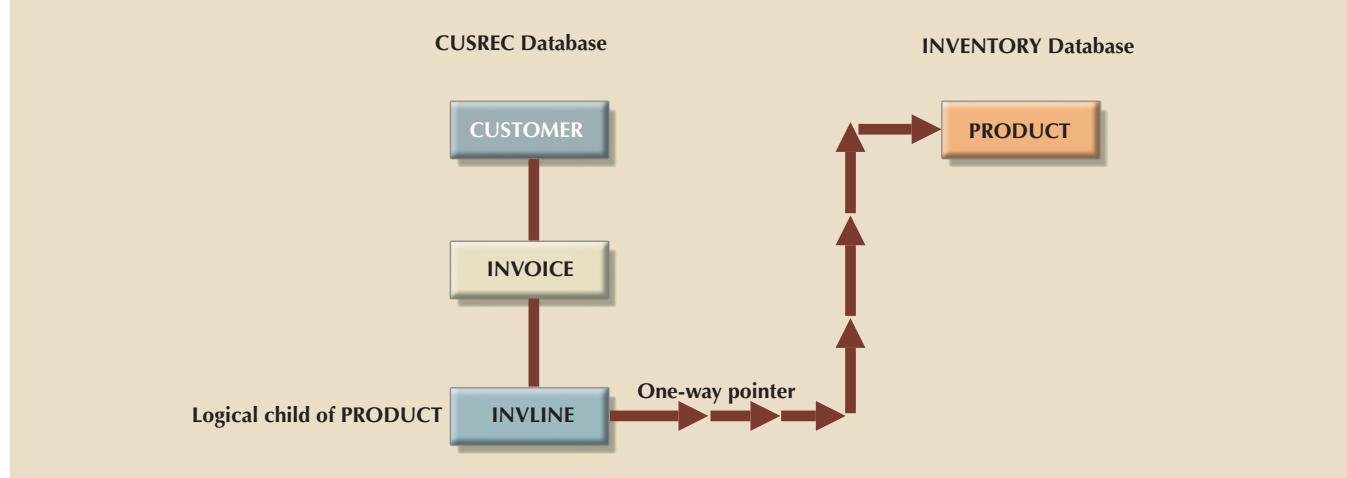
IMS supports three different types of logical relationships, as follows:

1. **Unidirectional logical relationships** are established by linking a logical child with a logical parent in a one-way arrangement. In this case, a pointer in the logical child points to the logical parent. (See Figure K.11.)

unidirectional logical relationships

In a hierarchical database, relationships that are established by linking a logical child with a logical parent in a one-way arrangement.

FIGURE K.11 A UNIDIRECTIONAL LOGICAL RELATIONSHIP



The two segments may be in the same database, or they may be located in different databases. If the two segments of the unidirectional relationship are located in different databases, the segments are treated independently of one another. Therefore, if a parent segment is deleted, the logical children are not deleted (see Figure K.12) because the logical parent does not point to the logical child.

1. **Bidirectional physically paired logical relationships** link a logical child with its logical parent in two directions. IMS creates a duplicate of the child segment in the logical parent's database and manages all operations (Insert, Delete, Replace) applied to the segments, as shown in Figure K.13. IMS uses pointers in the logical child segments pointing to their logical parents. The segments may be in one database, or

bidirectional physically paired logical relationships

In the hierarchical model, a relationship that links a logical child with its logical parent in two directions.

FIGURE K.12 TWO UNIDIRECTIONAL LOGICAL RELATIONSHIPS

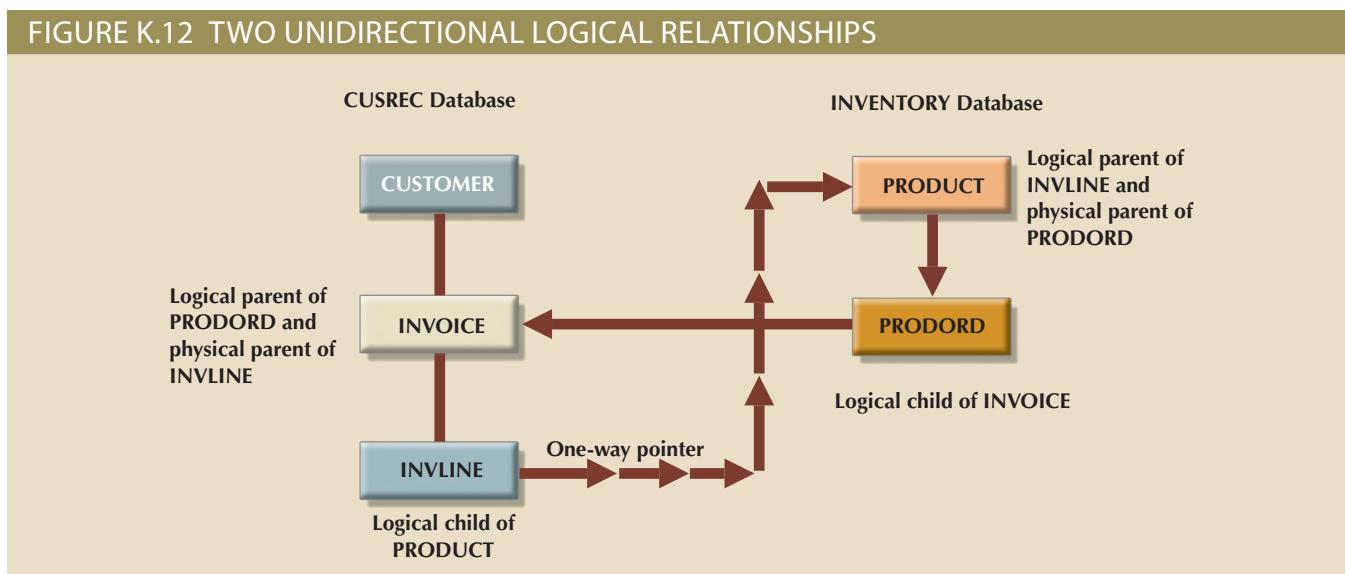
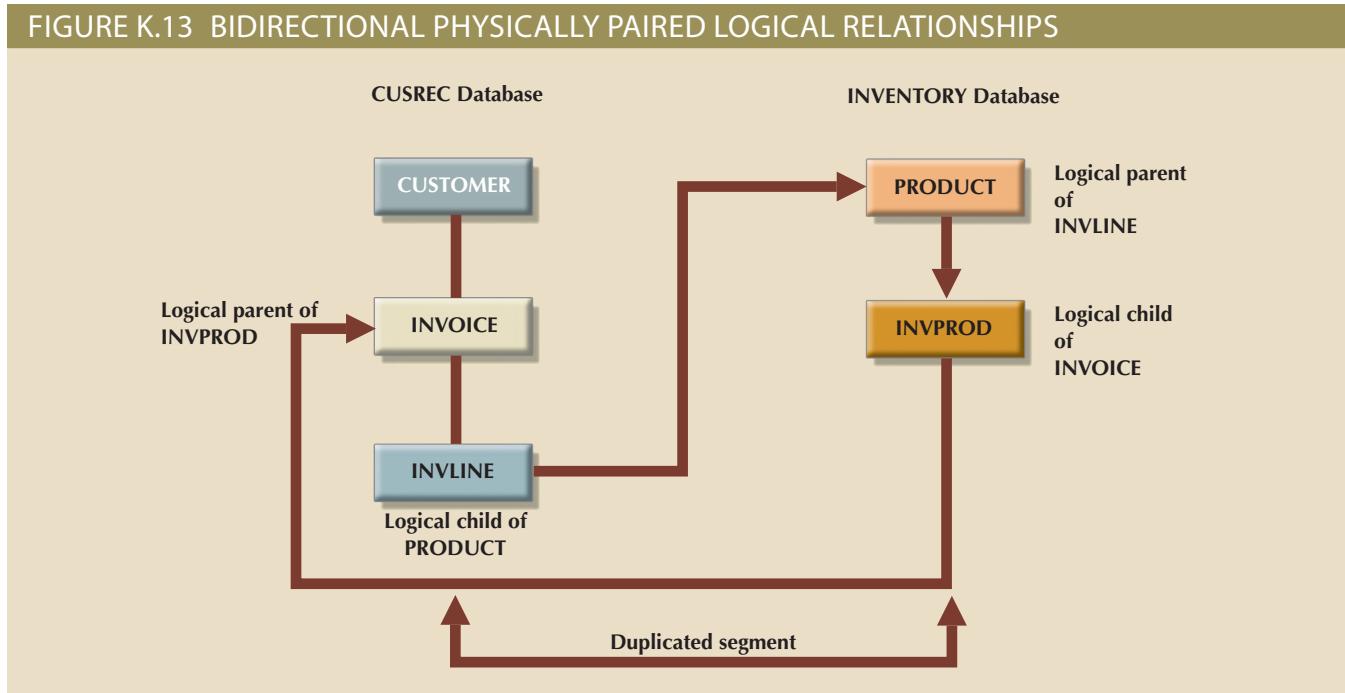


FIGURE K.13 BIDIRECTIONAL PHYSICALLY PAIRED LOGICAL RELATIONSHIPS



they may be in different (physical) databases. In this type of relationship, the user can navigate from the CUSREC database to the INVENTORY database, and vice versa, because a two-way link exists between the INVOICE and the PRODUCT segments through their common child INVLINE. Although the process creates data redundancy, IMS manages the redundancies transparently.

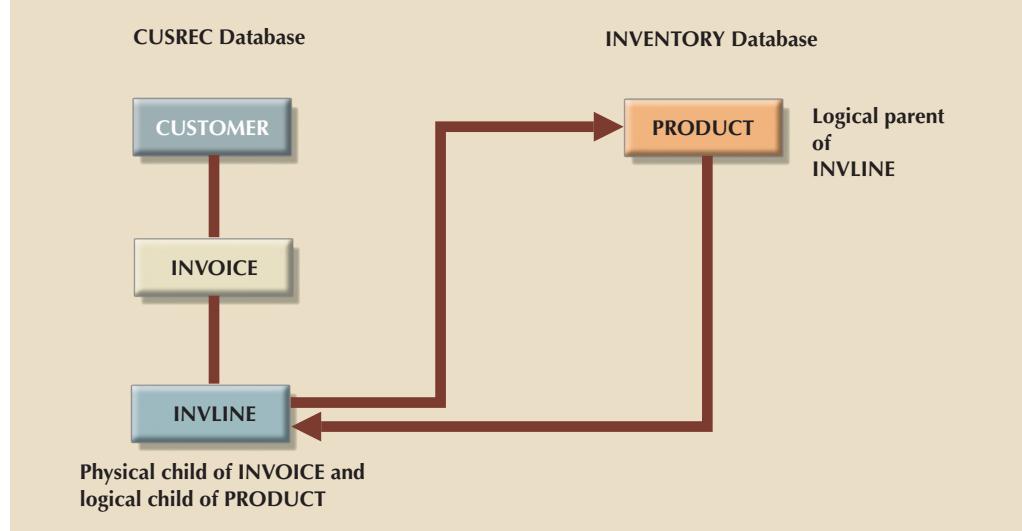
2. **Bidirectional virtually paired logical relationships** are created when a logical child segment is linked to its logical parent in two directions. The virtually paired relationship is different from the physically paired relationship in that no duplicates are created; IMS stores one pointer in the logical parent to point to the logical child's database and another pointer in the logical child to point to the logical parent. Thus, the virtually paired method reduces data duplication and overhead in the management of both hierarchical paths. (See Figure K.14.)

**bidirectional
virtually paired
logical relationships**

In the hierarchical model, a relationship created when a logical child segment is linked to its logical parent in two directions.

The virtually paired relationship is different from the physically paired relationship in that no duplicates are created.

FIGURE K.14 A BIDIRECTIONAL VIRTUALLY PAIRED LOGICAL RELATIONSHIP



The creation of bidirectional virtually paired logical relationships is a delicate, cumbersome task that requires a skilled designer with extensive knowledge of the physical details this task requires. For example, if you want to implement logical relationships, IMS requires that you follow the rules listed in Table K.11.

TABLE K.11

RULES FOR DEFINING LOGICAL RELATIONSHIPS IN PHYSICAL DATABASES

RULE	LOGICAL CHILD
1	A logical child must have a physical and a logical parent.
2	A logical child can have only one physical and one logical parent.
3	A logical child is defined as a physical child in the physical database of its physical parent.
4	A logical child is always a dependent segment in a physical database and can, therefore, be defined at any level except the first level of the database.
5	In its physical database, a logical child cannot have a physical child defined at the next lower level in the database that is also a logical child.
6	A logical child can have a physical child. However, if the logical child is physically paired with another logical child, only one of the paired segments can have physical children.

TABLE K.11

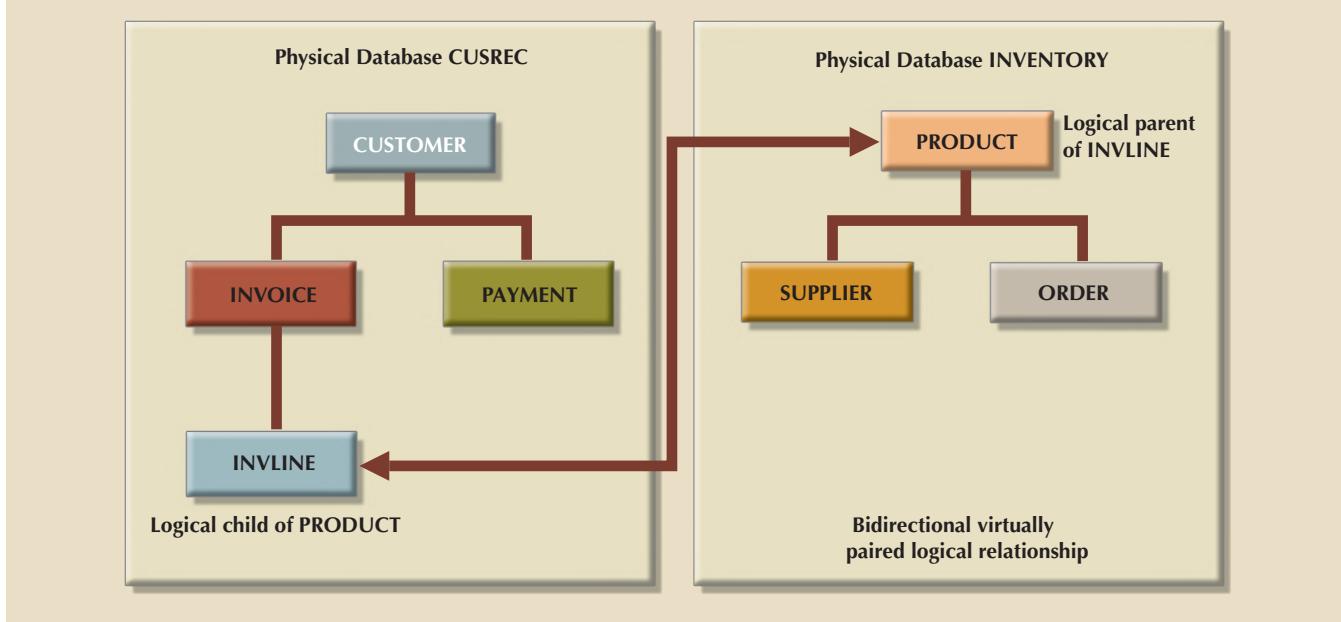
RULES FOR DEFINING LOGICAL RELATIONSHIPS IN PHYSICAL DATABASES (CONTINUED)

RULE	LOGICAL PARENT
1	A logical parent can be defined at any level in the physical database, including the root level.
2	A logical parent can have one or more logical children. Each logical child related to the same logical parent defines a logical relationship.
3	A segment in a physical database cannot be defined as both a logical parent and a logical child.
4	A logical parent can be defined in the same physical database as its logical child or in a different database.
RULE	PHYSICAL PARENT
1	A physical parent of a logical child cannot also be a logical child.

Source: IBM IMS Manual, IMS/ESA Version 3 Database Administration Guide, Release 1, 2nd ed., October, 1990, Purchase, NY 10577, pp. 155–56.

Assuming that the designer has the required knowledge of the implementation details, you can conclude that using logical relationships solves the problem of relating INVLINE and PRODUCT by creating a logical link between the two database segments, as shown in Figure K.15.

FIGURE K.15 A BIDIRECTIONAL VIRTUALLY PAIRED LOGICAL RELATIONSHIP BETWEEN TWO DATABASES



Based on the structure shown in Figure K.15, PRODUCT will be the logical parent of INVLINE and INVLINE will be the logical child of PRODUCT. Therefore, (I)nsert, (R)eplace, and (D)elete rules must be defined for each segment in the relationship—for CUSTOMER, INVOICE, and INVLINE in the CUSREC database and for PRODUCT in the INVENTORY database. For example, if a CUSTOMER segment is erased, all of the corresponding CUSTOMER children must be erased, too.

Similarly, if a PRODUCT segment is to be deleted, all of the corresponding INVLINE segments must also be deleted.

The use of logical parents is rather limited. One of DL/1's restrictions is that any given segment can have only one logical parent. That restriction severely limits IMS's ability to deal with complex structures. In fact, the two-parent problem is one of the reasons the network model examined in Appendix L was developed.

K-7 Altering the Hierarchical Database Structure

The hierarchical model's database structure modifications are cumbersome. For example, suppose the sales department manager asks the data processing department's database administrator to add a VENDOR field to the INVOICE segment. That is a simple request, yet even that minor alteration is not naturally supported by the hierarchical system.

Database modifications require the performance of the following tasks in sequence:

1. Unload the database.
2. Define the new database structure.
3. Load the old database into the new structure.
4. Delete the old database.

Since those four tasks are time-consuming and potentially dangerous from a database point of view, database structure modifications require very careful planning, excellent system coordination skills, and a high level of technical understanding of the DBMS.

Key Terms

bidirectional physically paired logical relationships, K-19	Get Next within Parent (GNP), K-16	record at a time, K-13
bidirectional virtually paired logical relationships, K-21	Get Unique (GU), K-15	segment (SEGM), K-9
database description (DBD) statement, K-2	Insert (ISRT), K-18	SENSEG (SENSitive SEGment), K-12
DBGEN, K-10	key, K-11	sequence field, K-11
Get Hold (GH), K-17	processing option (PROCOPT), K-11	transparent, K-6
Get Next (GN), K-16	program communication block (PCB), K-2	unidirectional logical relationships, K-19
	program specification block (PSB), K-11	

Appendix L

The Network Database Model

Preview

In this appendix, you will learn about network database model implementation. (You learned about the network database model concepts in Chapter 2, Data Models.) Like the hierarchical database model, the network model may be represented by a tree structure in which 1:M relationships are maintained. However, the network model easily handles complex multiparent relationships without resorting to the creation of logical (as opposed to physical) database links.

Also remember from Chapter 2 that a close kinship exists between hierarchical and network models. For example, the network model's owner corresponds to the hierarchical model's parent, and the network model's member corresponds to the hierarchical model's child. However, the network model places a set between the owner and the member, using that set to describe the relationship between the two. You will see that the set makes it possible to describe more complex relationships between owners and members than was feasible in the hierarchical model.

Although the pointer movement is more complex in the network model than in its hierarchical counterpart, the DBMS creates and maintains the pointer system, making it transparent to the user and even to the applications programmer. However, the cost of such pointer system transparency is greater system complexity. For example, you will learn that the schema requires careful delineation of the model's components. In short, the network model requires the database administrator to pay close attention to the model's physical environment. In turn, application programmers must note the model's physical details.

Data Files and Available Formats

[MS Access](#) [Oracle](#) [MS SQL](#) [My SQL](#)

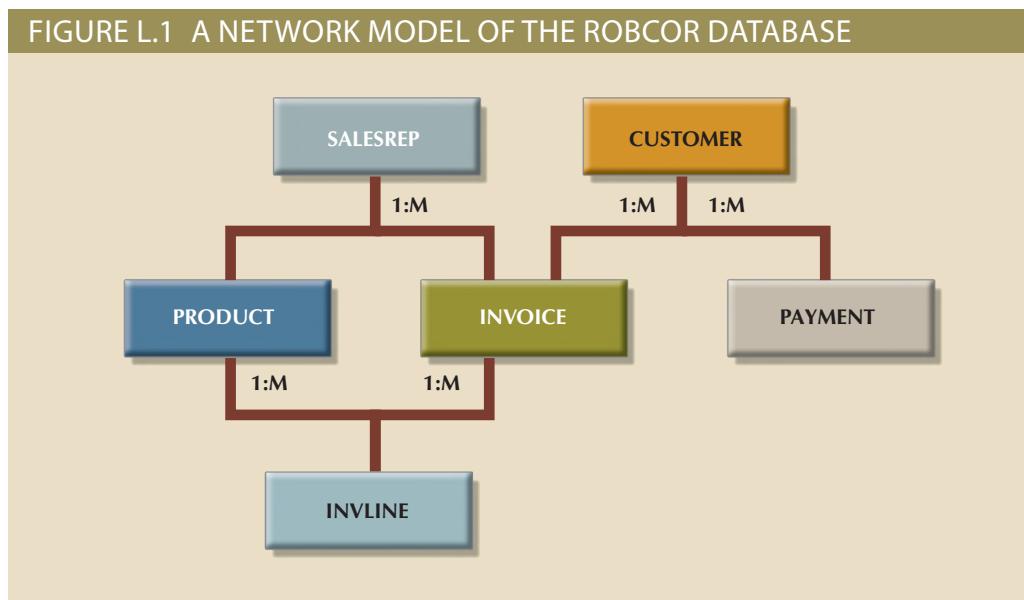
[MS Access](#) [Oracle](#) [MS SQL](#) [My SQL](#)

There are no data files for this appendix.

Data Files Available on cengagebrain.com

L-1 A Quick Review of Basic Network Data Model Concepts

In Chapter 2, you learned that the network model's end users *perceive* the network database to be a collection of records in one-to-many (1:M) relationships. But unlike the hierarchical database model, a record in the network database model can have more than one parent. Both 1:M and multiparent relationships are evident in the ROBCOR database shown in Figure L.1. Figure L.1 depicts a simple network database model for the ROBCOR Corporation. ROBCOR engages in retail sales, and its management wants to automate both the sales and the billing operations.



The following basic network model concepts are illustrated in Figure L.1:

- SALESREP, CUSTOMER, PRODUCT, INVOICE, PAYMENT, and INVLINE represent record types.
- INVOICE is owned by both SALESREP and CUSTOMER. Similarly, INVLINE has two owners, PRODUCT and INVOICE.
- The network database model may still include *hierarchical* one-owner relationships (for example, CUSTOMER owns PAYMENT).

Finally, relationships among records must be decomposed into a series of *sets* before a network database model can be implemented. For example, Figure L.2 shows two sets that describe the relationships between owners and members.

1. The SALES set includes all of the INVOICE tickets that belong to a specific CUSTOMER.
2. The PAIDBY set defines the relationship between CUSTOMER (the owner of the set) and PAYMENT (the member of the set).

ROBCOR's network database model contains other sets, too. In fact, before the network database model can be implemented, *all* of its data structures must be decomposed into sets of 1:M relationships. The sets that can be defined for Figure L.2's network database model are listed in Table L.1. Each set listed represents a relationship between owners and members. When you implement a network database design, every set must be given a name and all owner and member record types must be defined.

FIGURE L.2 SET ILLUSTRATION

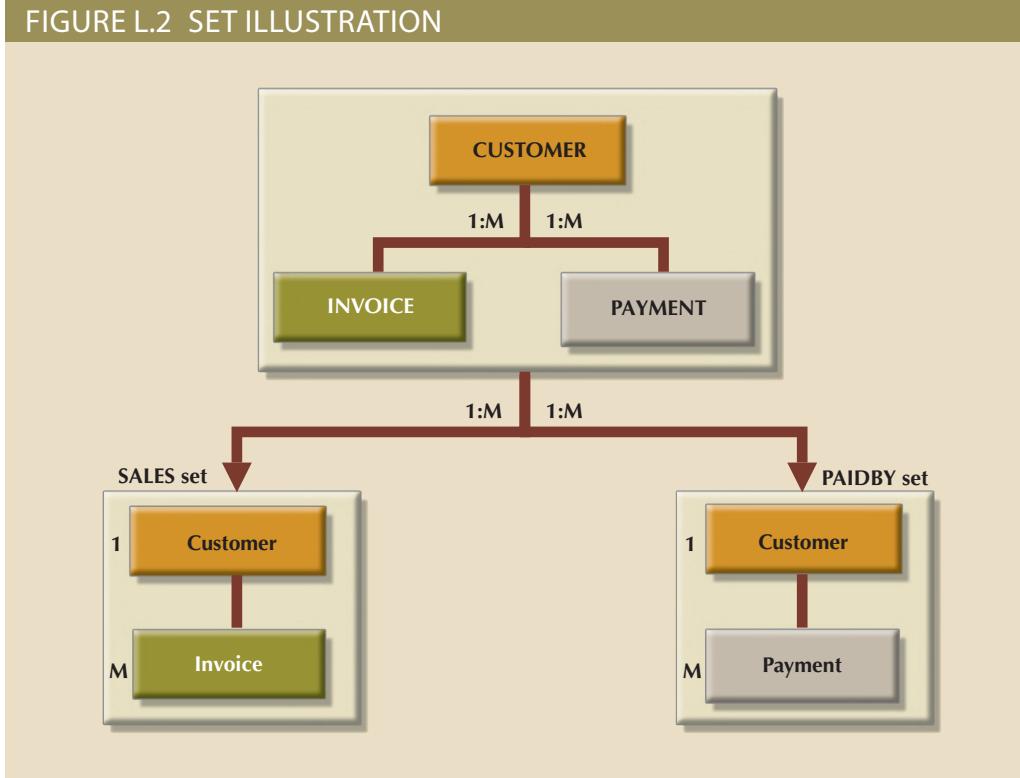


TABLE L.1

A TABLE OF SETS FOR THE NETWORK MODEL SHOWN IN FIGURE L.2

SET NAME	OWNER	MEMBER
PAIDBY	CUSTOMER	PAYMENT
SALES	CUSTOMER	INVOICE
INVLIN	INVOICE	INVLIN
COMMISSION	SALESREP	INVOICE
PRODSOLD	PRODUCT	INVLIN

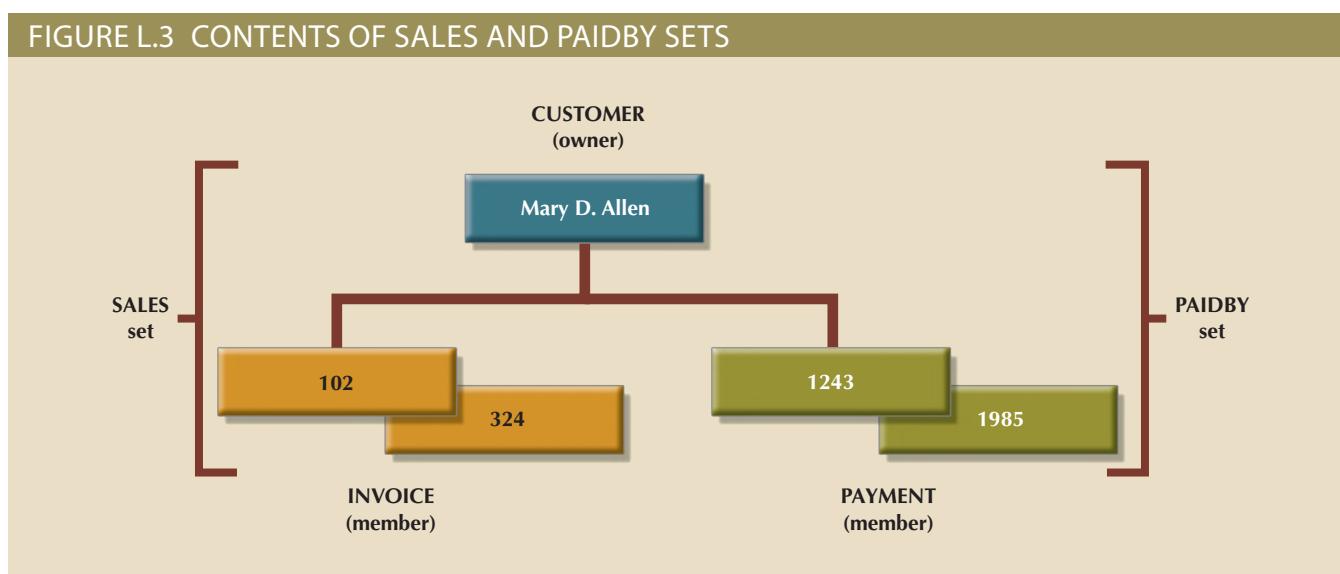
Figure L.3 depicts the contents of the member records for the PAIDBY set and the SALES set. The Mary D. Allen owner record has been used to illustrate how the data fit into the network structure. As you examine Figure L.3, note that the CUSTOMER named Mary D. Allen is the owner of two sets, SALES and PAIDBY. Mary D. Allen is also the owner of two INVOICE records, Invoices 102 and 324, which are members of the SALES set. She is also the owner of two PAYMENT records, Payments 1243 and 1985, which are members of the PAIDBY set. Given that structure, the user may navigate through any one of those two sets, using the data manipulation language (DML) provided by the DBMS.

Keep in mind that the INVOICE and PAYMENT records shown in Figure L.3 are related only to the CUSTOMER Mary D. Allen. When the user accesses another CUSTOMER record, a different series of INVOICE and PAYMENT records are available for that customer. Therefore, network database designers must also be aware of currency. The word **currency** indicates the position of the record pointer within the database and refers to the most recently accessed record.

currency

This term indicates the position of the record pointer within the database and refers to the most recently accessed record.

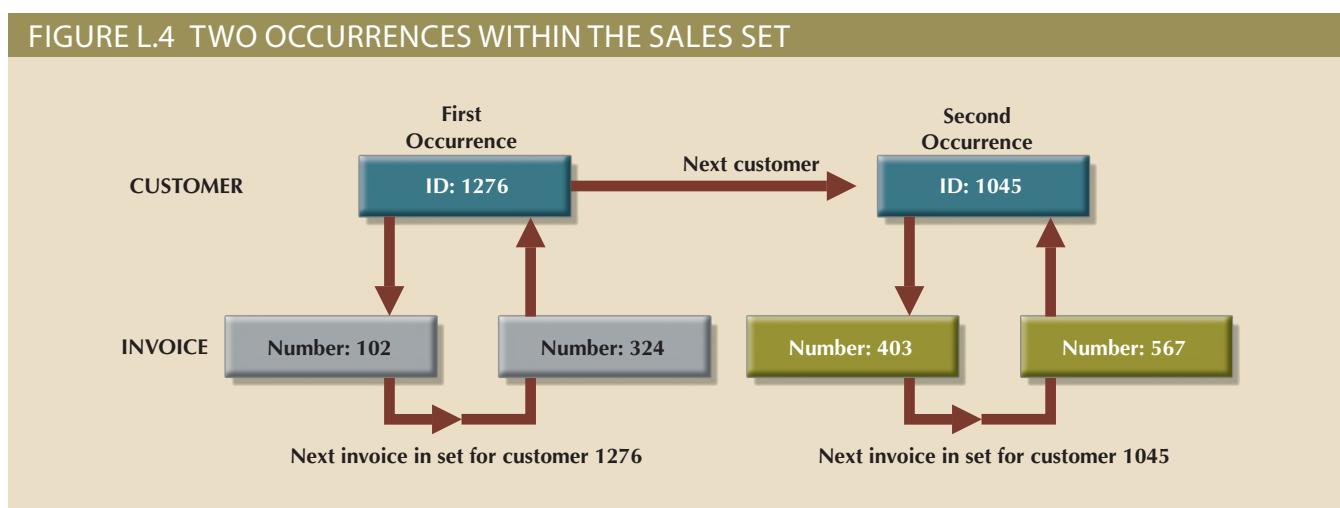
FIGURE L.3 CONTENTS OF SALES AND PAIDBY SETS



The DBMS automatically updates the pointers after the execution of each operation. A pointer exists for each record type in a set. A pointer's current value refers to a current record. Actually, two types of currency pointers exist: a record pointer and a set pointer. Record pointers, also known as record type pointers, exist for each record type within the database, *and they always point to the current record within each record type*. Because a set must contain two record types, an owner and a member, the set pointer points to either an owner record or a member record.

To illustrate the use of pointers, let's examine the condition shown in Figure L.4. Figure L.4 depicts two occurrences of the SALES set. The first occurrence corresponds to CUSTOMER number 1276. The second occurrence corresponds to CUSTOMER number 1045. Note that the occurrences are determined by the owner of the set: Every time you move to a new CUSTOMER record, a new group of INVOICE member records is made available.

FIGURE L.4 TWO OCCURRENCES WITHIN THE SALES SET



Given the SALES set components, the owner record is CUSTOMER and the member record is INVOICE. Therefore, the pointer locations for the current record of each record type and for the current record of a set *after the completion of each operation* will correspond to those found in Table L.2.

TABLE L.2

POINTER LOCATIONS WITHIN THE SALES SET

OPERATION	RECORD TYPE POINTERS		CURRENT RECORD OF THE SET
	CUSTOMER POINTER	INVOICE POINTER	SET POINTER
LOCATE CUSTOMER=1276 ^a	1276	NULL ^b	CUSTOMER RECORD (1276)
LOCATE FIRST IN SALES SET ^c	1276	102	INVOICE RECORD (102)
LOCATE NEXT IN SALES SET ^d	1276	324	INVOICE RECORD (324)
LOCATE INVOICE=403	1276	403	INVOICE RECORD (403)
LOCATE OWNER IN SALES SET	1045	403	CUSTOMER RECORD (1045)

^aThe summary in this table employs a pseudosyntax. For example, LOCATE CUSTOMER=1276 indicates a search for a CUSTOMER record whose customer number is 1276.

^bUsing network database jargon, NULL is used to indicate that no operation has been initiated yet and no pointer location has yet been designated.

^cLOCATE FIRST IN SALES SET means “Locate the *first member record in the current SALES set*.”

^dLOCATE NEXT IN SALES SET means “Locate the *next member record in the SALES set*.”

L-2 The Database Definition Language (DDL)

The database standardization efforts of the Data Base Task Group (DBTG) led to the development of standard **data definition language (DDL)** specifications. Those specifications include DDL instructions that are used to define a network database.

The examples of DDL will be based on Honeywell's Integrated Data Store/II (L-D-S/II) network database management system. Since L-D-S/II's DDL is very extensive, a subset of it will be used. (If you want to learn more about L-D-S/II, look at an L-D-S/II reference manual.) Don't despair if you don't have access to L-D-S/II; because the network model is based on CODASYL standards, database definition and creation are similar when other commercial applications are used.

Network database definition and creation are not interactive processes. Therefore, they must be done through the use of DBMS utility programs at the system prompt level. Creating an L-D-S/II database requires three steps: a logical definition of the database, using the DDL; a physical definition of the database, using the DMCL (device media control language); and the physical creation of the database storage files on secondary storage.

To see what general procedures are followed to design, create, and manipulate a network database, examine Figure L.5. (The illustration is based on CP-6 L-D-S/IL. CP-6 is a Honeywell operating system.)

The network database schema view or **schema** describes the entire database as seen by the database administrator. The schema defines the database name; the record type for each record; and the field, set, owner, and member records. The database subschema view, or *subschema*, describes the portion of the database used by each application program.

As you examine Figure L.5, note that the schema view and subschema view(s) are normal text files. Those schema and subschema text files may be created with any text-processor program. The files contain DDL and DMCL instructions, which describe the database and application views of the database and indicate what utility programs must be invoked to validate and create the database structure. Subschema views must be defined and validated for each application that uses the database.

L-D-S/II has an **Interactive Database Processor (IDP)**, which allows users to manipulate databases. The IDP front end is intended for users who have some programming knowledge; it is not well suited for most end users.

data definition language (DDL)

The language that allows a database administrator to define the database structure, schema, and subschema.

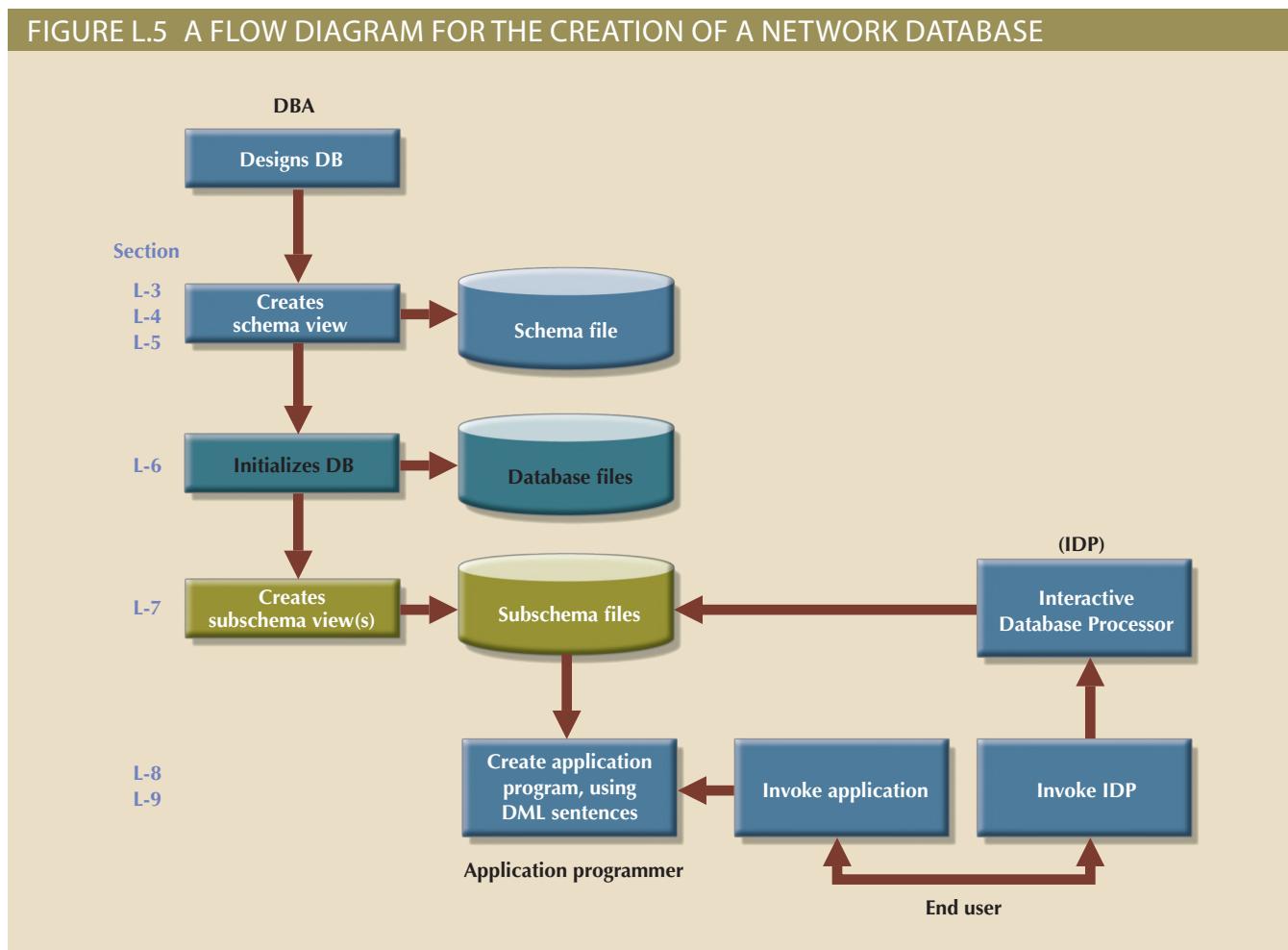
schema

A logical grouping of database objects, such as tables, indexes, views, and queries, that are related to each other. Usually, a schema belongs to a single user or application.

Interactive Database Processor (IDP)

In an IDS/II network DBMS, a processor that allows users to manipulate databases. The IDP front end is intended for users who have some programming knowledge and is not well-suited for most end users.

FIGURE L.5 A FLOW DIAGRAM FOR THE CREATION OF A NETWORK DATABASE



L-3 The Schema Definition

The first step in implementing a network database is defining the entire database *as seen by the database administrator* (DBA). The ROBCOR database will be used to illustrate the schema definition. The complete DDL description for the ROBCOR database is listed in Table L.3.

TABLE L.3

SCHEMA DEFINITION FOR THE ROBCOR DATABASE

LINE NUMBER	DLL CODE
1	DBACS TRANSLATE SCHEMA ROBCOR DDL END
2	SCHEMA NAME IS ROBCOR
3	AREA NAME IS MTSU
4	RECORD NAME IS CUSTOMER
5	LOCATION MODE IS CALC USING CUSID
6	DUPLICATES ARE NOT ALLOWED
7	WITHIN MTSU
8	02 CUSID TYPE IS CHARACTER 5
9	02 CUSTNAME TYPE IS CHARACTER 20

TABLE L.3

SCHEMA DEFINITION FOR THE ROBCOR DATABASE (CONTINUED)

LINE NUMBER	DLL CODE
10	02 CUSTADDRESS TYPE IS CHARACTER 35
11	RECORD NAME IS INVOICE
12	LOCATION MODE IS CALC USING INVNUM
13	DUPLICATES ARE NOT ALLOWED
14	WITHIN MTSU
15	02 INVNUM TYPE IS DECIMAL 5
16	02 INVDATE TYPE IS DECIMAL 8
17	02 INVAMOUNT TYPE IS DECIMAL 6,2
18	RECORD NAME IS INVLINE
19	LOCATION MODE IS VIA INVLINE SET
20	WITHIN MTSU
21	02 LIPRO TYPE IS CHARACTER 4
22	02 LIQTY TYPE IS DECIMAL 2
23	02 LIPRICE TYPE IS DECIMAL 4,2
24	RECORD NAME IS PAYMENT
25	LOCATION MODE IS CALC USING PAYNUM
26	SET DUPLICATES ARE NOT ALLOWED
27	WITHIN MTSU
28	02 PAYNUM TYPE IS DECIMAL 5
29	02 PAYDATE TYPE IS DECIMAL 8
30	02 PAYAMOUNT TYPE IS DECIMAL 6,2
31	RECORD NAME IS SALESREP
32	LOCATION MODE IS CALC USING SLSNUM
33	SET DUPLICATES ARE NOT ALLOWED
34	WITHIN MTSU
35	02 SLSNUM TYPE IS DECIMAL 5
36	02 SLSDATE TYPE IS DECIMAL 8
37	RECORD NAME IS PRODUCT
38	LOCATION MODE IS CALC USING PRODNUM
39	SET DUPLICATES ARE NOT ALLOWED
40	WITHIN MTSU
41	02 PRODNUM TYPE IS CHARACTER 4
42	02 PRODDATE TYPE IS DECIMAL 8
43	02 PRODQTY TYPE IS NUMERIC 6,2
44	SET NAME IS SALES
45	OWNER IS CUSTOMER
46	SET IS PRIOR PROCESSABLE
47	ORDER IS PERMANENT
48	INSERTION IS NEXT
49	MEMBER IS INVOICE
50	INSERTION IS AUTOMATIC
51	RETENTION IS MANDATORY
52	LINKED TO OWNER
53	SET SELECTION IS THRU SALES
54	OWNER IDENTIFIED BY APPLICATION
55	SET NAME IS PAIDBY

TABLE L.3

SCHEMA DEFINITION FOR THE ROBCOR DATABASE (CONTINUED)

LINE NUMBER	DLL CODE
56	OWNER IS CUSTOMER
57	SET IS PRIOR PROCESSABLE
58	ORDER IS PERMANENT
59	INSERTION IS NEXT
60	MEMBER IS PAYMENT
61	INSERTION IS AUTOMATIC
62	RETENTION IS MANDATORY
63	LINKED TO OWNER
64	SET SELECTION IS THRU PAIDBY
65	OWNER IDENTIFIED BY APPLICATION
66	SET NAME IS INVLINE
67	OWNER IS INVOICE
68	SET PRIOR PROCESSABLE
69	ORDER IS PERMANENT
70	INSERTION IS NEXT
71	MEMBER IS INVLINE
72	INSERTION IS AUTOMATIC
73	RETENTION IS MANDATORY
74	LINKED TO OWNER
75	SET SELECTION THRU INVLINE
76	OWNER IDENTIFIED BY APPLICATION
77	SET NAME IS COMMISSION
78	OWNER IS SALESREP
79	SET PRIOR PROCESSABLE
80	ORDER IS PERMANENT
81	INSERTION IS NEXT
82	MEMBER IS INVOICE
83	INSERTION IS AUTOMATIC
84	RETENTION IS MANDATORY
85	LINKED TO OWNER
86	SET SELECTION IS THRU COMMISSION
87	OWNER IDENTIFIED BY APPLICATION
88	SET NAME IS PRODSOLD
89	OWNER IS PRODUCT
90	SET PRIOR PROCESSABLE
91	ORDER IS PERMANENT
92	INSERTION IS NEXT
93	MEMBER IS INVLINE
94	INSERTION IS AUTOMATIC
95	RETENTION IS MANDATORY
96	LINKED TO OWNER
97	SET SELECTION IS THRU PRODSOLD
98	OWNER IDENTIFIED BY APPLICATION
99	END SCHEMA

To understand the DDL sequence shown in Table L.3, you must first learn how the database components work together. Keep in mind that a network database is basically a system driven by pointers. Think of a network database as a system having two components: the data and the pointer structures. The data (records with fields) are the raw facts kept in permanent storage devices for processing and retrieval. The pointers represent the structure that models the data and describes the required relationships (sets).

More precisely, the pointers define the way relationships are represented among entities. When an application program stores data in the network database, two different structures are updated: the data and the sets (pointers). Remembering that information will help you understand the DDL commands used to describe the database.

L-4 An Explanation of the Schema Definition

Table L.3's main schema definition components may be described this way:

- Line 1 invokes the **Database Administrator Control System (DBACS)**. The DBACS is the database definition processor that reads the ROBCOR database definition and validates the schema. (The DBACS works like a compiler.)
- Line 2 defines the schema name, which may be up to 30 characters long. Line 3 invokes the **AREA clause**. An **AREA** is a section of physical storage space that is reserved to store the database. The AREA clause allows the (physical) storage of a database in more than one location, thereby improving access speed. The area name must be unique, and there must be at least one area defined for the database.

L-4a Record Definitions

In L-D-S/II, you must first define all of the required record types. Table L.3 illustrates how that is done.

- Lines 4–10 in Table L.3 yield the CUSTOMER record definition. The **RECORD NAME clause** initiates the record's definition by assigning it a unique name. A valid schema must contain at least one record type.
- The **LOCATION MODE clause** in Table L.3 determines where the record will be (physically) stored in the database and how the record will be retrieved. Four location modes are supported under L-D-S/II: DIRECT, CALC, VIA SET, and INDEXED.
 1. DIRECT is the fastest location mode and requires that the application program assign a unique numeric key for each record. Using the DIRECT approach, the user exercises direct control over the arrangement of the records in the database. The DIRECT location mode allows the application program (programmer) to exercise the greatest degree of control over the location and retrieval of records from the database.
 2. CALC mode in Table L.3 uses a hashing algorithm over a record's field to generate the database key for each record. The same algorithm is used to retrieve the records. The DBA indicates the field over which L-D-S/II will apply the algorithm in the schema definition, text. This method yields a uniform dispersion of the records across the database. Note the use of CALC in lines 5 and 6 of the database schema definition and observe that line 6 specifies that DUPLICATES of key values are not allowed for this record type.
 3. VIA ... SET places member records together near the owner record occurrence in a set. The VIA ... SET approach is particularly useful when member records will be accessed sequentially. Note especially line 19 in Table L.3's database schema

Database Administrator Control System (DBACS)

In the network model, the database definition processor that reads the database definition and validates the schema (The DBACS works like a compiler.).

AREA

A named section of permanent storage space that is reserved to store the database.

RECORD NAME clause

In the IDS/II network DBMS, this clause initiates the record's definition by assigning it a unique name. An IDS/II network database must contain at least one record type.

LOCATION MODE clause

In an IDS/II network DBMS, this clause determines where a record will be (physically) stored in the database and how the record will be retrieved.

definition: To store an INVLINE occurrence, the INVOICE record must first be stored; then the INVLINE records will be stored around it.

4. INDEXED defines an independent storage structure. Indexed records do not participate in any sets. Instead, indexes are stored in an independent file. A unique primary index is created over a record's field. The index represents the order in which the records are stored in the database. There can be one primary key and several secondary keys for each record. Note the example in Table L.4.

TABLE L.4

DEFINING THE INDEX FILE

LINE	CODE	COMMENTS
1	RECORD NAME IS JOB_HISTORY	
2	LOCATION MODE IS INDEXED USING EMP_NUM	Record field is EMP_NUM. Primary key is EMP_NUM.
3	WITHIN EMP_HISTORY	Area name is EMP_HISTORY.
4	KEY NAME IS EMP_NUM	
5	ASCENDING EMP_SSNUM	
6	DUPLICATES ARE NOT ALLOWED	
7	KEY NAME IS JOB_DATE	Secondary key is JOB_DATE.
8	ASCENDING DATE_EMP	Record field is DATE_EMP.
9	DUPLICATES ARE NOT ALLOWED	
10	02 EMP_NUM TYPE IS DECIMAL 9	Record field
11	02 EMP_NAME TYPE IS CHARACTER	Record field
	02 DATE_EMP TYPE IS DECIMAL 8	Record field
	02 COMP_NAME TYPE IS CHARACTER	Record field
	02 JOB_SALARY TYPE IS DECIMAL 6	Record field

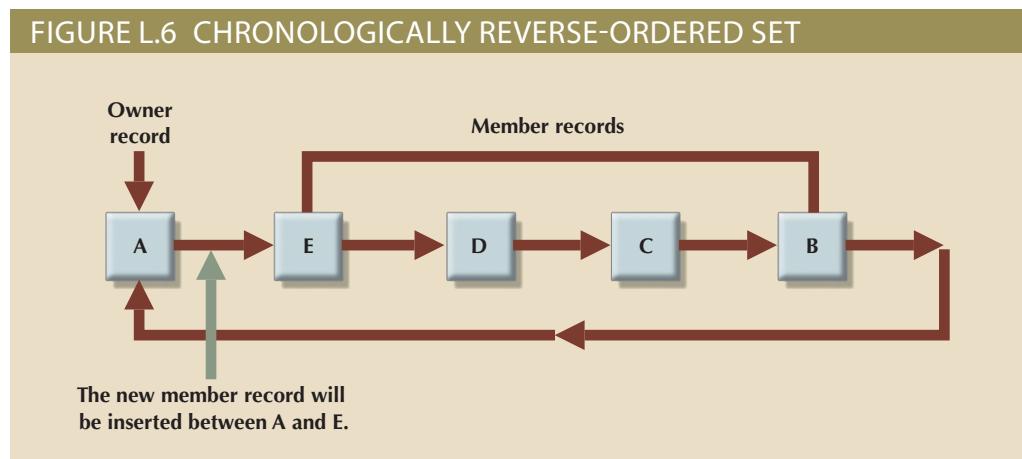
As you examine Table L.4, note these features:

- EMP_NUM represents the record's key field.
- There can be only one record type in an indexed area.
- Secondary keys may also be defined for the record.
- The DUPLICATES clauses determine whether the system will allow the use of duplicate primary key values. Because each CUSTOMER must have a unique customer identification number (CUSID) in the application, the DUPLICATES clause specifies that duplicates are not allowed.
- The WITHIN clauses specify in which area the records will be stored. In this case, all of the records that make up the ROBCOR database will be stored in the area named MTSU.
- The TYPE clauses allow you to define any of the following data types: fixed binary, float hexadecimal, decimal, character, database key, or unspecified (string). The database key will store the record key, and the unspecified data type is a string. (Note that field definitions in L-D-S/II resemble COBOL data definitions.) Using a COBOL-like syntax, you may define the name of the field, the TYPE of the field, and its length.

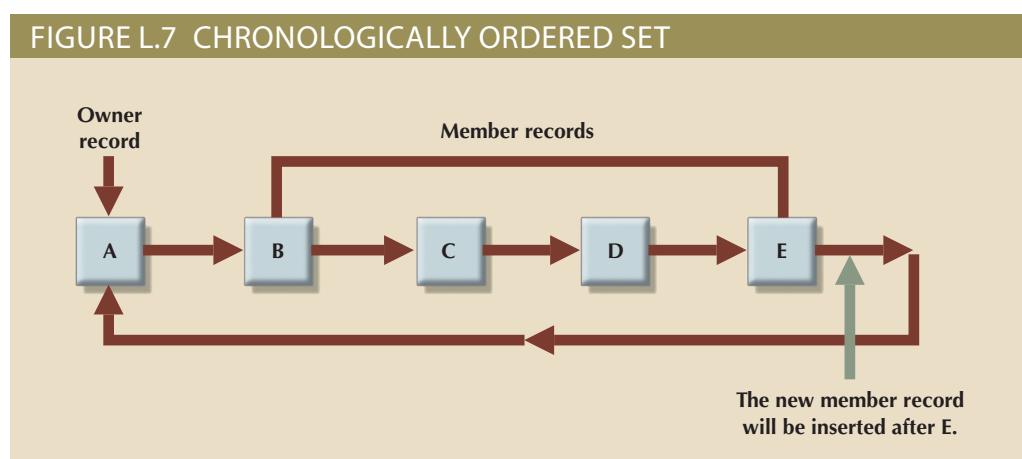
L-4b Set Definitions

After you have defined all of the records that make up your database, you must define the sets or relations among record types. Lines 44–54 in Table L.3 yield a set definition, as follows:

- Line 44 names the set. Note that the name must be unique within the current schema.
- Line 45 identifies the OWNER record type, which must be a valid record type already defined in the schema. A record can be an owner or a member of more than one set. However, a set may have only one OWNER. Line 46's SET IS PRIOR PROCESSABLE clause allows the L-D-S/II system to include a pointer to the previous record, thereby allowing efficient backward processing.
- The ORDER clause (lines 47 and 48) specifies where the record will be inserted within the set. The INSERTION clause can be FIRST, LAST, NEXT, or PRIOR.
- FIRST and LAST refer to the owner record. Specifically, FIRST defines the position directly *after* the owner record. The use of FIRST yields a chronologically reverse-ordered set, as shown in Figure L.6.



- LAST defines the position directly *before* the owner record in the set, yielding a chronologically ordered set shown in Figure L.7.



- NEXT and PRIOR in Table L.3 refer to the position relative to the current record of the set. The current record may be either the owner or a member of the set, whichever was last selected.
- The MEMBER clause in Table L.3 identifies the set's member record type. (The record must have been defined previously.) A given set may contain several member record types.
- The INSERTION and RETENTION clauses in Table L.3 define the way in which L-D-S/II associates the member records with their respective owner records. Valid parameters are shown in Table L.5.

TABLE L.5**VALID INSERTION AND RETENTION PARAMETERS**

CODE	COMMENT
INSERTION IS {AUTOMATIC}	The member record is automatically a member of the set when it is first stored.
INSERTION IS {MANUAL}	The record is not a member of any set when it is first stored. It can be related (manually) to a set later.
RETENTION IS {MANDATORY}	A member record should always belong to a set.
RETENTION IS {OPTIONAL}	The member record does not need to belong to a set.

- The INSERTION clause in Table L.3 specifies when a member record will be linked to an owner record.
- The RETENTION clause indicates whether a record should always belong to a specified set (mandatory) or if it can be in the database without belonging to any set (optional). The definition of the INSERTION and RETENTION parameters helps assure enforcement of the database's integrity.
- The LINKED TO OWNER clause in Table L.3 creates a pointer to the member record's owner. Such a link allows you to find the owner when the current record is a member record.
- The SET SELECTION clause in Table L.3 (lines 53 and 54) identifies how the current record of a set is selected prior to the record's retrieval or insertion. The application program will identify the owner record first, then make that record the current record before allowing the insertion or retrieval of any member record.

You can see that the network model uses several pointers to create the database's logical structure. For example, the database schema includes pointers to the next record, pointers to the prior record, pointers to the owner record, and so on. The degree of physical detail involved in the database definition is also very clear. As a result, learning the intricacies of such a database environment takes a considerable amount of time and effort.

Network database programmers must also be familiar with the available storage structures at the physical level. The DBACS not only validates and translates the schema specifications, but also defines and validates the database's physical storage characteristics. (The physical characteristics are defined using a device media control language.)

L-4c Device Media Control Language

After defining the database schema, the database administrator (DBA) must define the physical storage characteristics. For example, the system must "know" how the database will be stored on disk, what the area name is, and what records and sets belong to

the specified area. The ROBCOR schema's DMCL is depicted in Table L.6. The DBACS translates the schema and DMCL files to validate the physical structure that will support the database.¹

TABLE L.6

THE ROBCOR SCHEMA DMCL

LINE NUMBER	DLL CODE
1	DBACS TRANSLATE SCHEMA ROBCOR DCML
2	MODE IS ALTER
3	END
4	SCHEMA NAME IS ROBCOR
5	AREA NAME IS MTSU
6	ALLOCATE 512 DATA_BASE_KEYS
7	RECORD NAME IS CUSTOMER TYPE IS 1
8	RECORD NAME IS INVOICE TYPE IS 2
9	RECORD NAME IS PAYMENT TYPE IS 3
10	RECORD NAME IS INVLINE TYPE IS 4
11	RECORD NAME IS SALESREP TYPE IS 5
12	RECORD NAME IS PRODUCT TYPE IS 6
13	SET NAME IS SALES
14	SET NAME IS PAIDBY
15	SET NAME IS INVLINE
16	SET NAME IS COMMISSION
17	SET NAME IS PRODSOLD
18	END_DMCL

The DMCL file contains the following five components:

1. The schema name.
2. The area name and physical characteristics.
3. The record definitions.
4. The set entry.
5. The key entries used to name all of the record keys found in the area.

L-5 Database Initialization

After the schema DDL and DMCL have been validated by the DBACS, the database must be initialized using a utility program named DBUTIL. The database initialization process creates the physical files that will contain the database. The database files will be located in the physical storage devices identified in the AREA clause specified in the DDL and DMCL schema files.

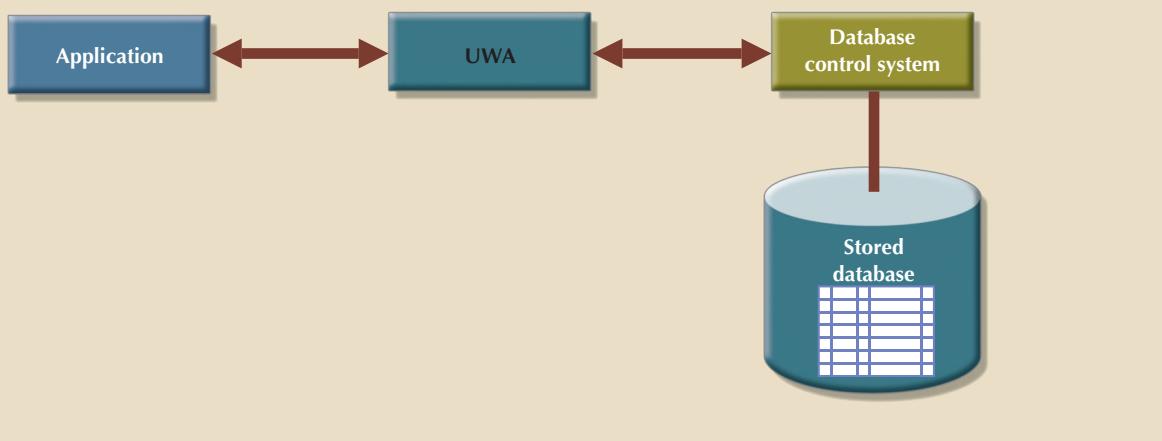
¹A complete description of all possible system options is available in Honeywell's *CP-6 L-D-S/II Database Administrator Manual*, order number CE36-02.

L-6 Subschema Definition

All applications programs view L-D-S/II databases through a *subschema*. The **subschema** contains all of the fields, records, and sets that are available to the application. In effect, the subschema is a “window” that the DBCS (Data Base Control System) opens to the application. The application uses this window to communicate with the database. Keep in mind that the subschema is contained within the database’s (total) conceptual schema. The DBCS validates all subschema entries against the schema. When an application program invokes a subschema, a User Work Area (UWA) is created by the DBCS. The **UWA** is a specific area of memory that contains several fields used to access and inform regarding the status of the database. The UWA also contains space for each record type defined in the subschema.

The UWA allows the application to communicate with the DBCS. Application programs read from and write to the UWA when the database is accessed or updated. Application programs can also check the database status after each operation to see if the operation was performed properly. The UWA’s role as the interface between an application and the database is illustrated in Figure L.8.

FIGURE L.8 THE UWA AS AN INTERFACE BETWEEN THE APPLICATION AND DATABASE



subschema

In the network model, the portion of the database “seen” by the application programs that produce the desired information from the data in the database.

UWA

User Work Area; a specific area of memory that contains several fields used to access and inform regarding the status of the database. The UWA also contains space for each record type defined in the subschema.

When an application retrieves a database record, the DBCS reads that record and places it in the space reserved for it by the application program’s UWA. The DBCS also updates all of the required UWA status fields. The application can also check and validate the database status after its last operation.

Subschemas are created manually by the DBA. In this case, the DBA must assure that all subschema definitions are correct and valid to the schema. A better way to create subschemas is to use the DBACS to create a full subschema from the main schema. That subschema will allow an unconstrained manipulation of the entire database. This all-encompassing subschema can then be modified by erasing all of the fields, records, and relations not required by the application program. L-D-S/II can generate subschemas for APL, BASIC, COBOL, and FORTRAN.

Table L.7 shows a COBOL subschema definition for the ROBCOR database. If this subschema is used, all sets and records of the database are available to the application program.

TABLE L.7

COBOL SUBSCHEMA DEFINITION FOR THE ROBCOR DATABASE

LINE NUMBER	DLL CODE
1	DBACS TRANSLATE COBOL SUBSCHEMA SUB_ROBCOR DDL END
2	TITLE DIVISION
3	SS SUB_ROBCOR WITHIN ROBCOR
4	MAPPING DIVISION
5	STRUCTURE DIVISION
6	REALM SECTION
7	RD MTSU
8	SET SECTION
9	SD COMMISSION
10	SD INVLINE
11	SD PAIDBY
12	SD PRODSOLD
13	SD SALES
14	KEY SECTION
15	RECORD SECTION
16	01 CUSTOMER
17	02 CUSTID DISPLAY PIC X(5)
18	02 CUSTNAME DISPLAY PIC X(20)
19	02 CUSTADDRESS DISPLAY PIC X(35)
20	01 INVLINE
21	02 LINEPROD DISPLAY PIC X(4)
22	02 LINEQTY DISPLAY PIC 9(2)V9(2)
23	02 LINEPRICE DISPLAY PIC 9(2)V9(2)
24	01 INVOICE
25	02 INVNUM DISPLAY PIC 9(5)
26	02 INVDATE DISPLAY PIC 9(8)
27	02 INVAMOUNT DISPLAY PIC 9(4)V9(2)
28	01 PAYMENT
29	02 PAYNUM DISPLAY PIC 9(5)
30	02 PAYDATE DISPLAY PIC 9(8)
31	02 PAYAMOUNT DISPLAY PIC 9(4)V9(2)
32	01 PRODUCT
33	02 PRODNUM DISPLAY PIC X(4)
34	02 PRODNAME DISPLAY PIC X(20)
35	02 PRODQTY DISPLAY PIC 9(4)V9(2)
36	01 SALESREP
37	02 SLSNUM DISPLAY PIC X(4)
38	02 SLSNAME DISPLAY PIC X(20)
39	END

Note that the subschema defined in Table L.7 contains the following three main components:

1. The *Title Division*, containing schema and subschema names.
2. The *Mapping Division*, containing all aliases used in the subschema.
3. The *Structure Division*, in which the area, sets, keys, and records used by the application are defined.

Given the components of the subschema depicted in Table L.7, note that a database file is referenced by its *realm*, rather than by its *area*, as is done in the schema definition. Actually, *realm* and *area* refer to the same thing. (The use of different terms to describe the same thing reflects the early lack of database terminology standards.)

The DBACS processes and validates the subschema definition in two independent steps. Those steps must be completed before any application programs based on the subschema can be compiled. (Keep in mind that the ROBCOR schema DDL and DMCL must be processed before the subschema can be processed.)

Figure L.9 shows the arrangement of the interactions between the DBMS and the ROBCOR information system components.

As you examine Figure L.9, note that each application program is given a UWA to manipulate the subset of the database needed by the application. The UWA is created at run time and uses the application's subschema definition data. Also remember that each subschema is a subset of the global database schema and must, therefore, be validated against the global schema. The DBMS (the DBCS component in L-D-S/II) is responsible for coordinating and controlling all of the interactions between the application programs, the user work areas, and the database.

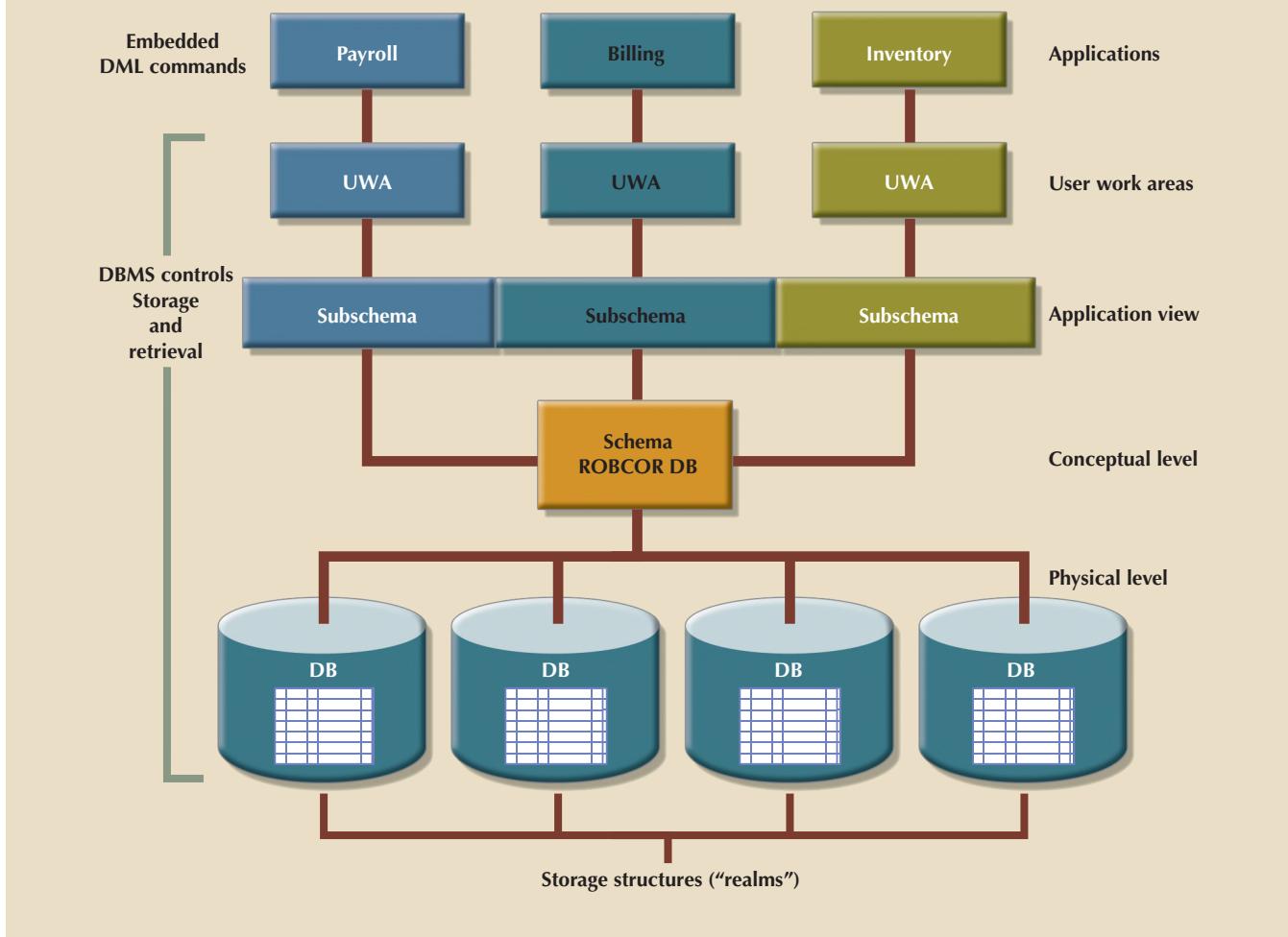
L-7 An Introduction to the Data Manipulation Language

Application programs navigate in a database by using a *data manipulation language (DML)*. As previously noted, L-D-S/II provides interfaces to four languages: APL, BASIC, COBOL, and FORTRAN. A COBOL-like syntax will be used to illustrate the DML's use.

The UWA has eight special status registers. The registers are used by the DBCS and the application program to share information about the status of the database. Here is a list of the registers' names (for COBOL) and an explanation of each:

1. *DB-STATUS*. This register returns the status of the DML statement after its execution. If no error occurs, the DB-STATUS returns zero. For example:
IF DB-STATUS = 00 Check for a “no error” condition
(COBOL SENTENCES)
...
...
2. *DB-REALM-NAME*. This register returns the name of the realm at the conclusion of DML sentences. Whether the DML's completion is successful or unsuccessful, the realm name is updated. The realm name can be blank. A COBOL program can check the value of this register.
3. *DB-SET-NAME*. This register returns the set name after an unsuccessful DML statement. (It can be blank.) A COBOL program can check the status of this register, but only the DBCS can update it.

FIGURE L.9 INTERACTION BETWEEN THE DBMS AND ROBCOR INFORMATION SYSTEM COMPONENTS



4. **DB-RECORD.** This register returns the record name at the conclusion of DML statements. Whether or not the statement was successful, the DB-RECORD is updated by the DBCS after each statement. DB-RECORD can be set to blank. A COBOL program can check its value.
5. **DB-PRIVACY-KEY.** The DBCS places the value of the PRIVACY key in this register during the schema translation. The PRIVACY key is a keyword used to restrict access to the database's authorized users.
6. **DIRECT-REFERENCE.** This register passes on a record key for DIRECT access. A COBOL command can update this register. The DBCS updates this register with the value of the key for the current record in the last DML statement.
7. **DB-DATA-NAME.** If the subschema was translated with the INCLUDE DATA NAMES option, the DBCS returns the DATA-ITEM name when an invalid data type problem occurs.
8. **DB-KEY-NAME.** The DBCS returns the name of the record key at the conclusion of an unsuccessful DML statement.

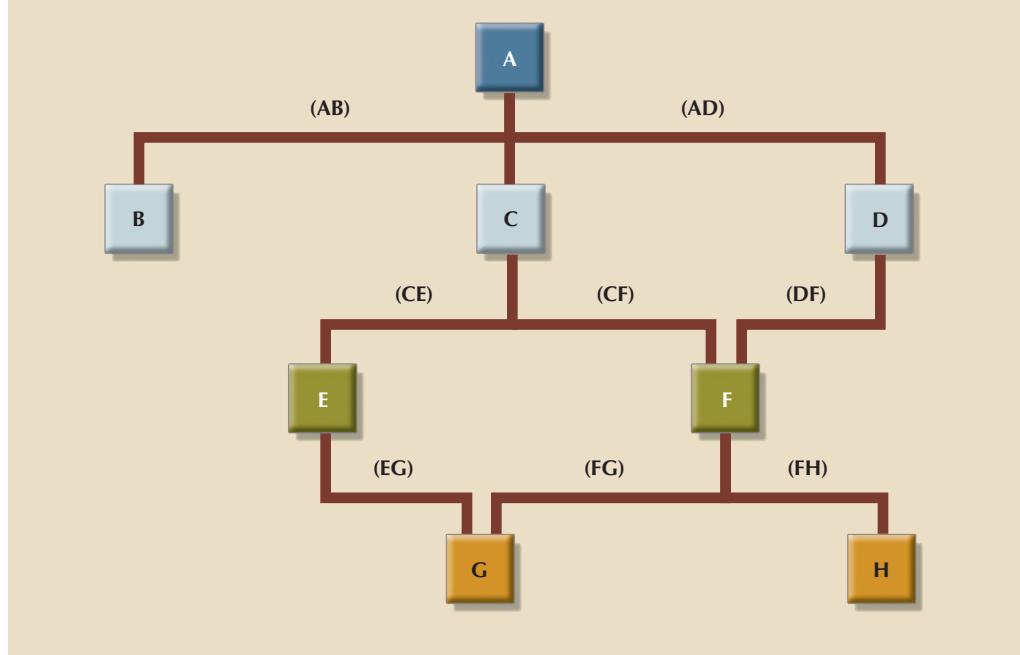
To understand the DML statements, it is important that you note the *currency* concept described in Section L-1. So you can appreciate the role played by currency, let's examine an instance of sequential file processing (see Figure L.10) with only one record type.

FIGURE L.10 AN INSTANCE OF SEQUENTIAL FILE PROCESSING WITH ONLY ONE RECORD TYPE



When working with a single record type, there is only one logical path for each record occurrence. If the pointer is located in record A, the next record will always be B, the record after that will always be C, and so on. However, when there are several record types in a network database environment, each record can be involved in more than one relation. That is, a record can be an owner or a member of more than one set. That condition is illustrated in Figure L.11.

FIGURE L.11 SEVERAL RECORD TYPES IN MULTIPLE RELATIONS



In Figure L.11, several logical paths may be taken to navigate from one record to another. For example, if the record pointer is in record A, the path may lead to B (AB set), C (AC set), or D (AD set). If the record pointer is in record F, the path may lead to D (DF set) or C (CF set), or you may elect to move to C (CF set) or H (FH set).

To keep track of the record and set pointers, L-D-S/II keeps five currency register records in the UWA. The currency registers are:

1. *Current record of the run unit.* A pointer updated by the DBCS after certain DML statements. This is the pointer to the last valid record accessed by the application.
2. *Current record of a set type.* A pointer for each set defined in the subschema. Such pointers specify the last record in each set that was accessed by the application.
3. *Current record of a realm.* A pointer for each realm specified in the subschema. Remember that a database can be stored in one or more areas or realms (physical files).

4. *Current record of a record type.* A pointer for each of the subschema's record types.
5. *Current record of a record key type.* A pointer for each record key type defined in the subschema. Each record key type points to a specific record. DBCS keeps a pointer to the last record accessed for each of the defined record key types.

L-8 Data Manipulation Language Commands

The data in a database (especially a transaction-type database) are subject to change. Therefore, end users must be able to add, delete, and modify the data. You will examine how data manipulation is done in a network database environment.

L-8a Opening Realms

An application program must invoke the READY command to access a database. The READY command makes the database available to the program. The three following usage modes are available:

1. UPDATE—Read/write to the database.
2. LOAD—Initial load of the database.
3. RETRIEVAL—Read from the database.

The command syntax conforms to the following sequence:

READY <real name>; USAGE IS {LOAD, UPDATE, RETRIEVAL}

L-8b Closing Realms

When the database's use is no longer required, close the realm using the syntax:

FINISH (realm list)

The realm list refers to the realm names that make up a database. If no realm names are specified, all realms (databases) are closed.

L-8c STORE

The STORE command saves a database record and updates the current record of the run unit in the UWA. The appropriate syntax is:

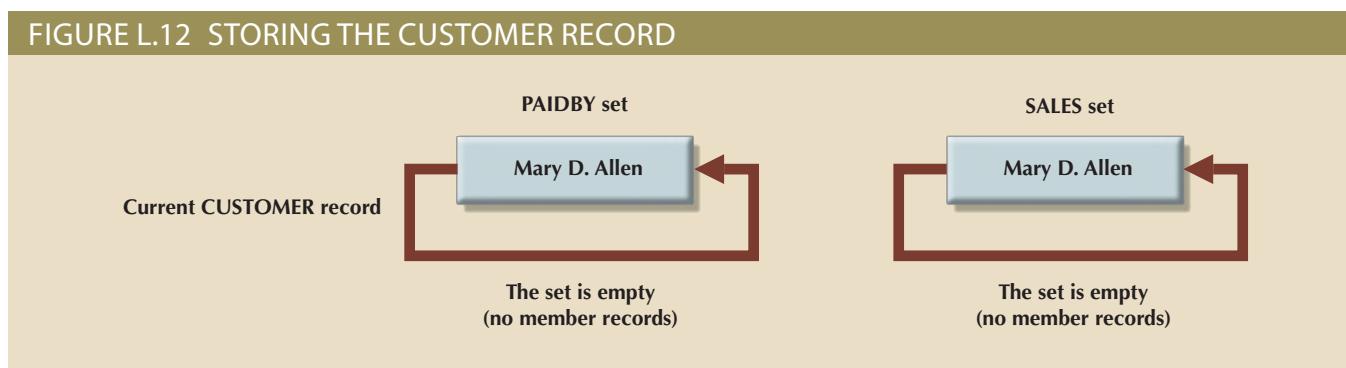
STORE (record-name)

To store a CUSTOMER record in the ROBCOR database, first move the new values to the corresponding fields:

```
MOVE "12421" TO CUSTID
MOVE "Mary D. Allen" TO CUSTNAME
MOVE "1418 E. Main Street" TO CUSTADDRESS
STORE CUSTOMER
```

Given that command sequence, the record will become the current record of the run unit, the current record of the PAIDBY and SALES sets, the current record of the realm, and the current record for the CUSTOMER record type, as shown in Figure L.12.

FIGURE L.12 STORING THE CUSTOMER RECORD



The order in which the records are stored is very important. Member records can be stored only after the record owner of the set has been stored. When a member record is stored, it is automatically inserted in all of the sets where the record was declared a member—if the *INSERTION IS AUTOMATIC clause was specified in the schema definition*. For example, a PAYMENT record may be inserted in the ROBCOR database by using:

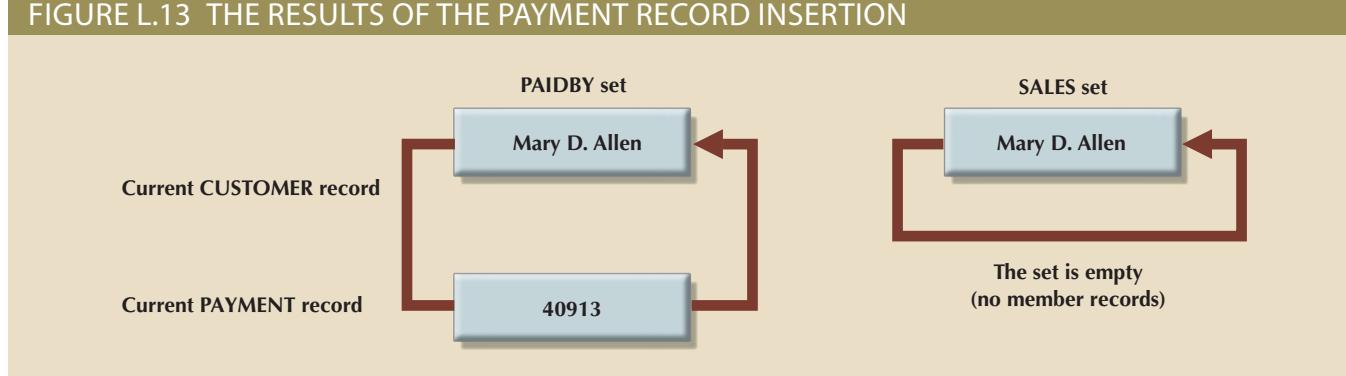
```
MOVE 40913 TO PAYNUM
MOVE "20181029" TO PAYDATE
MOVE 123.00 TO PAYAMOUNT
STORE PAYMENT
```

In that case, the PAYMENT record is inserted into the database; it is also automatically inserted into the PAIDBY set and linked to the CUSTOMER Mary D. Allen CUSTOMER record. The new values of the sets are indicated in Figure L.13.

If a record belongs to more than one set, the programmer must make sure that the current occurrences of the owner records (for the sets to which the new record belongs) are correct before the record is inserted. For example, if the INVOICE record belongs to two sets (SALES and COMMISSIONS), the SALESREP record must be stored prior to the storage of the INVOICE record:

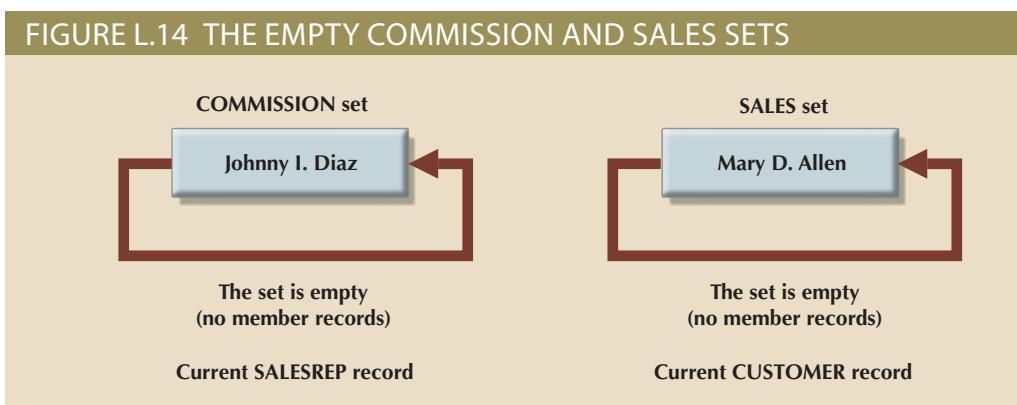
```
MOVE "D234" TO SLSNUM
MOVE "Johnny I. Diaz"
STORE SALESREP
```

FIGURE L.13 THE RESULTS OF THE PAYMENT RECORD INSERTION



After the insertion, the sets are represented as shown in Figure L.14.

FIGURE L.14 THE EMPTY COMMISSION AND SALES SETS

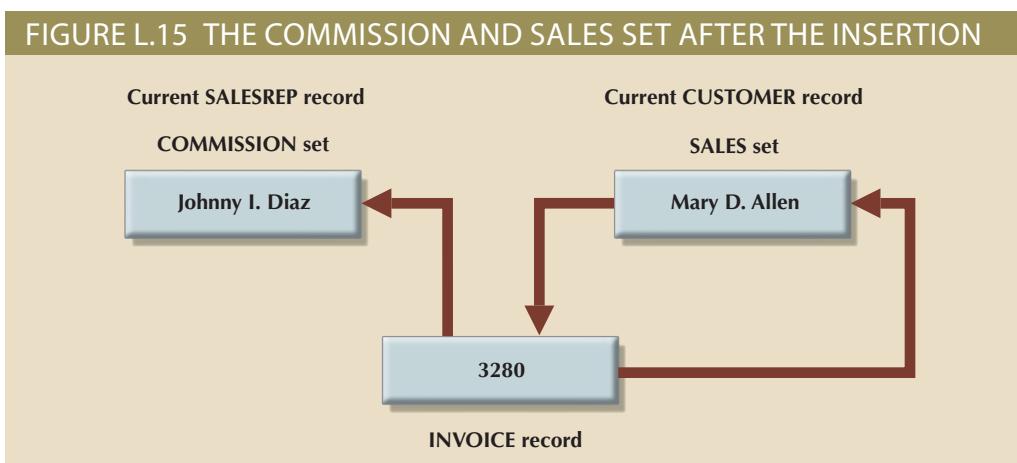


Next, the INVOICE record can be stored:

```
MOVE 3280 TO INVNUM
MOVE "20180114" TO INVDATE
MOVE 169.50 TO INVAMOUNT
STORE INVOICE
```

After that insertion, the sets look like Figure L.15.

FIGURE L.15 THE COMMISSION AND SALES SET AFTER THE INSERTION



L-8d FIND

The FIND command is used to locate records in the database and works with the LOCATION MODE used in the schema definition. The FIND command updates the currency values of the UWA. The syntax for the FIND command varies according to the access type.

Direct Access Mode

The command syntax for the direct access mode is:

```
FIND (record name); DB-KEY IS (dbkey)
```

The dbkey is the DIRECT-REFERENCE special field in the UWA. The direct-reference value must be moved to the DIRECT-REFERENCE field before the record can be accessed. For example:

```
MOVE "101" TO DIRECT-REFERENCE
FIND CUSTOMER; DB-KEY IS DIRECT-REFERENCE
```

The CUSTOMER record type must have been described as LOCATION MODE IS DIRECT in the schema definition.

CALC Access Mode

There are two options. The command syntax for each option follows:

```
FIND ANY <record name>
```

FIND DUPLICATE <record name>

Either FIND locates a record with LOCATION MODE IS CALC. The value to the key field is given before issuing the command:

```
MOVE "12421" TO CUSID
FIND ANY CUSTOMER RECORD
```

This command sequence locates the CUSTOMER record “Mary D. Allen.”

To use the CALC access mode, the CUSTOMER record must have been defined in the ROBCOR schema as LOCATION MODE CALC USING CUSID. That means that the contents of CUSID are used to find the record.

To store the occurrence of a PAYMENT record in the PAIDBY set, the commands shown in Table L.8 are necessary.

TABLE L.8

STORE THE OCCURRENCE OF A PAYMENT RECORD IN THE PAIDBY SET

PSEUDOCODE	COMMENT
MOVE "12421" TO CUSID	
FIND ANY CUSTOMER	Makes CUSTOMER the current record.
MOVE 40913 TO PAYNUM	
MOVE "20181029" TO PAYDATE	
MOVE 123.00 TO PAYAMOUNT	
STORE PAYMENT	

Navigating Within Sets

There are four options. The command syntax for each option follows:

Find PRIOR <record name> FIRST <set-name>

Find PRIOR <record name> NEXT <set-name>

Find PRIOR <record name> PRIOR <set-name>

Find PRIOR <record name> LAST <set-name>

As the syntax suggests, the FIND command locates the FIRST, NEXT, PRIOR, or LAST occurrence of a given record type within a set. The pseudocode in Table L.9 shows an example.

TABLE L.9

AN EXAMPLE OF THE FIND SYNTAX

PSEUDOCODE	COMMENT
MOVE "12421" TO CUSID	
FIND ANY CUSTOMER RECORD	Locates the owner of a set
IF DB-STATUS NOT = 00	
DISPLAY "CUSTOMER NOT FOUND"	
GO TO ERROR-RTN	
FIND FIRST INVOICE RECORD WITHIN SALES	Locates the first INVOICE in the SALES set for customer 12421
IF DB-STATUS NOT = 00	Check status
DISPLAY "ERROR"	
GOTO ERROR-RTN	

Locating Owner Records

To locate the owner of a member record in a set, use a modification of the FIND syntax:

FIND OWNER WITHIN <set-name>

An example of the command syntax is shown in Table L.10.

TABLE L.10

LOCATING OWNER RECORDS

PSEUDOCODE	COMMENT
MOVE "12421" TO CUSID	Makes CUSTOMER the current record
FIND ANY CUSTOMER	
IF DB-STATUS NOT = 00	Check status
DISPLAY "ERROR"	
GOTO ERROR-RTN	
FIND FIRST INVOICE RECORD WITHIN SALES	
IF DB-STATUS NOT = 00	Check status
DISPLAY "ERROR"	
GOTO "ERROR-RTN"	
FIND OWNER WITHIN COMMISSION	
IF DB-STATUS NOT = 00	Check status
DISPLAY "ERROR"	
GOTO ERROR-RTN	

The command sequence in Table L.10 is a good example of how the application program navigates the database. First, locate the CUSTOMER record 12421 for a customer named “Mary D. Allen.” Next, move into the SALES set to locate Mary D. Allen’s first INVOICE. Finally, the

FIND OWNER WITHIN COMMISSION

command locates the SALESREP record for that INVOICE.

L-8e CONNECT

The purpose of the CONNECT command is to insert an existing record as a set member. Both the member and the owner records must already be stored in the database. The command syntax is:

CONNECT <record-name> TO <set-name>

The CONNECT command is used when the INSERTION IS MANUAL and the OWNER IDENTIFIED BY APPLICATION clauses were specified for the member record in the schema definition. The user has to manually CONNECT the record with the appropriate owner record in each of the sets to which the record belongs. Assuming the PAYMENT record was defined as INSERTION IS MANUAL in the PAIDBY set, the correct sequence of commands is shown in Table L.11.

TABLE L.11

INSERTING AN EXISTING RECORD AS A SET MEMBER

PSEUDOCODE	COMMENT
MOVE "12421" TO PAYNUM	Makes CUSTOMER the current record
FIND ANY CUSTOMER	
MOVE "40913" TO PAYNUM	
MOVE "20181029" TO PAYDATE	
MOVE 123.00 TO PAYAMOUNT	
STORE PAYMENT	Stores PAYMENT
CONNECT PAYMENT TO PAIDBY	Inserts record in PAIDBY set

L-8f DISCONNECT

The DISCONNECT command removes a record from a set. The command is used only when records were declared as AUTOMATIC OPTIONAL or MANUAL OPTIONAL members of a set in the schema definition. The syntax is:

DISCONNECT <record-name> FROM <set-name>

Table L.12 shows an example.

TABLE L.12

REMOVE A RECORD FROM A SET

PSEUDOCODE	COMMENT
MOVE "40913" TO PAYNUM	Locates the PAYMENT record
FIND ANY PAYMENT	
DISCONNECT PAYMENT FROM PAIDBY	Disconnects the PAYMENT record from the PAIDBY set

Note that the DISCONNECT command in Table L.12 does not physically remove the record from the stored database; it merely manipulates the pointers to bypass the record.

L-8g GET

The GET command reads a record from the database, making the record's field available to the program. Only the fields defined in the subschema are available. The command syntax is:

GET <record-name>

Table L.13 shows an example.

TABLE L.13

READ A RECORD WITH GET

PSEUDOCODE	COMMENT
MOVE "12421" TO CUSTID	Locates the customer record
FIND ANY CUSTOMER	
GET CUSTOMER	Reads the customer's data

L-8h MODIFY

The MODIFY command changes the current record's field contents, using the syntax:

MODIFY <record-name>

The command flushes the contents of the UWA buffers to the database. An example of the MODIFY command is shown in Table L.14.

TABLE L.14

MODIFY A RECORD	
PSEUDOCODE	COMMENT
MOVE "12421" TO CUSTID	Locates the customer record
FIND ANY CUSTOMER	
GET CUSTOMER	Reads the customer record
MOVE "245 S.W. Clark St." TO CUSTADDRESS	Changes the address in the UWA buffer
MODIFY CUSTOMER	Writes the changes to the (physical) database

L-8i ERASE

The ERASE command removes the current record from the database *and automatically removes all member records associated with it*. The command syntax is:

ERASE <record-name> ALL MEMBERS

The ERASE command ensures that the record is eliminated from all of the sets in which it was declared a member. If the record was declared owner of one or more sets, all member occurrences related to the record are also removed. Table L.15 shows an example.

TABLE L.15

DELETE A RECORD	
PSEUDOCODE	COMMENT
MOVE "14206" TO INVNUM	
FIND ANY INVOICE	Locates the invoice
ERASE INVOICE ALL MEMBERS	Erases the INVOICE record and all member records of all sets from which INVOICE is the owner record

The command sequence shown in Table L.15 will erase the INVOICE records from:

1. All of the sets (COMMISSIONS and SALES) for which it was declared a member.
2. All of the INVLINE records associated with the INVOICE in the INVLINES set.
3. All of the INVLINE members associated with the PRODSOLD set.

L-9 The Network Model's Contribution to Database Systems

The network database model provided several advantages over its file-system and hierarchical-database predecessors. In fact, the network database model paved the way for subsequent database developments through CODASYL's attempt to standardize basic database concepts such as schema, subschema, and DML. The network database model also set the stage for more complex and better data modeling by providing support for relations in which a record could be related to more than one owner or parent record.

Key Terms

AREA, L-9	data definition language (DDL), L-5	RECORD NAME clause, L-9
currency, L-3	Interactive Database Processor (IDP), L-5	schema, L-5
Database Administrator Control System (DBACS), L-9	LOCATION MODE clause, L-9	subschema, L-14
		UWA (user work area), L-14

Appendix M

MS Access Tutorial

Preview

After a database has been designed properly it must be implemented and the applications that make the database useful to the end users must be developed. Microsoft Access is a great tool for prototyping a database's implementation and its applications. Therefore, we will show you how to:

- Create the database
- Create and modify tables
- Enter data into the tables
- Import tables
- Define the relationship(s) between the tables
- Create queries
- Create forms
- Create menus linked with macros
- Create reports

Data Files and Available Formats

MS Access **Oracle** **MS SQL** **My SQL**

MS Access **Oracle** **MS SQL** **My SQL**

Ch07_SaleCo



Data Files Available on cengagebrain.com



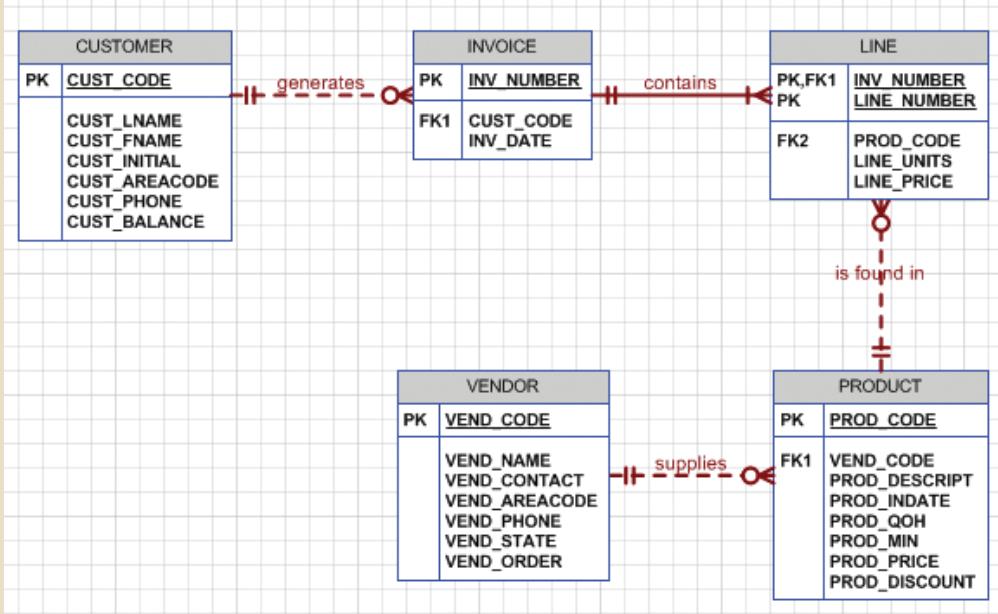
Note

This tutorial assumes that you know how to perform basic operations in the Microsoft Windows environment.

M-1 Create the Database

If you have studied our *Database Systems: Design, Implementation, and Management* text, you already know the important ground rule: The database design is the crucial first step in the journey that lets you successfully implement and manage a database. The database design shown in Figure M.1 in the form of an Entity Relationship Diagram (ERD) will be the basis for this tutorial.

FIGURE M.1 THE SALECO DATABASE ERD



Note

If necessary, review the text's Chapters 4 through 6 to ensure that you have a sound basis for database design work. You can study Appendix A1, Designing Databases with Visio Professional 2010: A Tutorial, if you want to create the database design shown in Figure M.1. Appendix A2 shows the steps in Visio 2013.

Given the database design shown in Figure M.1, you are ready to implement it in Microsoft Access. The first step is to start the MS Access DBMS software, which is part of the Microsoft Office Professional suite. Use the same routine you use to open all other Microsoft office components to open Microsoft Access.

After Access opens choose the **Blank desktop database** option.

- When you select the **Blank desktop database** option shown in Figure M.2, Figure M.3 will appear. Note the default folder and notice that the default database name is **Database1.accdb**.
- Select the folder in which you want to store the database, rename the database file to **SaleCo**, and click **Create** as shown in Figure M.4. This option will create a new local Access database.

FIGURE M.2 CREATE A NEW DATABASE

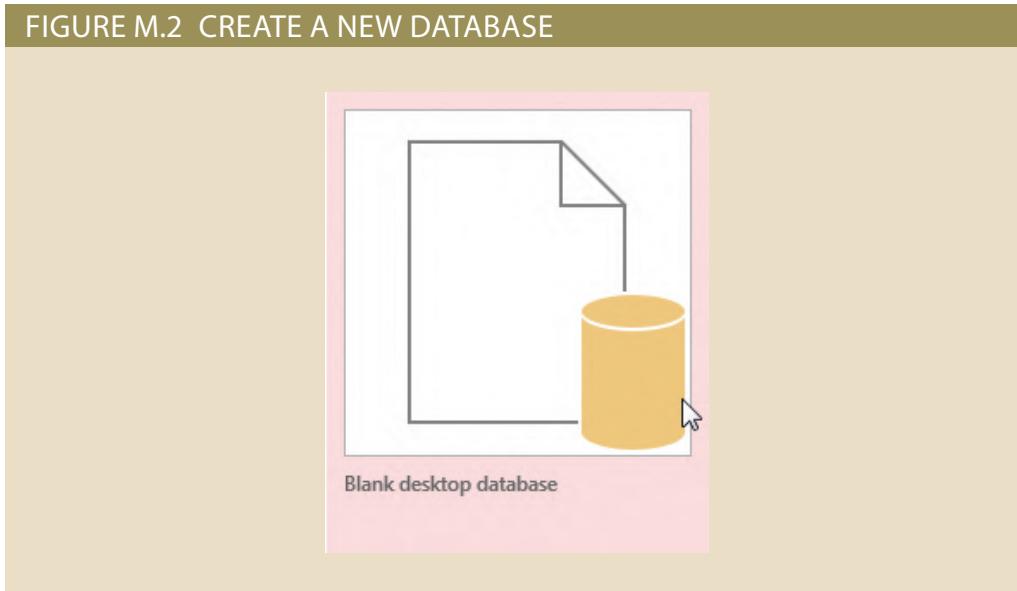


FIGURE M.3 DEFAULT FOLDER AND DATABASE NAME

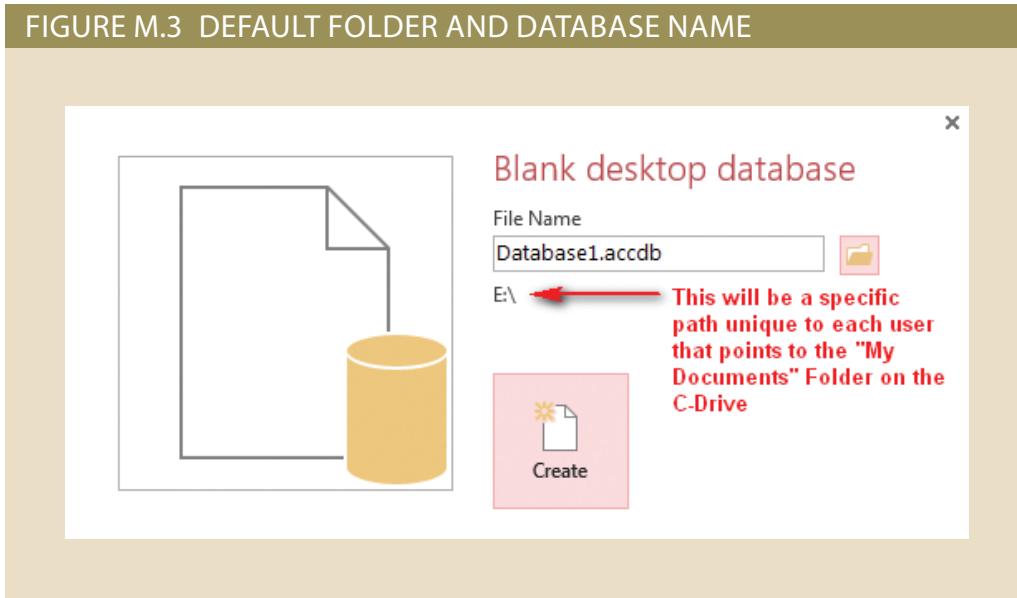
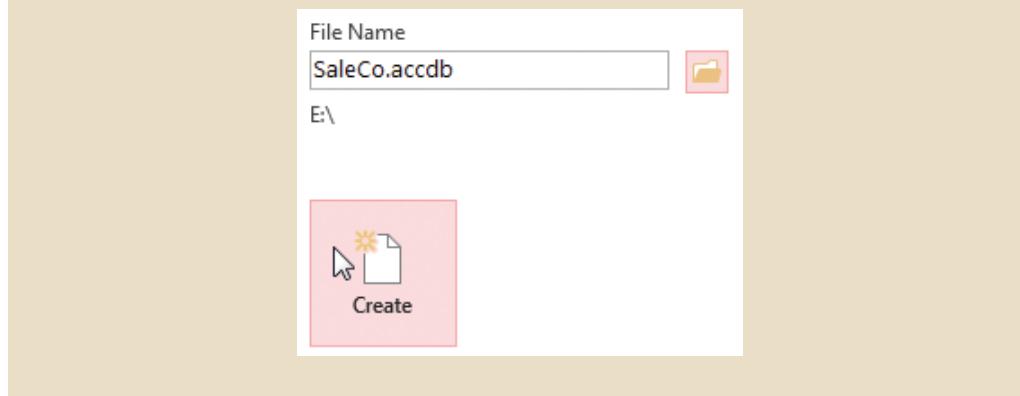
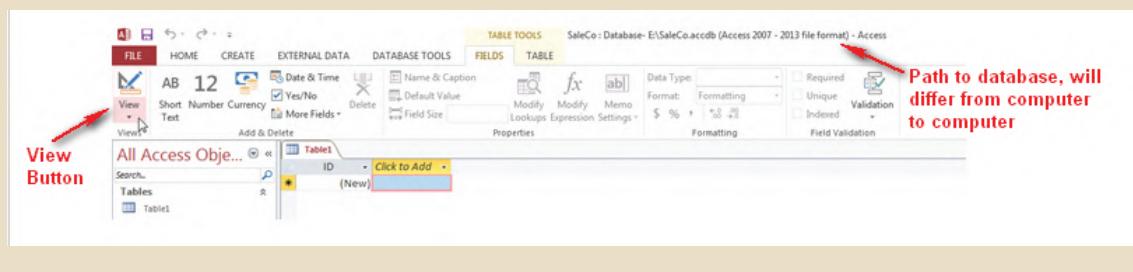


FIGURE M.4 SELECT FOLDER AND RENAME DATABASE



After following the steps outlined above, MS Access will open the **SaleCo** database as seen in Figure M.5. The next section teaches you how to create your first MS Access table.

FIGURE M.5 THE SALECO DATABASE

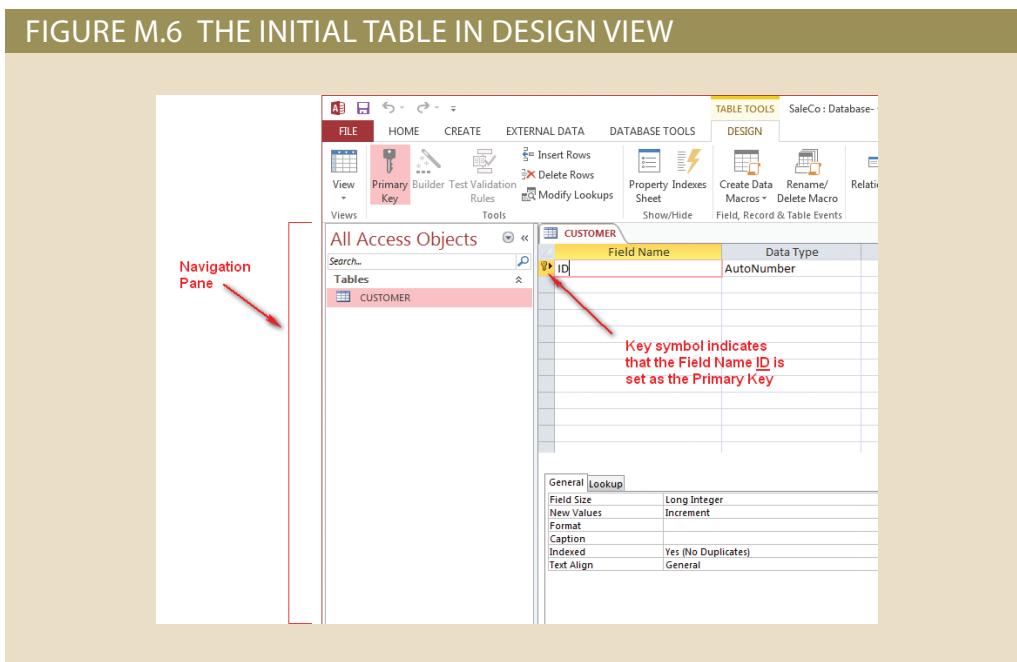


M-2 Creating the Table Structures

A new table named Table1 appears when the **SaleCo** database is generated.

- Click the **View** button as shown in Figure M.5 to switch to **Design View**. This button switches back and forth from Design View (edit mode) to Datasheet View (entry mode).
- Type **CUSTOMER** in the Save As dialog box and click **OK**.
- The table changes from Table 1 to CUSTOMER. You are now in Design View, which allows you to name and define the fields of the table.
- Access automatically creates a field named ID with an AutoNumber data type as seen in Figure M.6.

FIGURE M.6 THE INITIAL TABLE IN DESIGN VIEW

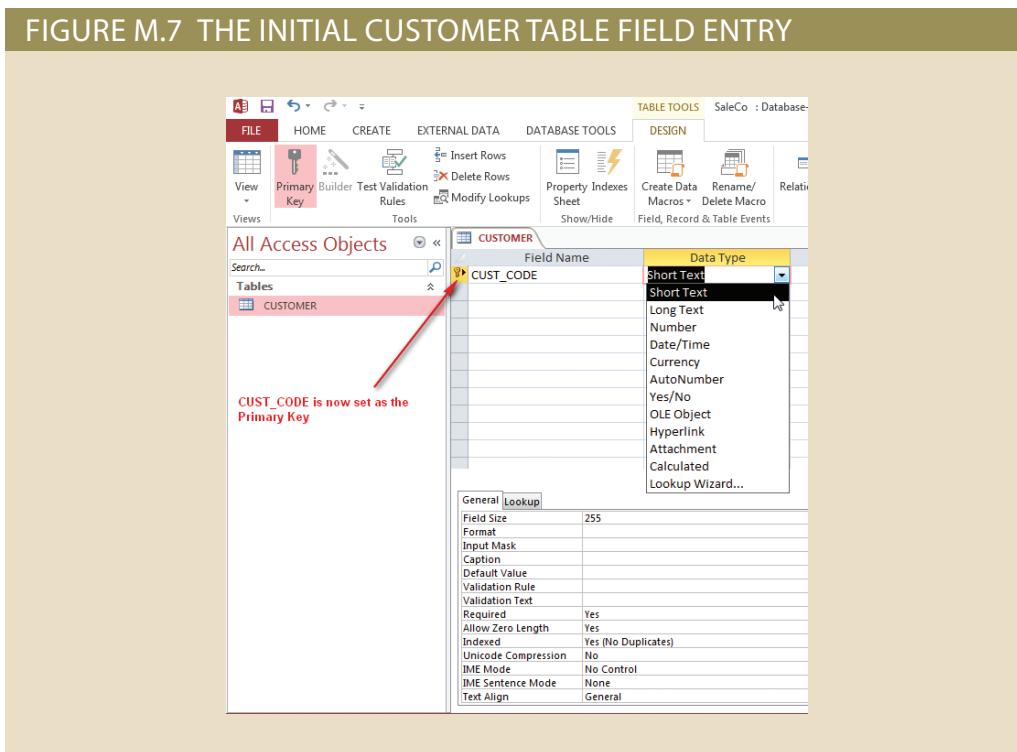


You are now ready to type in the first field name. (See Figure M.1 again to see what fields are included in the CUSTOMER table.)

Rename ID to **CUST_CODE** and change the data type to **Short Text** as seen in Figure M.7.

The CUST_CODE values will all consist of a five-character Short Text string, so the **Field Properties** box default **Field Size** of 255 should be reset to **5**. (Just select the “255” and type in the value “5” to get the job done.)

FIGURE M.7 THE INITIAL CUSTOMER TABLE FIELD ENTRY



- As you examine Figure M.7, note that the **Field Properties** box entry for the CUST_CODE indicates that the **Indexed** default value is set to **Yes (No Duplicates)**.
- Note that this property enforces entity integrity, because no duplicate values will be permitted. This is automatically set when the primary key (PK) is assigned.

Add an optional description explaining what the field is as seen in Figure M.8.

FIGURE M.8 OPTIONAL DESCRIPTION

The screenshot shows the 'Field Properties' dialog box for the 'CUS_CODE' field in 'Table1'. The 'Field Name' is 'CUS_CODE', 'Data Type' is 'Short Text', and the 'Description (Optional)' is 'Customer Code, Primary Key for the CUSTOMER Table'.

Field Name	Data Type	Description (Optional)
CUS_CODE	Short Text	Customer Code, Primary Key for the CUSTOMER Table

Using the procedures you have just learned, go ahead and create the remaining attributes for the **SaleCo** database's CUSTOMER table. (Use the ERD in Figure M.1 as your guide.) The completed table is shown in Figure M.9.

- Note that in Figure M.9 the CUST_BALANCE is a **Currency** data type and that the default value is set to 0. Therefore, the CUST_BALANCE value will be shown as 0.00.
- Make sure that you use a **Short Text** format for all of the remaining attributes.
- Limit the field lengths for the following attributes as follows: CUST_LNAME (**20**), CUST_FNAME (**20**), CUST_INITIAL (**1**), CUST_AREACODE (**3**), and CUST_PHONE(**8**).

Given the self-describing field names, there is little need to continue the entry of more detailed descriptions. For example, the CUST_LNAME is clearly a customer's last name, so no further description is necessary.

FIGURE M.9 THE COMPLETED CUSTOMER TABLE

The screenshot shows the 'Field Properties' dialog box for the 'CUST_CODE' field in the 'CUSTOMER' table. The 'Field Name' is 'CUST_CODE', 'Data Type' is 'Short Text', and the 'Description (Optional)' is 'Customer Code, Primary Key for the CUSTOMER Table'. A red arrow points to the 'Field Name' column with the text 'Enter these field names'. Another red arrow points to the 'Data Type' column with the text 'Enter these data types'. The 'Default Value' is set to '0'. A tooltip for 'Data Type' says: 'Type determines the kind of values can store in the field. Press F1 for help on data types.' The 'Format' dropdown is set to 'Currency'.

CUSTOMER		CUSTOMER table tab
Field Name	Data Type	Description (Optional)
CUST_CODE	Short Text	
CUST_LNAME	Short Text	
CUST_FNAME	Short Text	
CUST_INITIAL	Short Text	
CUST_AREACODE	Short Text	
CUST_PHONE	Short Text	
CUST_BALANCE	Currency	

General **Lookup**

Format	Currency
Decimal Places	Auto
Input Mask	
Caption	
Default Value	0
Validation Rule	
Validation Text	
Required	No
Indexed	No
Text Align	General

Enter these field names

Enter these data types

Type determines the kind of values can store in the field. Press F1 for help on data types.

A table can have many fields, each one with a specific data type. A data type determines the type of values that a field can have. MS Access supports various data types; for a complete description of such data types, see Table M.1.

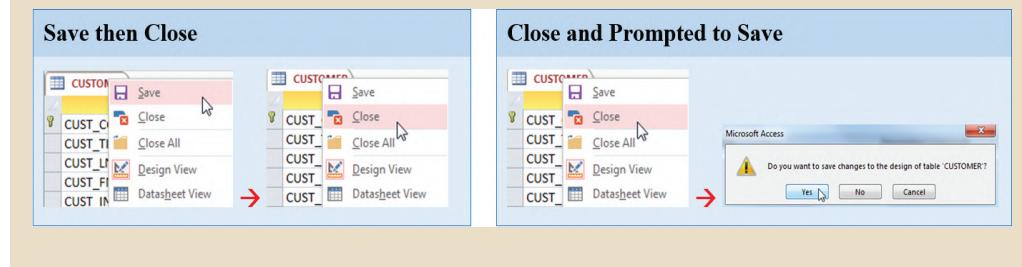
TABLE M.1

DATA TYPES

DATA TYPE	PROPERTIES
Short Text	Text or numbers not used in calculations with a maximum field size of 255
Long Text	Text with a field size greater than 255, commonly used for comments or notes
Number	Numeric data that can be used in calculations
Date/Time	Dates and times
Currency	Monetary values including symbols
AutoNumber	Sequential integers that are automatically generated by Access.
Yes/No	Either a Yes or No value (only 2 options)
OLE Object	Object Linking and Embedding (OLE), commonly used for Word documents and Excel files
Hyperlink	Link to web and email addresses
Attachment	External files, commonly used for image files, spreadsheets and documents
Calculated	A calculation resulting from other fields in the table
Lookup Wizard	Creates a lookup field, which displays a list of values you can choose from. Used to match the data types of other fields so that relationships can properly be created. The final data type is either Short Text or Number.

After defining all the CUSTOMER table fields, you are ready to save the table. Right-click the **CUSTOMER** table tab and click **Save** then right-click again and click **Close**. (Note: If you just close the table without saving first, Access will prompt you to save the table before closing. Click **Yes** to save changes. The two options are shown below in Figure M.10.)

FIGURE M.10 SAVE AND CLOSE THE CUSTOMER TABLE



As you saw in Figure M.6, to the left of the table tab there is an area known as the Navigation Pane. The Navigation Pane displays all the objects you have created in the database. There are four main object types in MS Access databases: Tables, Queries, Forms and Reports (see Table M.2). Double-clicking the object's icon in the Navigation Pane will open the specified object. The View button can then be used to change views.

TABLE M.2

OBJECTS AND THEIR FUNCTIONS

OBJECT	ICON	FUNCTION
Table		Stores all of the raw data in the database. Tables are linked with a common field (attribute) to reduce data.
Query		A specific request for data manipulation issued by the end user; a question or a task asked by the end user.
Form		Allows the end user to easily enter data.
Report		A printout of the data that can be customized to contain headers, footers, graphics, and calculations on groups.

M-2a Data Entry

In this section you will learn how to enter and edit data in a table. In the Navigation Pane, right-click the **CUSTOMER** table and click the **Open** option shown in Figure M.11 to generate the table datasheet view shown in Figure M.12. (You can also double-click the CUSTOMER table.)

FIGURE M.11 OPENING THE CUSTOMER TABLE FOR DATA ENTRY

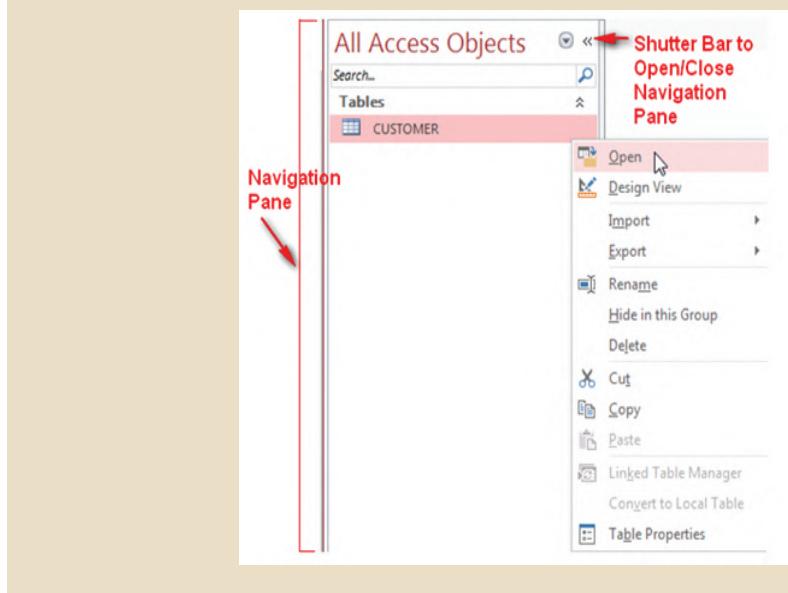


FIGURE M.12 THE CUSTOMER DATASHEET VIEW

CUST_CODE	CUST_LNAM	CUST_FNAM	CUST_INITI	CUST_AREA	CUST_PHON	CUST_BALAN
\$0.00						

Enter in the data shown in Figure M.13. Suppose you now try to enter the second record. After accepting the 0.00 CUST_BALANCE value, you tap the **Enter** key to move the cursor to the second record. Now type in the same **10010** CUST_CODE value you used for the first record as shown in Figure M.14. Then try to close the CUSTOMER table. Your reward will be the error message shown in Figure M.14, because you violated entity integrity requirements.

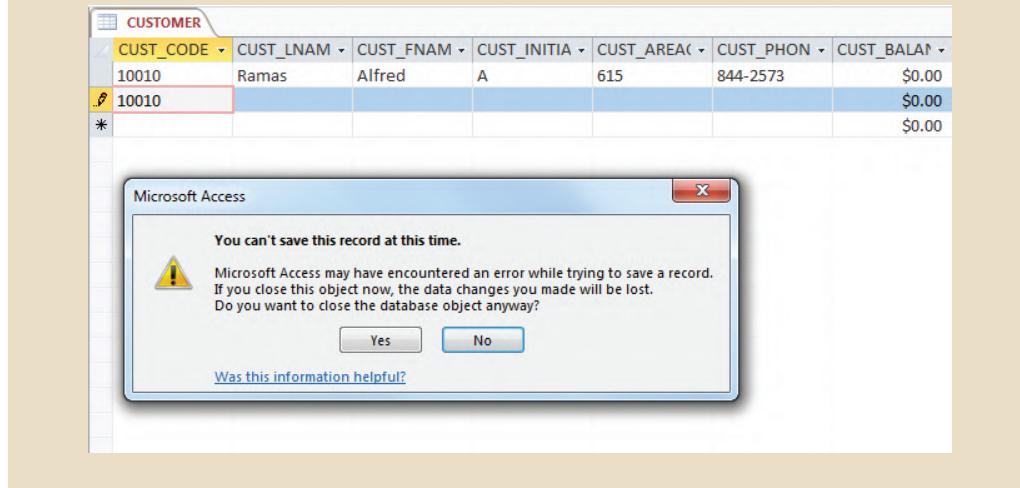
FIGURE M.13 ENTERING THE FIRST CUSTOMER RECORD

CUST_CODE	CUST_LNAM	CUST_FNAM	CUST_INITIA	CUST_AREA	CUST_PHON	CUST_BALAN
10010	Ramas	Alfred	A	615	844-2573	\$0.00

FIGURE M.14 ENFORCEMENT OF ENTITY INTEGRITY

Click **OK** to acknowledge the error message. This action will generate the result you see in Figure M.15.

FIGURE M.15 RECORD NOT SAVED WARNING



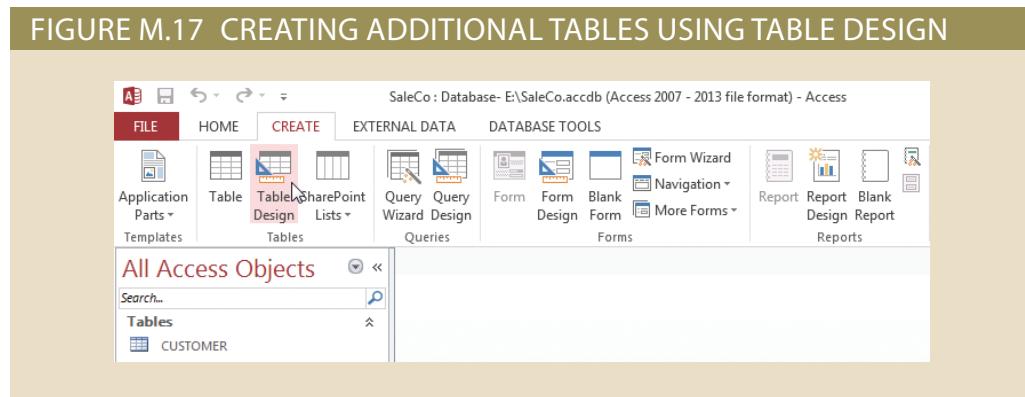
If you click the **Yes** button shown in Figure M.15, Access will return you to the Navigation Pane you saw in Figure M.11. In this case, the second record will not be saved and the table simply retains the first record “as is.” If you click the **No** button, the cursor returns to the data entry screen. This action lets you make the necessary correction by typing in a CUST_CODE value other than 10010. Click **No** to return to Design View and finishing adding the remaining records.

You are now ready to enter the remaining records. When you are done, the CUSTOMER table contents should match Figure M.16. Save and close the table.

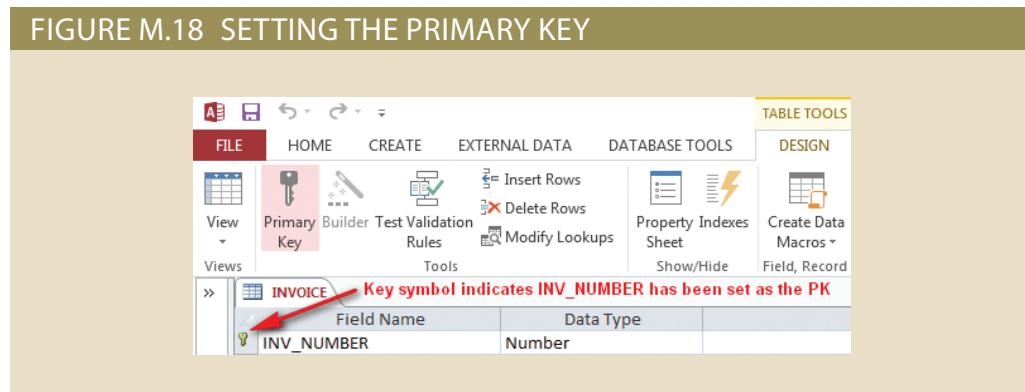
FIGURE M.16 COMPLETED CUSTOMER TABLE ENTRIES

CUST_CODE	CUST_LNAM	CUST_FNAM	CUST_INITIA	CUST_AREA	CUST_PHON	CUST_BALAN
10010	Ramas	Alfred	A	615	844-2573	\$0.00
10011	Dunne	Leona	K	713	894-1238	\$0.00
10012	Smith	Kathy	W	615	894-2285	\$162.52
10013	Olowksi	Paul	F	615	894-2180	\$536.75
10014	Orlando	Myron		615	222-1672	\$0.00
10015	O'Brien	Amy	B	713	442-3381	\$0.00
10016	Brown	James	G	615	297-1228	\$221.19
10017	Williams	George		615	290-2556	\$768.93
10018	Farriss	Anne	G	713	382-7185	\$216.55
10019	Smith	Olette	K	615	297-3809	\$0.00

The next step is to create the INVOICE table. Click the **CREATE** tab and then **Table Design** as shown in Figure M.17.



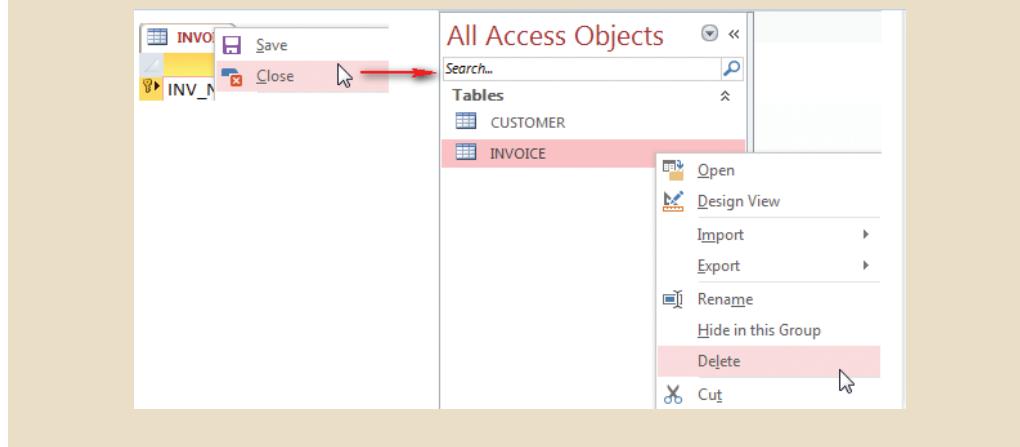
In the Field Name column, enter the field name **INV_NUMBER** and select the **Number** data type. Select the attribute **INV_NUMBER** and click the **Primary Key** button located on the DESIGN tab as shown in Figure M.18. The narrow column to the left of the **Field Name** column now shows the key symbol to indicate that the INV_NUMBER attribute is now the PK. You did not have to set this in the CUSTOMER table because Access automatically set the ID field as the primary key. We simply renamed the field. The PK stayed defined in the field. Right-click the **Table1** table tab and click **Save**. Enter **INVOICE** as the Table Name and click **OK**.



Note
If you try to save or close the table without assigning a primary key, MS Access will prompt you with a message suggesting to define one. You can save the table without a primary key by clicking No.

Right-click the **INVOICE** table tab and close the INVOICE table; right-click the **INVOICE** icon and select **Delete** as shown in Figure M.19. Then click **Yes**. We will import the remaining tables including the INVOICE table in the next step. This step was just to demonstrate how to create additional tables after the initial Access-generated table is modified.

FIGURE M.19 DELETING THE INVOICE TABLE

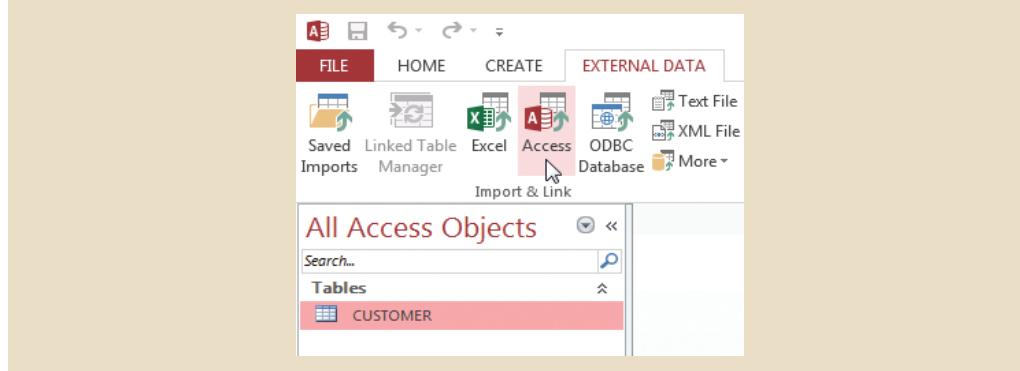


You are now ready to create the remaining tables (INVOICE, LINE, PRODUCT, and VENDOR) and their contents. However, rather than wasting your time on such repetitive tasks, we will show you how to import these items from a database you already have. That database, named **Ch07_SaleCo**, is used in the text's Chapter 7, Introduction to Structured Query Language (SQL). This database is available online, so go ahead and download it now into a folder of your choice. You will learn how to import database components from the **Ch07_SaleCo** database to your **SaleCo** database in the next section.

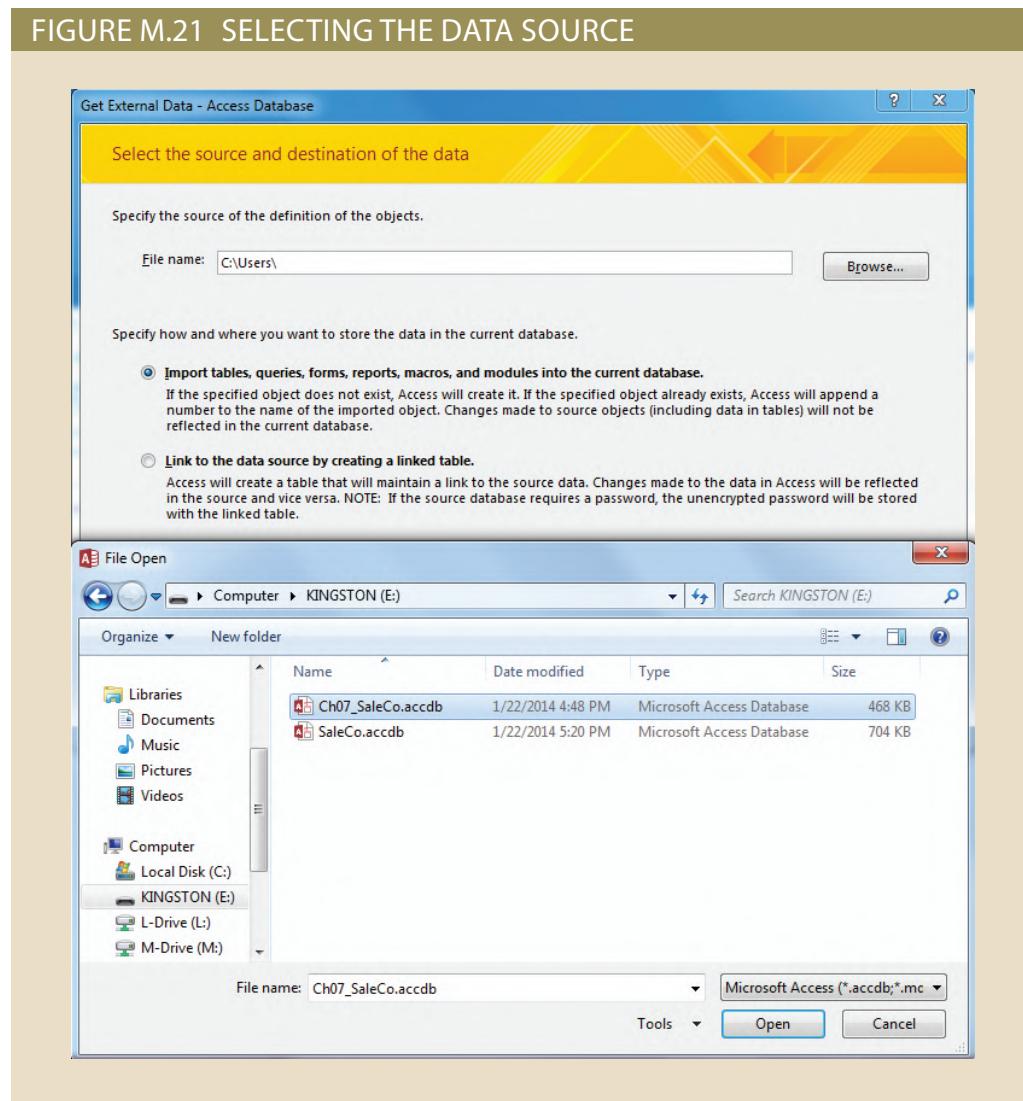
M-3 Importing Objects from Other MS Access Databases

To import a table—or, for that matter, any other Access object—start by selecting the **EXTERNAL DATA** tab; then click the **Access** button, as shown in Figure M.20.

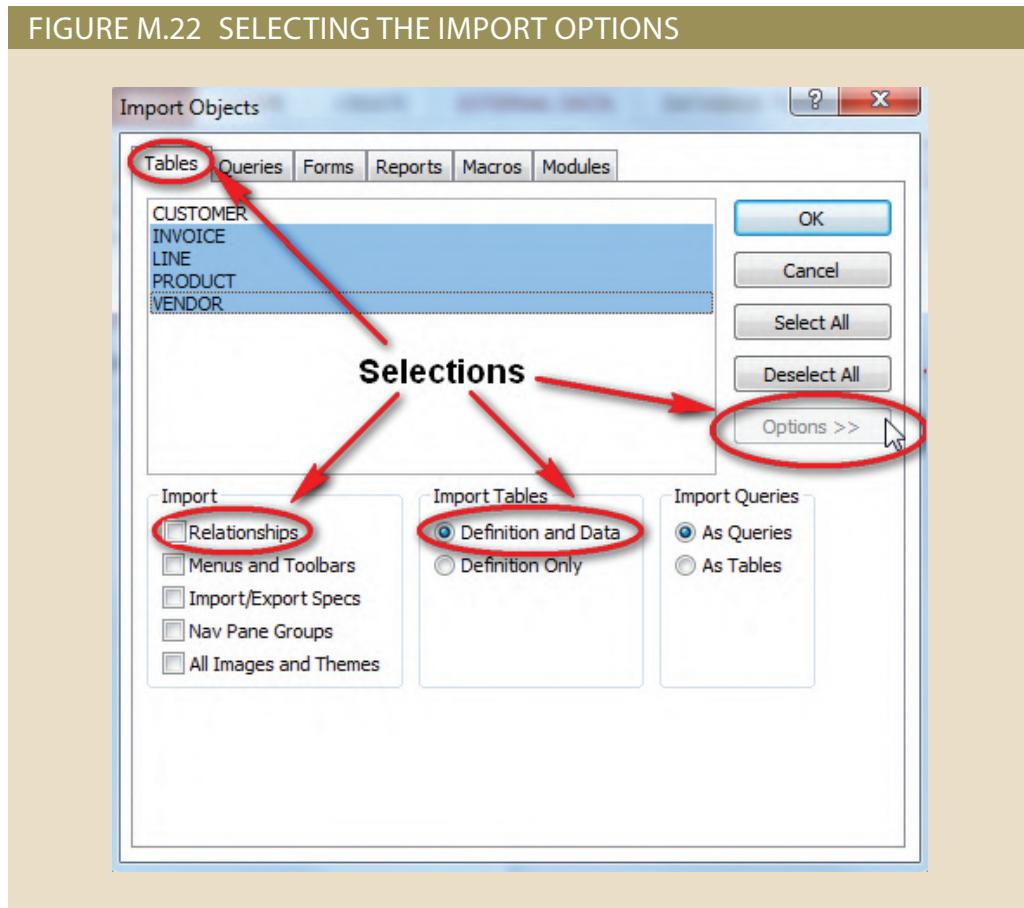
FIGURE M.20 STARTING THE IMPORT SEQUENCE



Next, select the data source. In this example, the **Ch07_SaleCo** database will be used, so move to the folder on cengagebrain.com (or to your student data folder) that contains this database, select the **Ch07_SaleCo** database as shown in Figure M.21, and click **Open**.



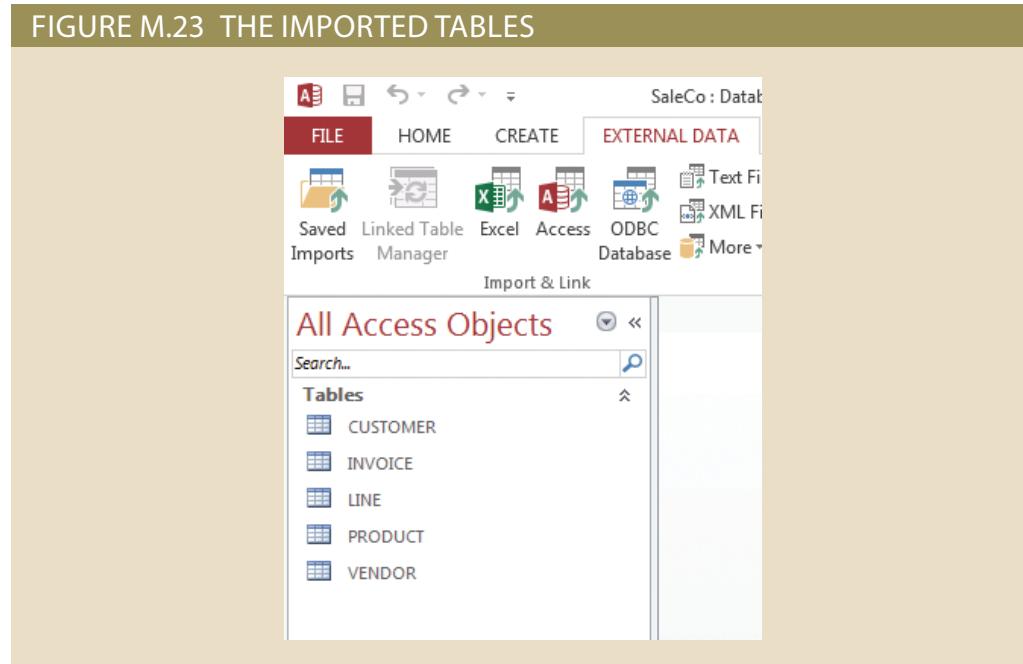
Now select what objects you want to import and in what format you want them imported. Figure M.22 shows what **Tables** are to be imported. Note that the **Options >>** have been selected to give you a chance to make the selections shown in Figure M.22.



As you examine Figure M.22, note the following selections:

- The INVOICE, LINE, PRODUCT and VENDOR tables have been marked.
- The **Relationships** option has been de-selected—no checkmark is shown in the box. (If the box contains a checkmark, click it to remove the checkmark. The reason for not including the relationships is that you will learn how to create such relationships later.)
- The **Definition and Data** option has been selected to ensure that the table structure and all the data contained in each selected table will be imported.

After you have made the selections shown in Figure M.22, click **OK** to import the tables. If you are presented with a Save Import Steps screen, click **Close**. The results of this action are shown in Figure M.23.



M-3a Editing the Imported Tables

As you can tell by looking at Figure M.23, the imported tables are all there. For example, you can open the INVOICE table in Design View to inspect the table structure (see Figure M.24). Note the INV_DATE Date/Time data type format.

FIGURE M.24 IMPORTED INVOICE TABLE STRUCTURE

The screenshot shows the Microsoft Access 'INVOICE' table structure. The 'INV_DATE' field is selected, and its properties are being viewed. A red arrow points to the 'Format' dropdown in the 'Field Properties' dialog, which is displaying various date and time formats. The 'Medium Date' format is selected.

Field Name	Data Type
INV_NUMBER	Number
CUST_CODE	Short Text
INV_DATE	Date/Time

Note the selected data format

To see the content of any of the tables, just double-click the table name in the Navigation Pane. For example, Figure M.25 shows the content of the INVOICE table in Datasheet View.

FIGURE M.25 THE INVOICE TABLE CONTENTS

The screenshot shows the Microsoft Access 'INVOICE' table in Datasheet View. The table contains 8 rows of data with the following structure:

INV_NUMBER	CUST_CODE	INV_DATE
1001	10014	16-Mar-18
1002	10011	16-Mar-18
1003	10012	16-Mar-18
1004	10011	17-Mar-18
1005	10018	17-Mar-18
1006	10014	17-Mar-18
1007	10015	17-Mar-18
1008	10011	17-Mar-18

While you have the INVOICE table open, you can change the spacing between the attribute columns. When you put the cursor on a boundary between the attribute columns, note

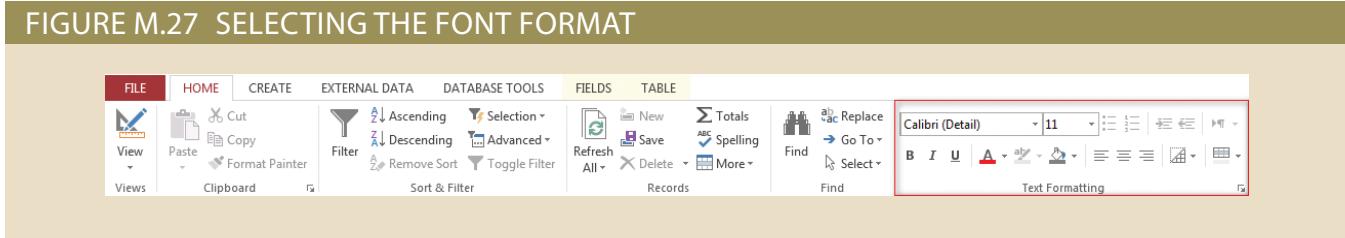
that the cursor changes to a two-sided arrow as shown in Figure M.26. You can drag the boundary to increase or decrease the column display width, or you can double-click any column boundary to let MS Access “Auto Fit” the column width. (Change the column display width to whatever size is required to show the column header or values.)

FIGURE M.26 CHANGING THE COLUMN SPACING

INV_NUMBE	CUST_CO	INV_DATE
1001	10014	16-Mar-18
1002	10011	16-Mar-18
1003	10012	16-Mar-18
1004	10011	17-Mar-18
1005	10018	17-Mar-18
1006	10014	17-Mar-18
1007	10015	17-Mar-18
1008	10011	17-Mar-18

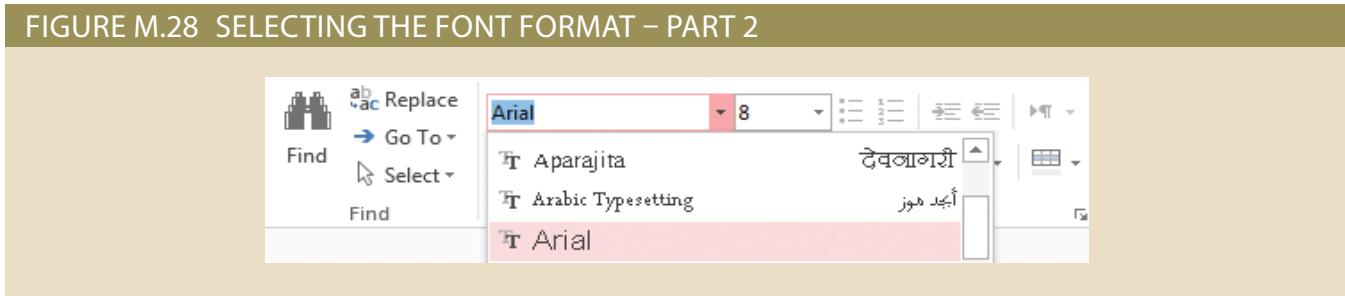
You can also change the presentation font by selecting the **Font** option in the Text Formatting group on the **HOME** tab, as shown in Figure M.27.

FIGURE M.27 SELECTING THE FONT FORMAT



Go ahead and experiment with different formatting options. A good rule to follow is this: If you don’t know what something is or what it does, just move your mouse over the option and a small help text (called a ScreenTip) will appear explaining the option. Note the selection of the font, style, and size in Figure M.28.

FIGURE M.28 SELECTING THE FONT FORMAT – PART 2



Now go ahead and review the table structures of the remaining tables. Use the database tables shown in Figure M.29 as your guide. Remember, you imported most of the tables from the **Ch07_SaleCo** database, so the table values match those in that database **at this point**.

FIGURE M.29 THE SALECO DATABASE TABLES

The screenshot shows four tables displayed simultaneously in overlapping windows:

- CUSTOMER** table:

CUST_CODE	CUST_LNAME	CUST_FNAME	CUST_INITIAL	CUST_AREACODE	CUST_PHONE	CUST_BALANCE
10010	Ramas	Alfred	A	615	844-2573	\$0.00
10011	Dunne	Leona	K	713	894-1238	\$0.00
10012	Smith	Kathy	W	615	894-2285	\$162.52
10013	Olowksi	Paul	F	615	894-2180	\$536.75
10014	Orlando	Myron		615	222-1672	\$0.00
10015	O'Brien	Amy	B	713	442-3381	\$0.00
10016	Brown	James	G	615	297-1228	\$221.19
10017	Williams	George		615	290-2556	\$768.93
10018	Farris	Anne	G	713	382-7185	\$216.55
10019	Smith	Olette	K	615	297-3809	\$0.00

- INVOICE** table:

INV_NUMBER	CUST_CODE	INV_DATE
1001	10011	18-Mar-18
1002	10011	18-Mar-18
1003	10012	18-Mar-18
1004	10011	17-Mar-18
1005	10018	17-Mar-18
1006	10014	17-Mar-18
1007	10015	17-Mar-18
1008	10011	17-Mar-18

- PRODUCT** table:

PROD_CODE	PROD_DESCRIP	PROD_INDATE	PROD_QOH	PROD_MIN	PROD_PRICE	PROD_DISCOUNT	VEND_CODE
111QER01	Power painter, 15 psi., 3-nozzle	03-Nov-11	8	5	\$109.99	0.00	25595
13-Q2P2	7.25-in. pvr. saw blade	13-Dec-11	32	15	\$14.99	0.05	21344
14-Q1L3	9.00-in. pvr. saw blade	13-Nov-11	18	12	\$17.49	0.00	21344
1546-Q02	Hrd. cloth, 1/4-in., 2x50	15-Jan-12	15	8	\$39.95	0.00	23119
1546-Q01	Hrd. cloth, 1/2-in., 3x50	15-Jan-12	23	5	\$43.99	0.00	23119
1547-Q01	Bdø jeansaw, 12-in. blade	30-Dec-11	8	5	\$109.99	0.05	24288
22320Q1E	Bdø jeansaw, 6-in. blade	24-Dec-11	6	5	\$89.87	0.05	24288
2238QDPO	Bdø cordless drill, 1/2-in.	20-Jan-12	12	5	\$38.95	0.05	25595
23109-HB	Clavi hammer	20-Jan-12	23	10	\$9.95	0.10	21225
23114-AA	Sledge hammer, 12 lb.	02-Jan-12	8	5	\$14.40	0.05	21344
54779-2T	Rat-tail file, 1/8-in. fine	15-Dec-11	43	20	\$4.99	0.00	21344
89-WRF-Q	Hicut chain saw, 16 in.	07-Feb-12	11	5	\$256.99	0.05	24288
PVC230RT	PVC pipe, 3.5-in., 8-ft	20-Feb-12	188	75	\$5.87	0.00	
SM-18277	1.25-in. metal screw, 25	01-Mar-12	172	75	\$6.99	0.00	21225
SW-23116	2.5-in. wd. screw, 50	24-Feb-12	237	100	\$8.45	0.00	21231
WR3/T3	Steel matting, 4'x8'x1/8", 5" mesh	17-Jan-12	18	5	\$119.95	0.10	25595

- VENDOR** table:

VEND_CODE	VEND_NAME	VEND_CONTACT	VEND_AREACODE	VEND_PHONE	VEND_STATE	VEND_ORDER
21225	Bryson, Inc.	Smithson	615	223-3234	TN	Y
21226	SuperLoc, Inc.	Flushing	904	216-8995	FL	N
21231	D&E Supply	Singh	615	228-3245	TN	Y
21344	Gomez Bros.	Ortega	615	889-2546	KY	N
22567	Dome Supply	Smith	901	678-1419	GA	N
23119	Randels Ltd.	Anderson	901	678-3998	GA	Y
24004	Brackman Bros.	Browning	615	228-1410	TN	N
24288	ORDVA, Inc.	Hakford	615	896-1234	TN	Y
25443	B&K, Inc.	Smith	904	227-0093	FL	N
25501	Damal Supplies	Smythe	615	890-3529	TN	N
25595	Rubicon Systems	Orton	904	456-0092	FL	Y

Note

All of the previous figures have been shown in Tabbed Document view. Figure M.29 is shown in Overlapping Window view. To change to Overlapping Window view, use the Access Options dialog box as follows:

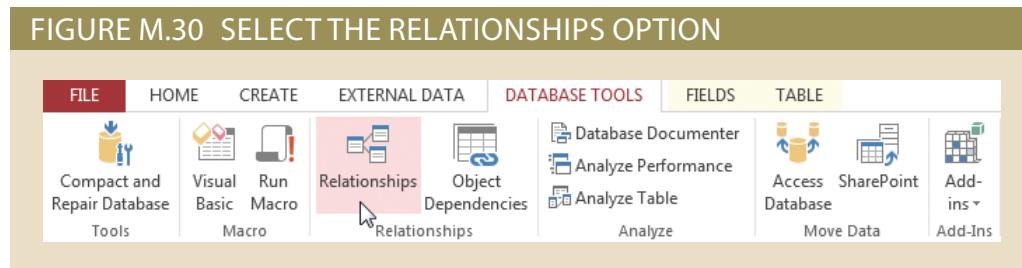
FILE/Options / Current Database / Document Window Options / Overlapping Windows

You will be prompted with a message saying "You must close and reopen the current database for the specified option to take effect." Close the SaleCo Database and reopen it. Follow this same procedure to change back to Tabbed Documents.

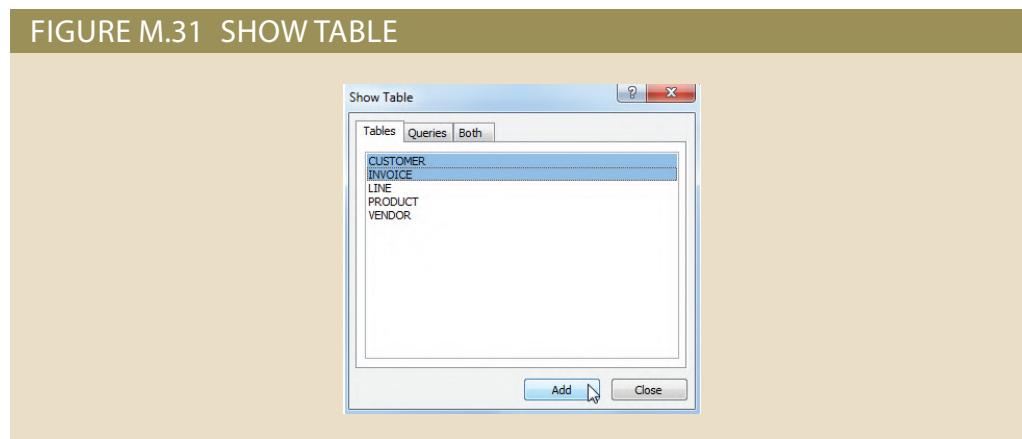
M-4 Setting the Relationships between Tables

Thus far, you have learned to create, populate, and import tables. Actually, if you were to create a real-world relational database, you would first create the table structures and then create the relationships among them. The reason for doing so is simple—you want to make sure that referential integrity is enforced. (If you don't quite remember what referential integrity is and why it is important, review the text's Chapter 3, The Relational Database Model, Section 3-2, Keys.)

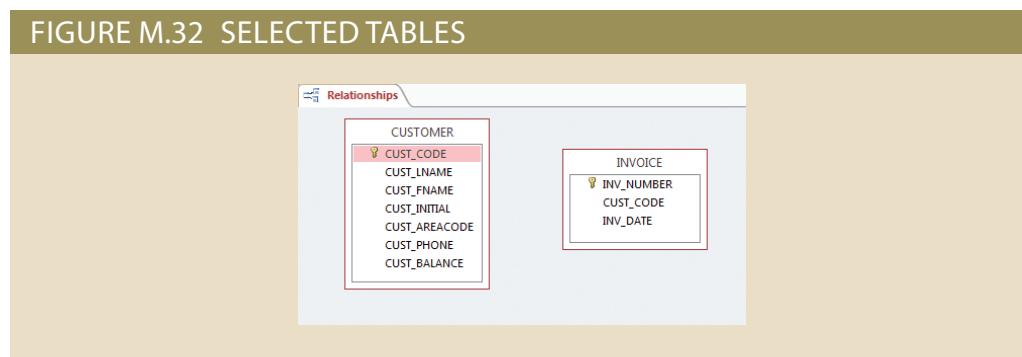
To create a relationship, start by selecting the **Relationships** button on the **DATABASE TOOLS** tab, as shown in Figure M.30.



After you click the Relationships button shown in Figure M.30, the **Show Table** dialog box will appear as seen in Figure M.31.

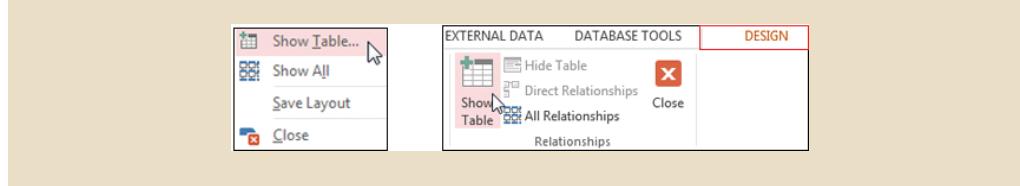


Now select the two tables as shown in Figure M.31 and then click the **Add** button. (You can also double-click each table to move it from the **Show Table** dialog box to the Relationships screen.) Click the **Close** button. This action will generate the screen shown in Figure M.32.



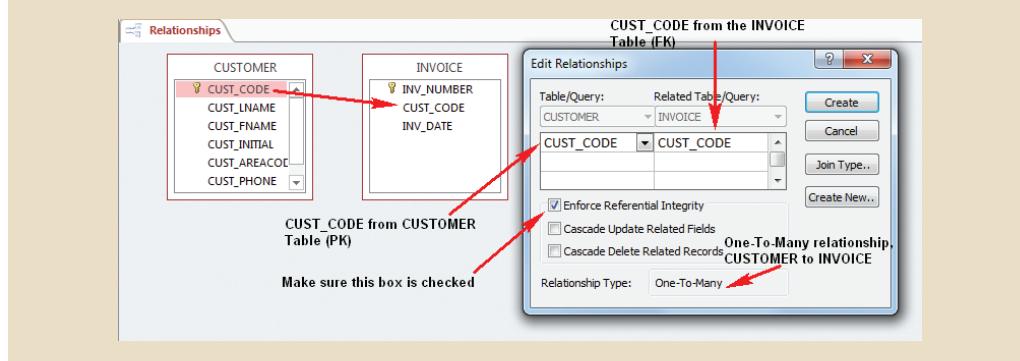
To add more tables, right-click anywhere on the blank relationships screen, as shown in Figure M.33, to show the context menu; then click **Show Table** or click the **Show Table** button located on the **DESIGN** tab. Both options will take you back to the screen shown in Figure M.31.

FIGURE M.33 ADD MORE TABLES



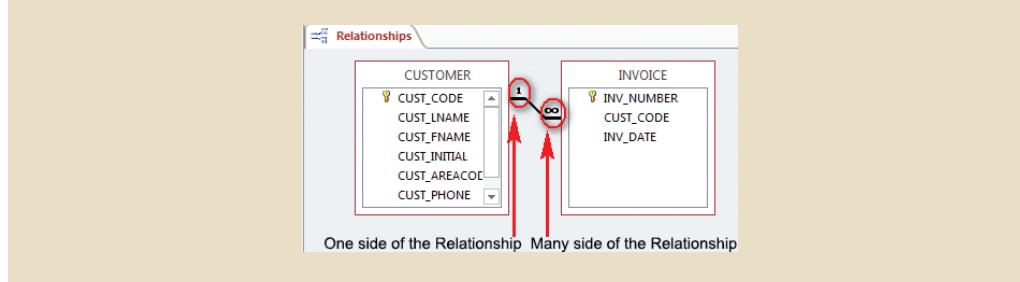
To create the relationship between the CUSTOMER and INVOICE tables, select **CUST_CODE** in the CUSTOMER table and drag and drop it on CUST_CODE in the INVOICE table. This action will produce the **Edit Relationships** dialog box you see in Figure M.34.

FIGURE M.34 CREATE RELATIONSHIP



Next, click to check the **Enforce Referential Integrity** checkbox option you see in Figure M.34's **Edit Relationships** dialog box and then click the **Create** button to create the relationship. This action will produce the results shown in Figure M.35. Note that the “1” side of the relationship is marked by the boldfaced 1, while the many (M) side of the relationship is marked by the infinity symbol ∞ .

FIGURE M.35 THE RELATIONSHIP BETWEEN CUSTOMER AND INVOICE



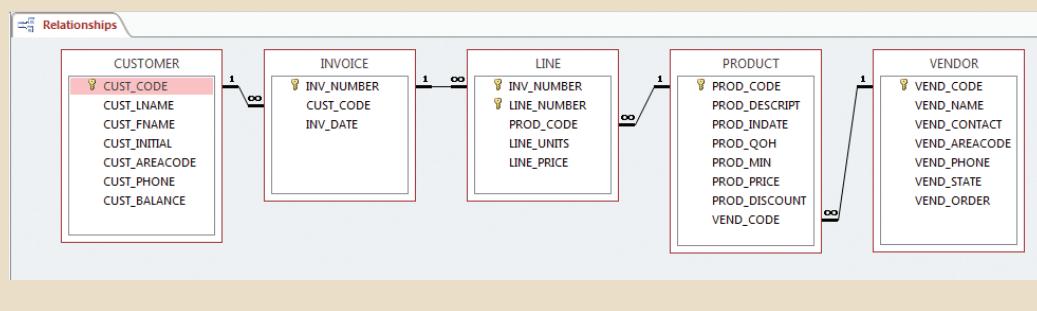


Note

Always drag and drop from the 1 side to the M side in a one-to-many (1:M) relationship.
If you are creating a relationship between tables in a 1:1 relationship, drag from the parent entity to the dependent entity.

Now go ahead and create the relationships between all the tables. When you are done, your results will resemble Figure M.36. (We have dragged and sized the tables to enhance the presentation.)

FIGURE M.36 THE RELATIONSHIPS FOR ALL THE SALECO DATABASE TABLES



After creating the relationships, click the **Close** button on the Ribbon and click **Yes** to save the changes.

M-5 Repairing and Compacting the Database

A word of caution is in order. When you do a lot of work on your database, it tends to grow rapidly and fills up with all sorts of temporary objects or objects that may no longer exist. Sooner or later, this situation will degrade the performance of your database. Therefore, it is a good idea to clean up the debris frequently.

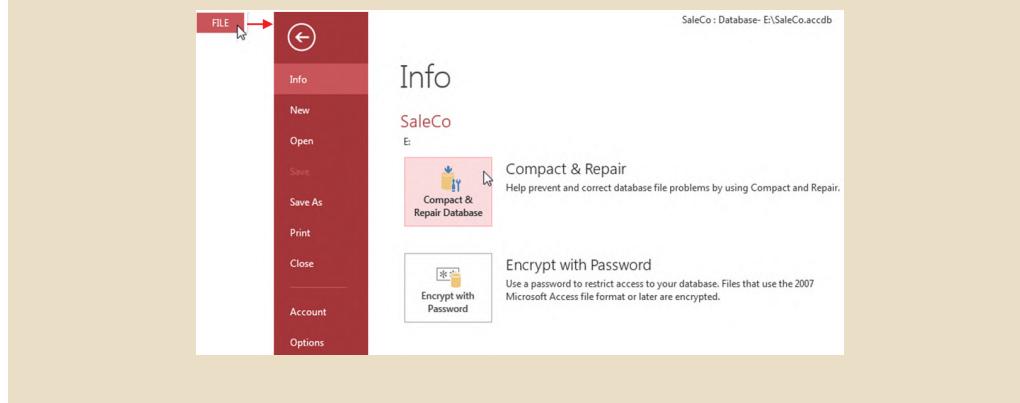
In the following sequence, note that the SaleCo database size is 704 KB at this point (see Figure M.37). Depending on the number of changes you made, your database size is likely to differ from the 704 KB you see here.

FIGURE M.37 THE SALECO DATABASE SIZE BEFORE THE CLEANUP

SaleCo.accdb 704 KB

The cleanup is accomplished by selecting the sequence shown in Figure M.38 – **FILE / Info / Compact & Repair Database**.

FIGURE M.38 COMPACT & REPAIR THE DATABASE



When the procedure is completed, look how much smaller the SaleCo database has become. Figure M.39 shows that the SaleCo database now uses only 504 KB.

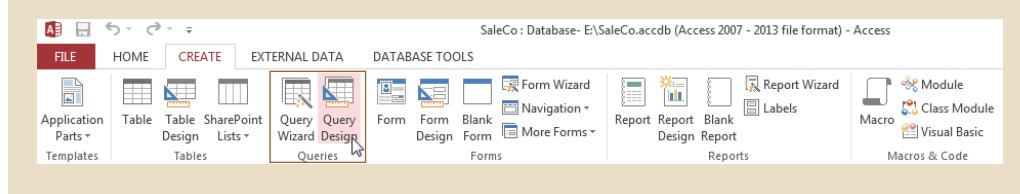
FIGURE M.39 THE SALECO DATABASE SIZE AFTER THE CLEANUP



M-6 Queries

Queries are used to extract data from the database and to help transform data into information. To create a query you will use the query design feature of MS Access. To open this feature, click the **CREATE** tab on the Ribbon and then click **Query Design** as shown in Figure M.40.

FIGURE M.40 CREATING THE FIRST QUERY



The **Show Table** dialog box appears as shown in Figure M.41. (Many of the wizards are useful, but you are better served by using the **Design View** option if you want to learn how to create and manipulate queries. All the queries in this tutorial will be created with the help of the MS Access Graphical User Interface, or GUI.)

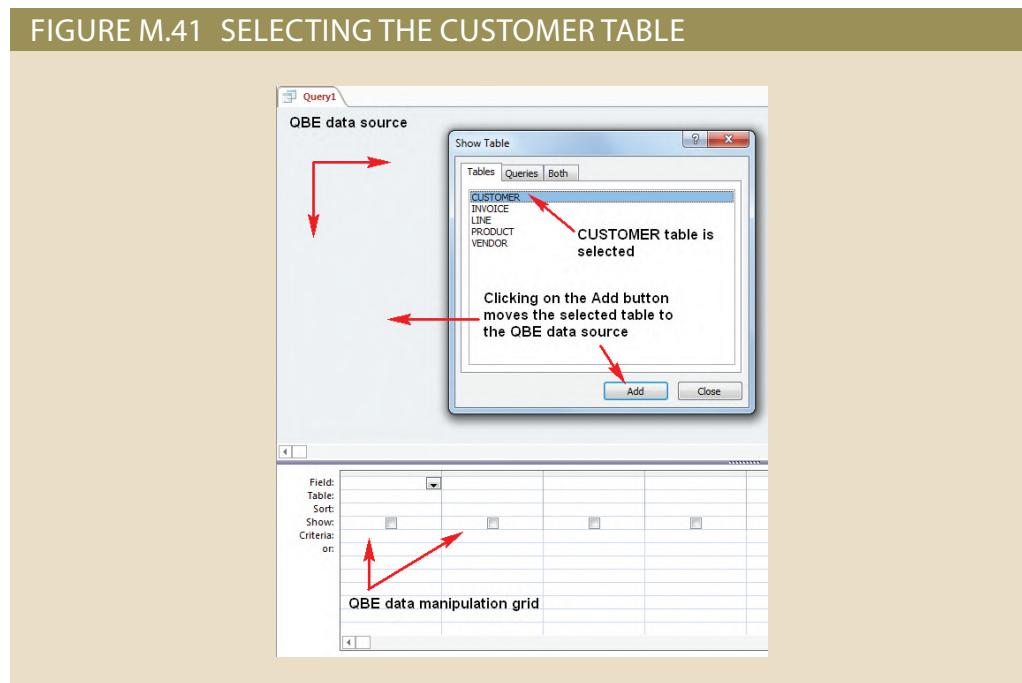
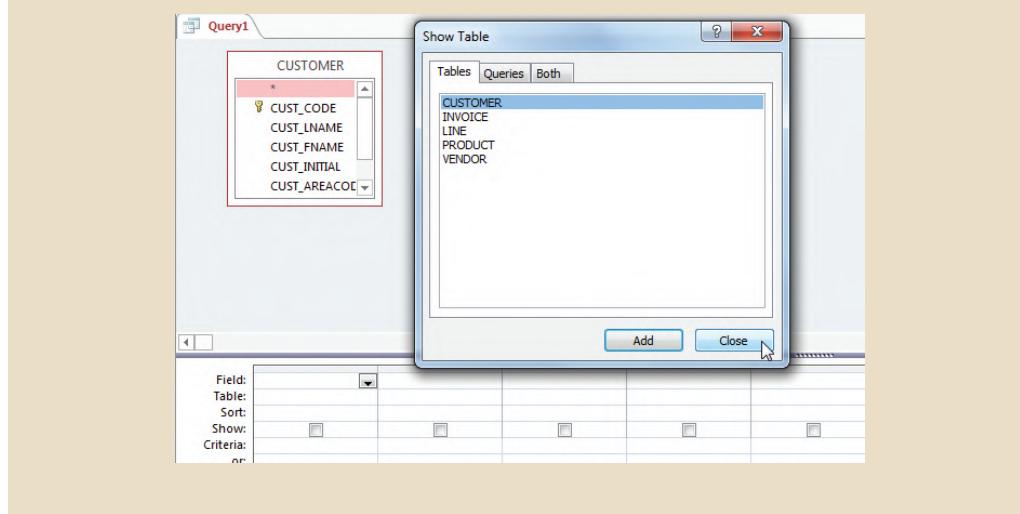


Figure M.41 shows three QBE (query by example) window components.

- The top (blank) portion of the window is the QBE window's ***data source*** display, which may be a one or more tables or queries.
- The bottom (grid) portion shows the available options in the QBE ***grid***. The grid represents the ***data manipulation*** portion of the QBE window. As its name suggest, the **Field:** option lets you place a table's—or query's—field on the output of this query. The **Table:** shows the field's origin. The **Sort:** option lets you sort selected field values in either ascending or descending order. The **Show:** option lets you select whether or not to display a selected field on the query output by checking or unchecking the box. The **Criteria:** option lets you add a condition to the query, so only the rows that meet the criteria are shown in the query output. And the **or:** option lets you add more conditions to the **Criteria:** constraint selection. (For example, you might place a restriction on the query output for a CUSTOMER table to show only customers named “Smith” **or** to also display customers whose current balance is over \$200.)

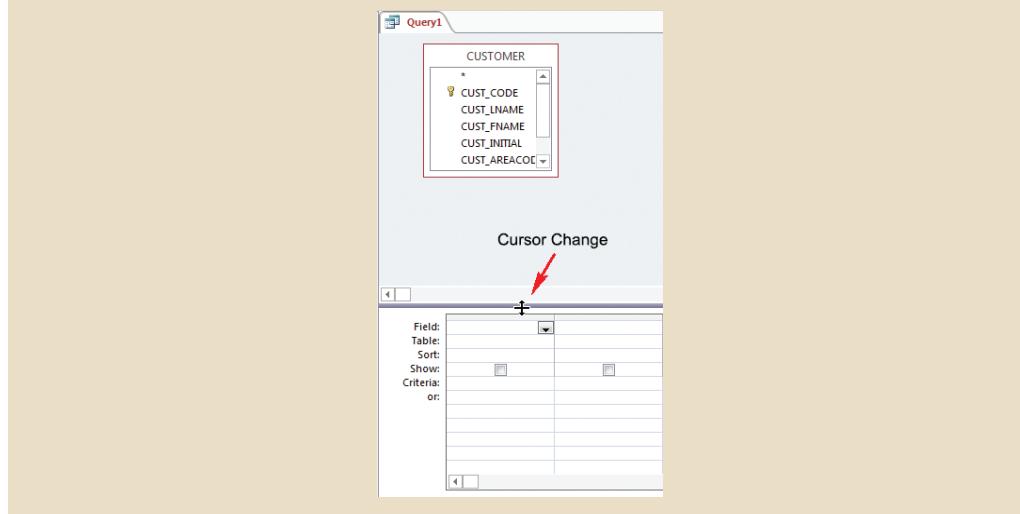
To place the CUSTOMER table on the data source portion of the screen, either select **CUSTOMER** as shown in Figure M.42 and then click the **Add** button or double-click the CUSTOMER table. When you have placed all the tables you need on the top (data source) portion of the screen, close the **Show Table** dialog box by clicking its **Close** button. (See Figure M.42.) In this example, only the CUSTOMER table has been selected as the data source.

FIGURE M.42 CLOSING THE SHOW TABLE WINDOW



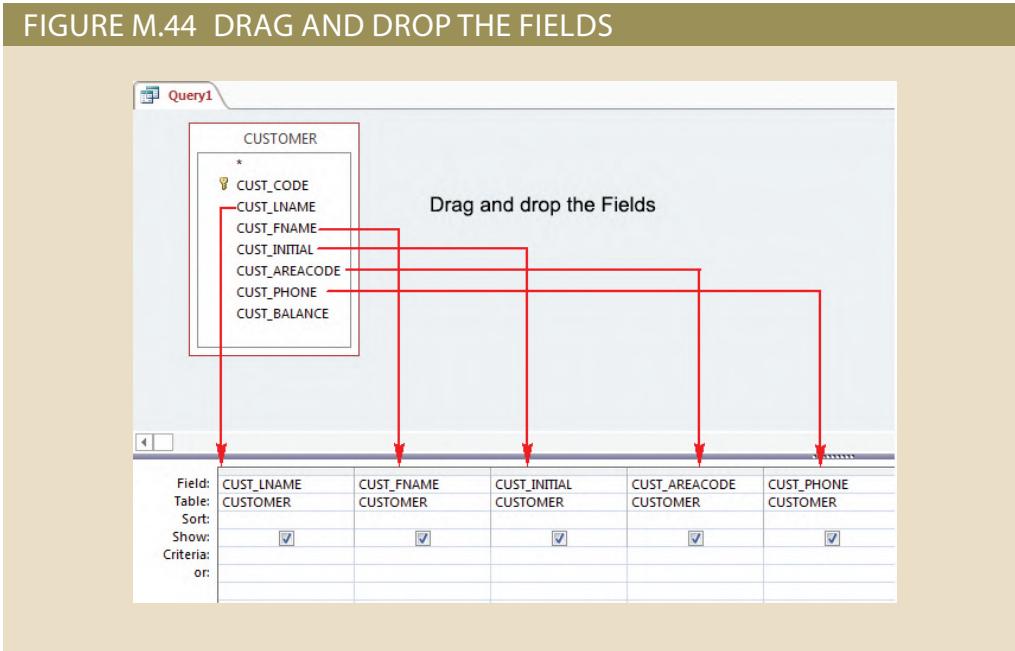
After you have selected the data source—the CUSTOMER table—and placed it on the data source portion of the QBE screen, size the QBE window’s components by dragging their limits. Figure M.43 shows that the cursor changes when you place it on the boundary between the data source component that now contains the CUSTOMER table and the data manipulation grid component. When the cursor changes its shape to the two-sided arrow seen here, you can drag the boundary up or down.

FIGURE M.43 SIZE THE TABLE WINDOW



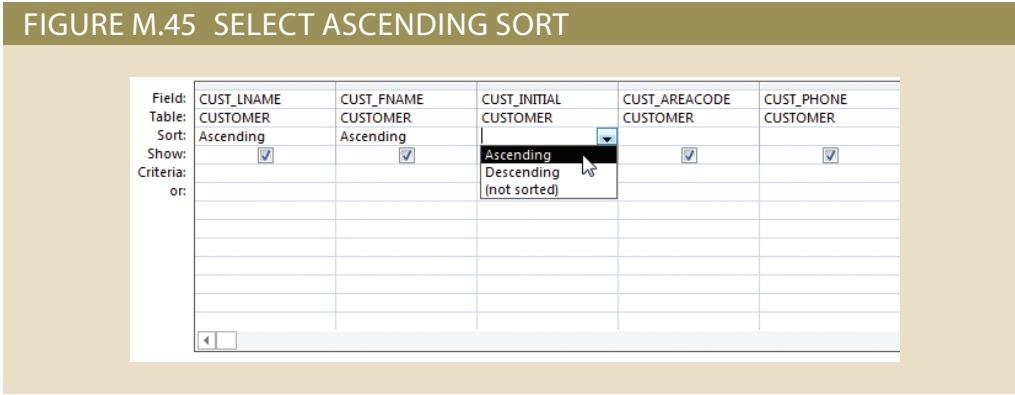
After enlarging the data component portion of the QBE window, drag the CUSTOMER table box lower corner to show all the fields. Then select the CUSTOMER table's fields and drag and drop them in the **Field:** spaces as shown in Figure M.44. You can drag and drop the individual fields from the CUSTOMER table to the field columns in the grid section or you can select multiple fields by CTRL-clicking them and dropping them on a single field space. (Access will automatically space them across the field columns.) Note that the field origin is automatically displayed in the Table: portion of the grid.

FIGURE M.44 DRAG AND DROP THE FIELDS



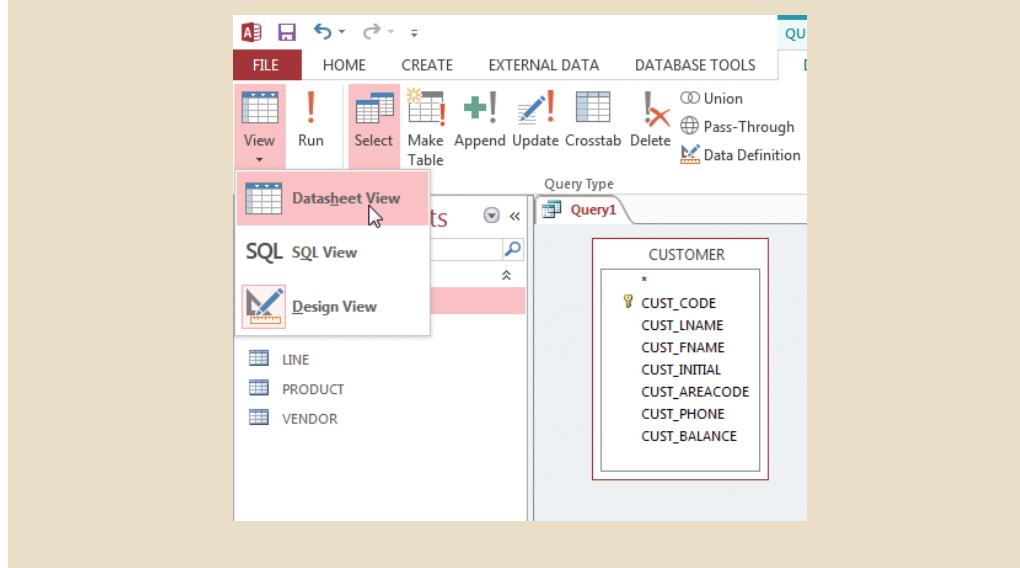
After you have selected the fields you want to use in the query, you can choose whether or not to sort the field values by clicking on the grid's **Sort:** option. Clicking on this option yields a drop-down list of sort options you see in Figure M.45. Click the option you want to use. Set the CUST_LNAME, CUST_FNAME, and CUST_INITIAL fields to sort the fields in ascending order from left to right, as shown in Figure M.45. If you do not set a sort order, MS Access will display the records in random order.

FIGURE M.45 SELECT ASCENDING SORT



If you want to see the effect of the query actions you have taken thus far, change the **View** from the current Design View to **Datasheet View**, as shown in Figure M.46. Or you could click the **Run** button on the **DESIGN** tab on the Ribbon.

FIGURE M.46 SELECT DATASHEET VIEW

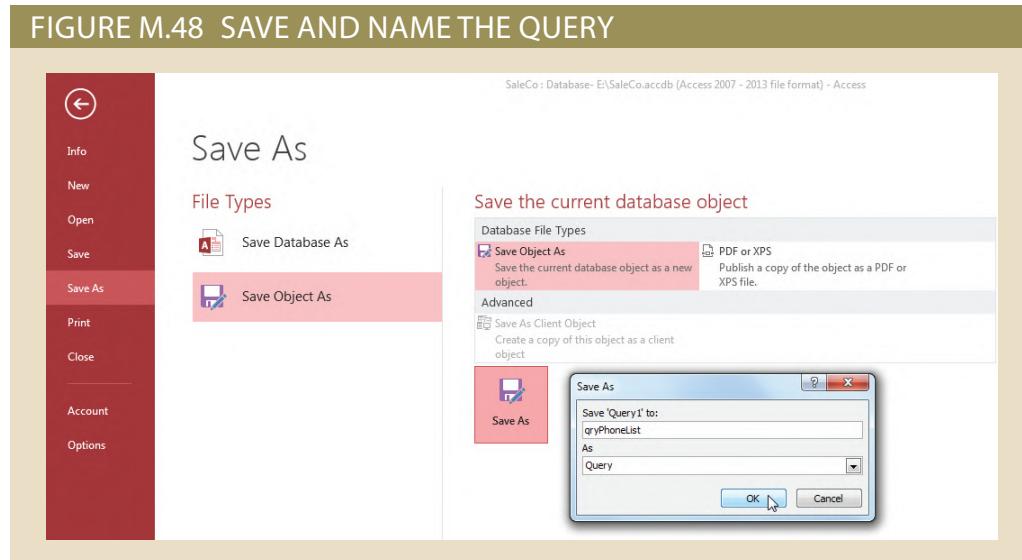


The query output is shown in **Datasheet View** as illustrated in Figure M.47.

FIGURE M.47 DATASHEET VIEW

CUST_LNAME	CUST_FNAME	CUST_INITIAL	CUST_AREACODE	CUST_PHONE
Brown	James	G	615	297-1228
Dunne	Leona	K	713	894-1238
Farriss	Anne	G	713	382-7185
O'Brien	Amy	B	713	442-3381
Olowski	Paul	F	615	894-2180
Orlando	Myron		615	222-1672
Ramas	Alfred	A	615	844-2573
Smith	Kathy	W	615	894-2285
Smith	Olette	K	615	297-3809
Williams	George		615	290-2556

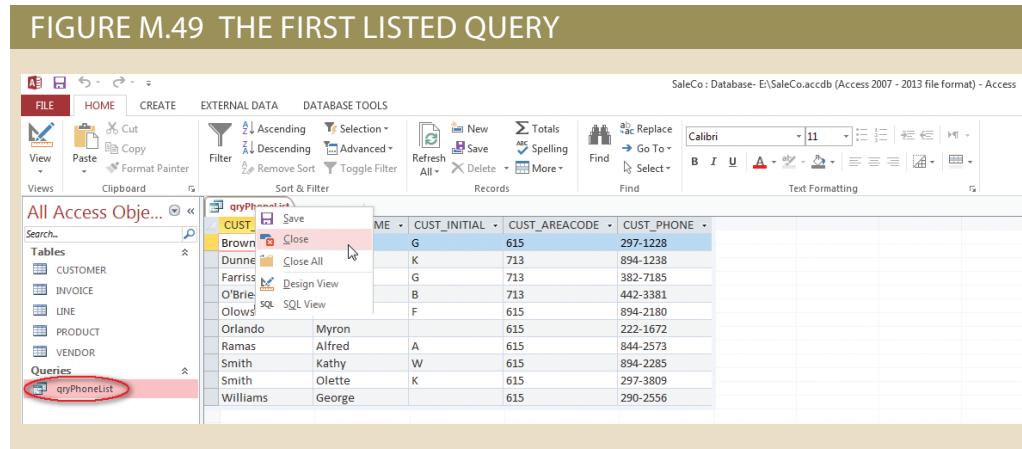
Before you do any additional work on the query, save it. Figure M.48 shows the selection of the **FILE / Save As / Save Object As / Save As** option.



The selection of the **FILE / Save As / Save Object As / Save As** option will produce the **Save As** dialog box you see in Figure M.48. The dialog box shows a default query name, in this case, **Query1**.

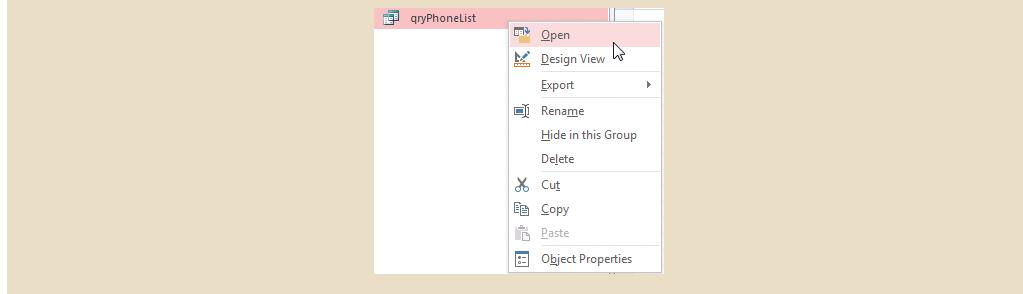
Always use a query name that is self-documenting. In this example, the query will be used as the basis for a phone list. Therefore, **PhoneList** is an appropriate name. However, you want to also show that this object is a query, so use the prefix **qry** to indicate that fact. Therefore, the appropriate name will be **qryPhoneList**. And you see that query name used in Figure M.48. (You will discover that this self-documenting naming convention is very desirable, because it enables you to easily keep track of multiple components in an application set. For example, you will learn how to create forms in Section M-7, and if you see two objects, **frmPhoneList** and **qryPhoneList**, you will know which object you are looking at, and it will be easy to see that the data source for the form is the query that has the same name.)

After you type the query name as shown in Figure M.48, click **OK** to save the query. Note that the saved query now shows up under the **Queries** header in the Navigation Pane as shown in Figure M.49. Right-click the **qryPhoneList** query tab and click **Close** as shown in Figure M.49.



Select the query **Open** option shown in Figure M.50. (You can also open the query by double-clicking the query name.)

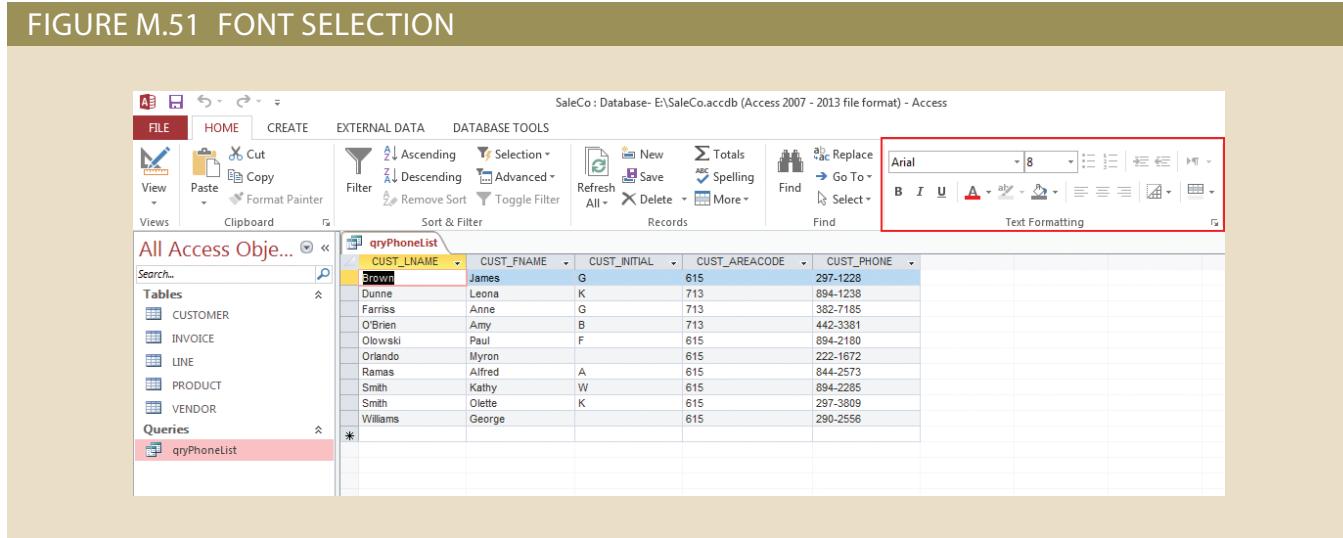
FIGURE M.50 OPEN THE QUERY



M-6a Editing the Query Output

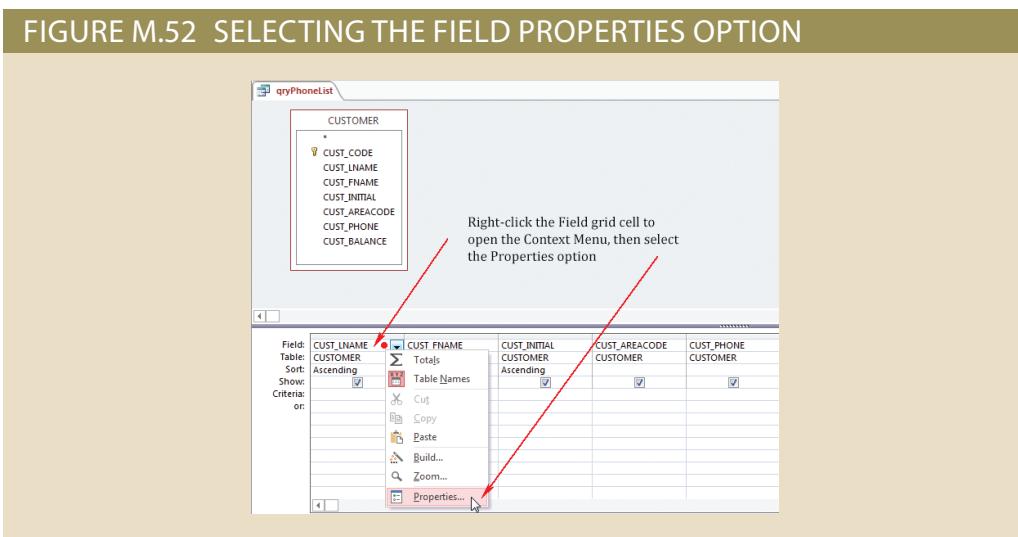
If you are not satisfied with the default font selection, font size, color, etc. when you open the query in its Datasheet View, you can easily modify those. Just click the **HOME** tab on the Ribbon to view the **Text Formatting** group shown in Figure M.51, where you can select the font, size, and style. Figure M.51 shows the selection of an Arial font, size of 8, and no style (bold, italics or underline).

FIGURE M.51 FONT SELECTION



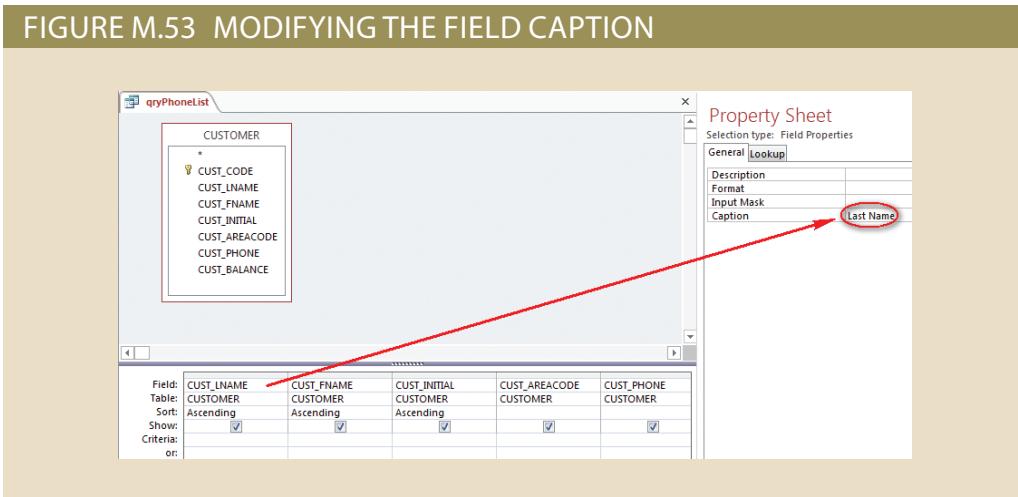
Because the default query output uses the field headers—such as **CUST_LNAME**—used by the query data source table, you might want to make the presentation more “finished” looking by changing the query field headers. You can get that job done while you are in the query design mode. Note the procedure illustrated in Figure M.52. That is, right-click a selected field name grid cell to open the context menu, and then select **Properties** to generate the Property Sheet window you see in Figure M.53.

FIGURE M.52 SELECTING THE FIELD PROPERTIES OPTION



You can change the field header by selecting the **Caption** option in the Property Sheet window you see in Figure M.53. Then type in the field header you want to use. In this case, the field header will be **Last Name**. Changing the query header does *not* affect the attribute name (CUST_LNAME) in the query data source—the CUSTOMER table.

FIGURE M.53 MODIFYING THE FIELD CAPTION



Repeat the editing for the remaining query fields and then open the query in its **Datasheet View** to see the editing results shown in Figure M.54.



Note

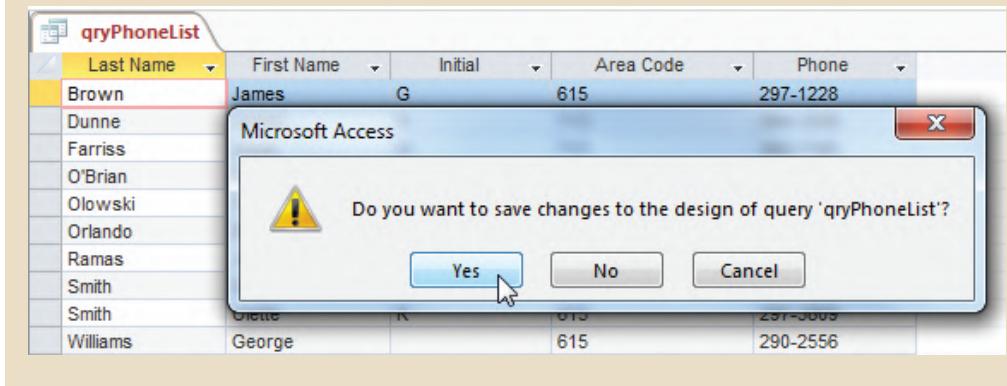
MS Access will automatically use the table field's captions, if they are defined. When defining a table, one of the field properties is the **Caption** property (see Figure M.9, under the General tab in the bottom section of the table definition window.) If you define captions for the table fields, those captions will be automatically be used for all queries, forms, and reports you create afterward.

FIGURE M.54 THE MODIFIED QUERY FIELD HEADERS

Last Name	First Name	Initial	Area Code	Phone
Brown	James	G	615	297-1228
Dunne	Leona	K	713	894-1238
Farriss	Anne	G	713	382-7185
O'Brien	Amy	B	713	442-3381
Ołowski	Paul	F	615	894-2180
Orlando	Myron		615	222-1672
Ramas	Alfred	A	615	844-2573
Smith	Kathy	W	615	894-2285
Smith	Olette	K	615	297-3809
Williams	George		615	290-2556

If you are satisfied with the results, remember to save the query again. (If you forget to save, Access will remind you as shown in Figure M.55.)

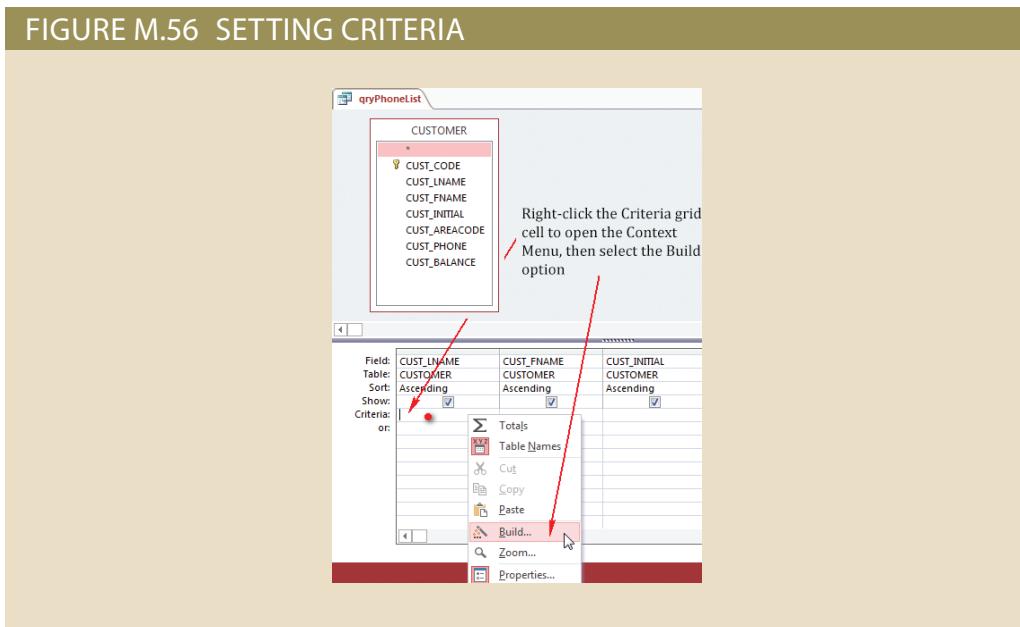
FIGURE M.55 REMINDER TO SAVE



M-6b Parameter Queries

You can easily create a query in which you specify the **criteria** governing the query output. Note the procedure summarized in Figure M.56. In the following example, the objective will be to limit the phone list output to a specified customer's last name. (A query whose output is limited through specified criteria that restrict output for one or more parameters is also known as a **parameter query**.)

FIGURE M.56 SETTING CRITERIA



If you want to limit the query phone list output to customers whose last name is “Smith,” you can simply type “Smith” in the CUST_LNAME criteria grid cell (marked by the red dot in Figure M.56). Unfortunately, that procedure means that the query must be changed each time a different last name limitation is required. You will have a much more flexible query if you let the end user specify the last name restriction through a dialog box. Such a dialog box is created automatically if you type

Like “*” & [Enter customer last name] & “*”

in the CUST_LNAME criteria grid cell. (Review the LIKE operator and wildcard characters such as “*” in Chapter 7, Introduction to Structured Query Language (SQL), Section 7-4d.)

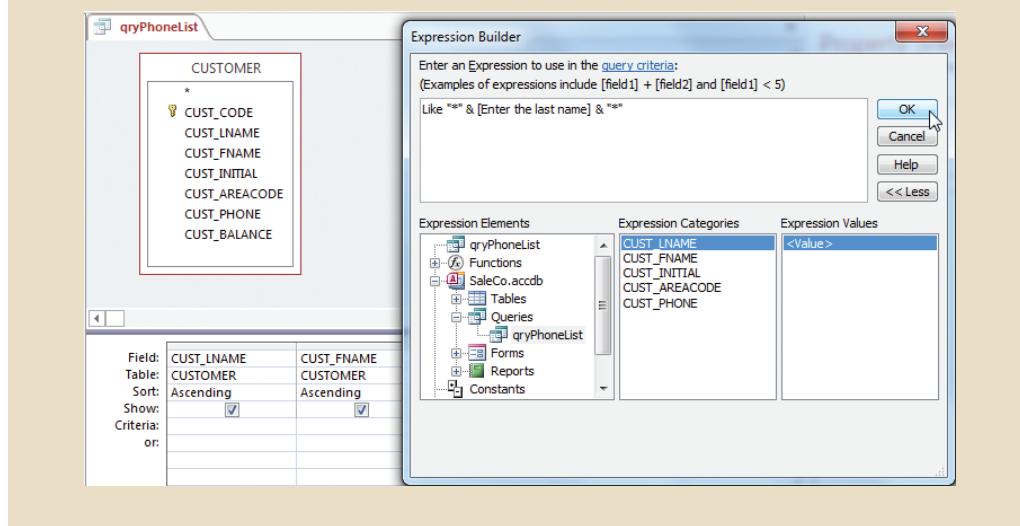


Note

Although you can type the simple criteria restriction directly into the grid, you should become used to the **Expression Builder**. This tool is especially useful when you later try to enter more complex criteria or even simple criteria with multiple components. You may have a difficult time typing the sometimes long character strings without making errors; it will be a lot easier to select items from a list than to do the typing. Therefore, we will use the Expression Builder in most examples.

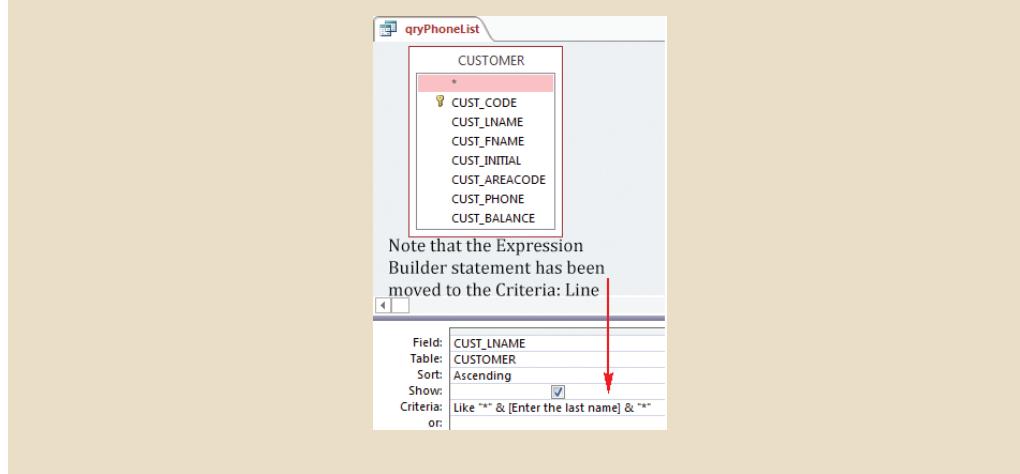
If you click the **Build...** option shown in Figure M.56, you will see the **Expression Builder** window. You can then type the entire expression as shown in Figure M.57.

FIGURE M.57 THE EXPRESSION BUILDER



When you have completed the expression shown in Figure M.57, click **OK** to close the Expression Builder and to transfer the expression to the QBE grid as shown in Figure M.58. (If you want to see the entire expression in the grid space, drag the grid column limit to widen it, as was done in Figure M.58.)

FIGURE M.58 THE COMPLETED CRITERIA LINE



Next, open the query in its **Datasheet View**. The expression you see in Figure M.58 will trigger the input request you see in Figure M.59. Type the last name **Smith** to generate the output shown in Figure M.60. Incidentally, the parameter search is not case-sensitive. Therefore, it does not matter whether you type **SMITH**, **smith**, **Smith**, or any other combination of lower- and uppercase letters. Also, keep in mind that the use of the ***** wildcard character in combination with **Like** will yield the results shown in Table M.3:

TABLE M.3

INPUT CRITERIA WITH “LIKE” AND “*”

INPUT	OUTPUT
None. (You just tap the Enter key, instead of typing a character and then tapping the Enter key.)	All records.
The letter s .	All records corresponding to a customer whose last name includes the letter s . For example, Ramas s , Williams, Olowski, Farris s , and Smith would all be included.
The letters br .	All records corresponding to a customer whose last name includes the letters br . For example, customer O'Brian and Brown would be included.

FIGURE M.59 NAME SELECTION

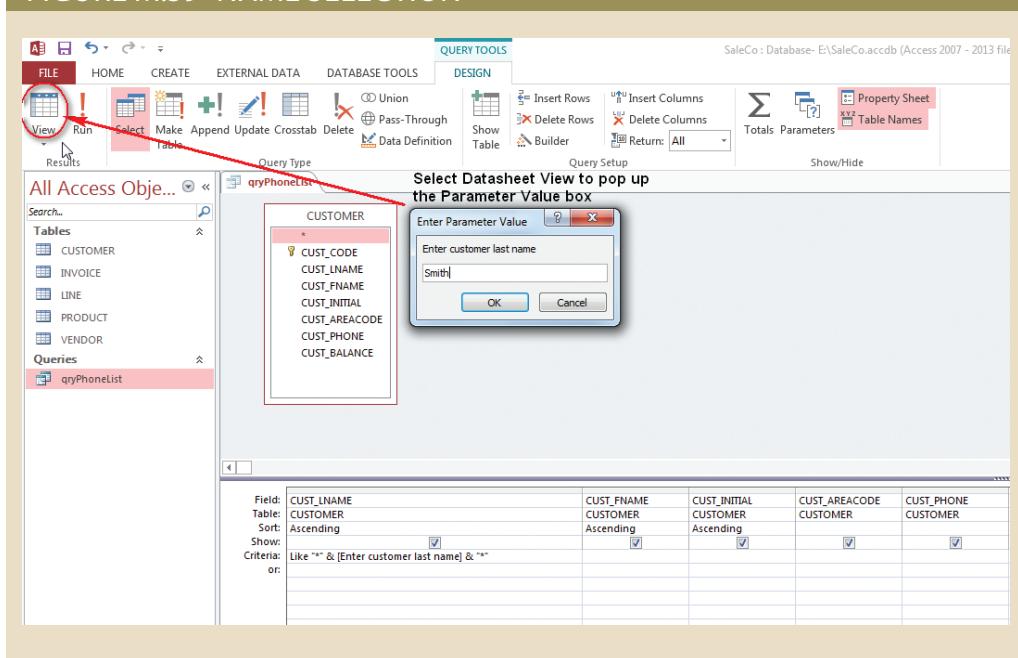
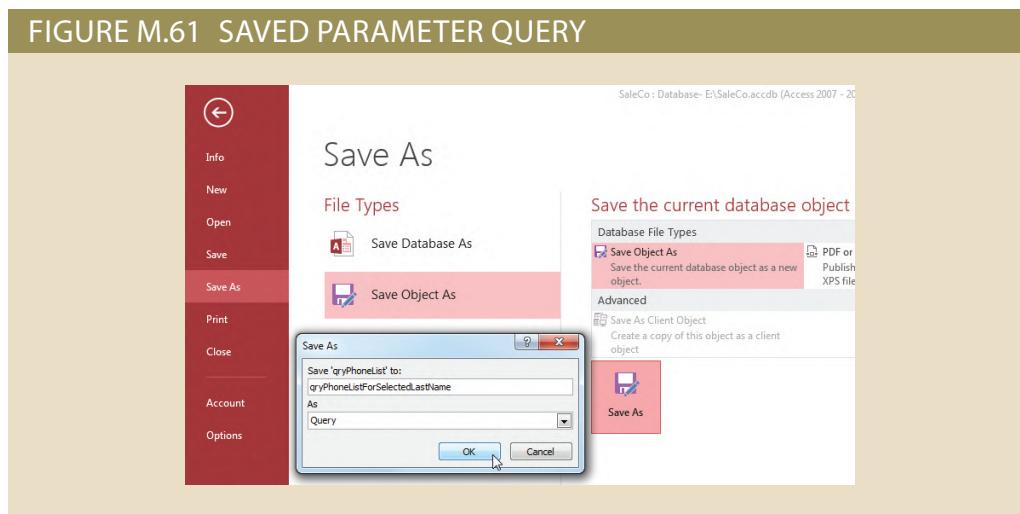


FIGURE M.60 PHONE NUMBERS FOR SELECTED NAME

Last Name	First Name	Initial	Area Code	Phone
Smith	Kathy	W	615	894-2285
Smith	Olette	K	615	297-3809

Use the **Save As** option to save a new version of this query. Save the **qryPhoneList** query you have just modified as **qryPhoneListForSelectedLastName** as seen in Figure M.61.

FIGURE M.61 SAVED PARAMETER QUERY



You can use the same technique to limit output by any selected criteria for any field. For example, Figure M.62 shows a query that limits its output by VEND_CODE values that are null using the PRODUCT table. (In short, the output will show all products that do not have a known vendor.) The output is shown in Figure M.63.

FIGURE M.62 PRODUCTS WITHOUT VENDORS

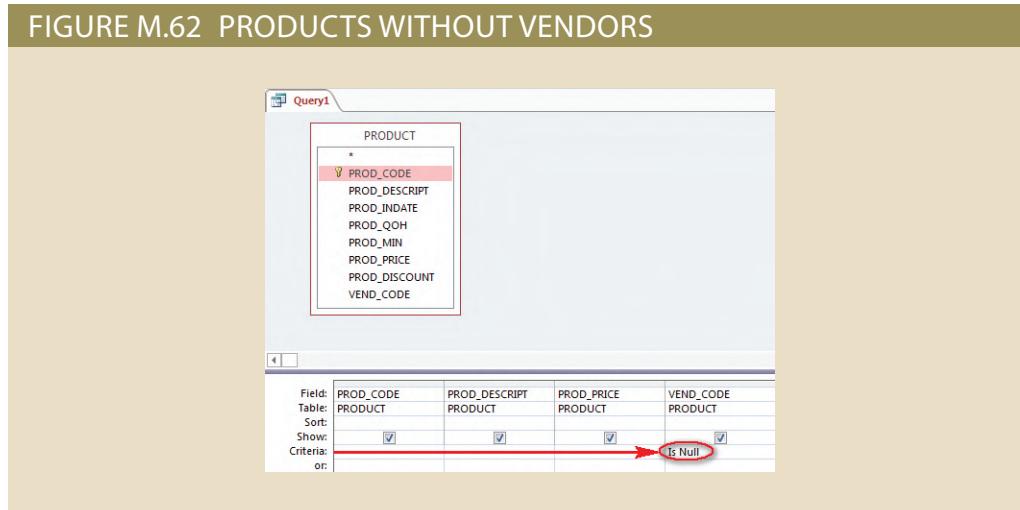


FIGURE M.63 PRODUCTS WITHOUT VENDORS OUTPUT

PROD_CODE	PROD_DESCRPT	PROD_PRICE	VEND_CODE
23114-AA	Sledge hammer, 12 lb.	\$14.40	
PVC23DRT	PVC pipe, 3.5-in., 8-ft	\$5.87	

Note that the query was saved as **qryProductsWithoutVendors**.

M-6c Multiple Table Queries

You can include any number of tables or queries in your queries. For example, note that the query in Figure M.64 includes three tables: CUSTOMER, INVOICE and LINE. You do not have to relate the tables, because that was done earlier when you setup the relationships (see Section M-4).

Examine the LINE table and look at INV_NUMBER 1004. The Customer ordered 3 products for \$4.99 and 2 products for \$9.95. The Invoice Amount before tax is \$34.87. We are going to skip right to adding the tax to the Invoice Amount to achieve the Invoice total for all invoices as seen in Figure M.64a.

- Select the **Totals** button at the top of the screen (shown by the summation symbol). This will add a Total row to each of the fields, and by default they will all be set to **Group By**.
- Right-click the empty field to the right on INV_DATE and select **Build**. Build the Expression named INV_TOTAL as seen in Figure M.64. The expression is as follows: $\text{Sum}(([LINE_UNITS]*[LINE_PRICE]) + (([LINE_UNITS]*[LINE_PRICE])*0.08))$
- Set the INV_TOTAL Property Sheet to **Format: Currency**.
- INV_TOTAL is now a calculated field derived from the price of the product * the quantity of product which would result in the (Invoice Amount) + (the tax on the Invoice Amount). Note that the default name for the Expression will be Expr1, change this to INV_TOTAL.
- Make sure in the **Total** row that **Group By** is selected for CUST_CODE, CUST_LNAME, CUST_FNAME, CUST_INITIAL, INV_NUMBER and INV_DATE. INV_TOTAL should be selected as an **Expression**.

FIGURE M.64 MULTIPLE TABLE QUERY

Save this query as **qryCustomerInvoices** and then open this query in **Datasheet View** to see its output shown in Figure M.64a.

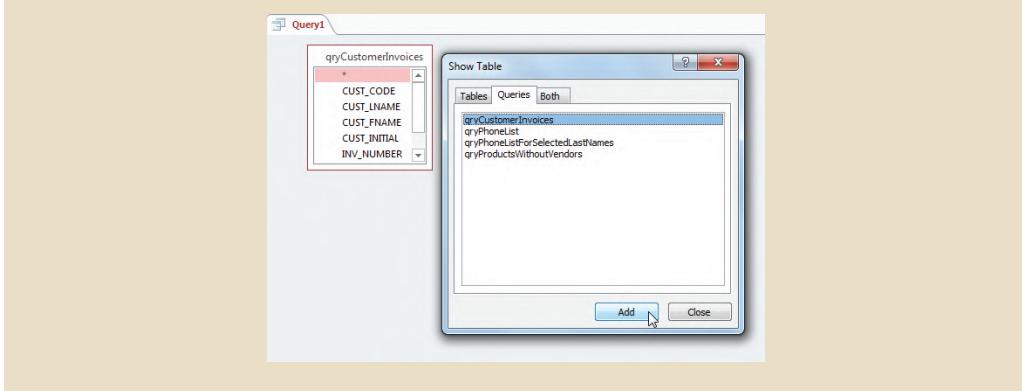
FIGURE M.64A THE QRYCUSTOMERINVOICES OUTPUT

CUST_CODE	CUST_LNAME	CUST_FNAME	CUST_INITIAL	INV_NUMBER	INV_DATE	INV_TOTAL
10011	Dunne	Leona	K	1002	16-Mar-18	\$10.78
10011	Dunne	Leona	K	1004	17-Mar-18	\$37.66
10011	Dunne	Leona	K	1008	17-Mar-18	\$431.08
10012	Smith	Kathy	W	1003	16-Mar-18	\$166.16
10014	Orlando	Myron		1001	16-Mar-18	\$26.94
10014	Orlando	Myron		1006	17-Mar-18	\$429.66
10015	O'Brien	Amy	B	1007	17-Mar-18	\$37.77
10018	Farriss	Anne	G	1005	17-Mar-18	\$76.08

M-6d Querying a Query

Suppose you want to know the total sales for each of the customers. If you run the **qryCustomerInvoices** query, you will see all the invoices for each of the customers. For example, if you look at Figure M.64a, you see that customer 10011, Leona Dunne, has three invoices for \$10.78, \$37.66, and \$431.08, respectively. The sum of these invoice totals will be \$479.52. How do write a query that will generate the sum of all the invoice totals for each of the customers? The answer turns out to be simple: As you can see in Figure M.65, you can write a query that uses the **qryCustomerInvoices** query output as its data source. (Note that the **Show Table** dialog box shows that the **Queries** tab was selected.)

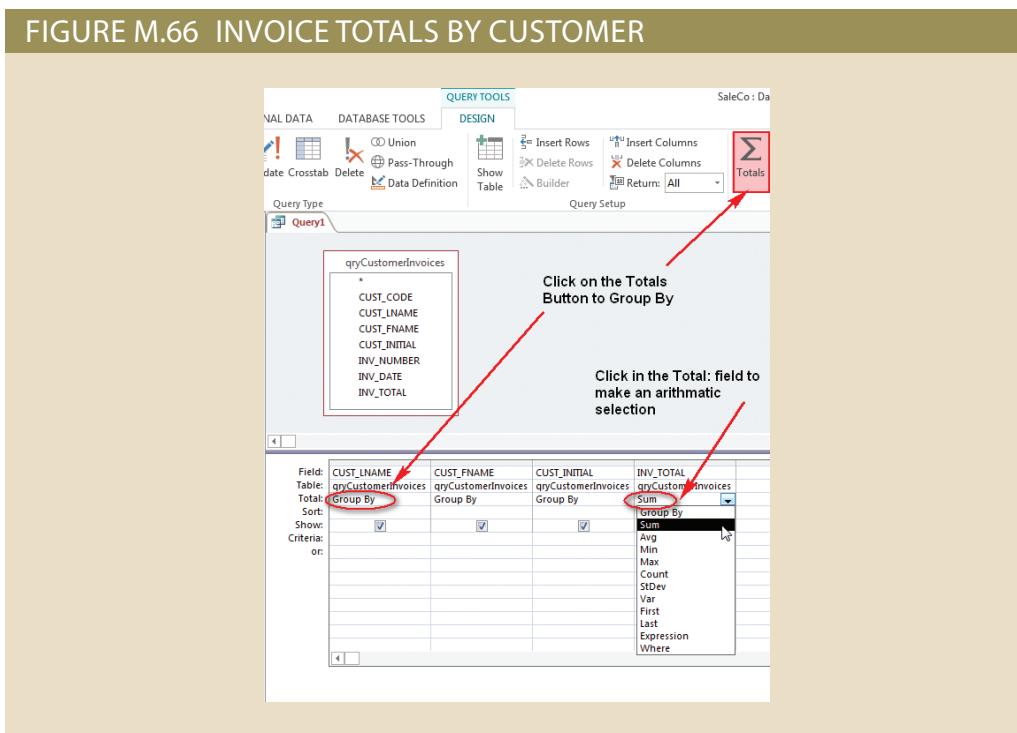
FIGURE M.65 QUERYING A QUERY



After defining the data source, you must define what fields you want to list in your query. Figure M.66 shows the remaining actions required to complete the query design:

- Select **CUST_LNAME**, **CUST_FNAME**, **CUST_INITIAL**, and **INV_TOTAL** fields.
- Click the **Totals** button on the Ribbon. This adds the **Total** row in the query grid with **Group By** option selected by default.
- In the **INV_TOTAL** field, click the **Total** row down arrow and select the **Sum** option from the list.
- Set the Properties for **INV_TOTAL** to **Format: Currency**.

FIGURE M.66 INVOICE TOTALS BY CUSTOMER



Now check the completed query in its **Datasheet View** to generate the results shown in Figure M.67. Note that the \$479.52 sum for customer Leona Dunne is correct. (Customer Leona Dunne's customer number is 10011.)

FIGURE M.67 CUSTOMER INVOICE TOTALS

qryCustomerInvoiceTotals			
CUST_LNAME	CUST_FNAME	CUST_INITIAL	SumOfINV_TOTAL
Dunne	Leona	K	\$479.52
Farris	Anne	G	\$76.08
O'Brian	Amy	B	\$37.77
Orlando	Myron		\$456.59
Smith	Kathy	W	\$166.16

Save the query as **qryCustomerInvoiceTotals** to complete the query design process.

M-7 Forms

While queries let you get data and/or information from the database, forms let you control the presentation format much better. In addition, forms will enable you to control data input and to present the results from multiple queries and/or tables. Forms can also be used to tie the application components together through menus and other devices. In short, forms are the way in which the end user is best connected to the database and they provide that “professional” look to your data management efforts.

Forms can be based on tables and/or queries. The simplest and most efficient way to create a form is to follow the steps outlined below (also shown in Figure M.68). The steps to quickly create a form are the following:

- Select the **CREATE** tab on the Ribbon.
- Select the **CUSTOMER** table then select the **Form** button to create the form.
- The result is the CUSTOMER form shown in Figure M.69.

FIGURE M.68 STARTING A NEW FORM

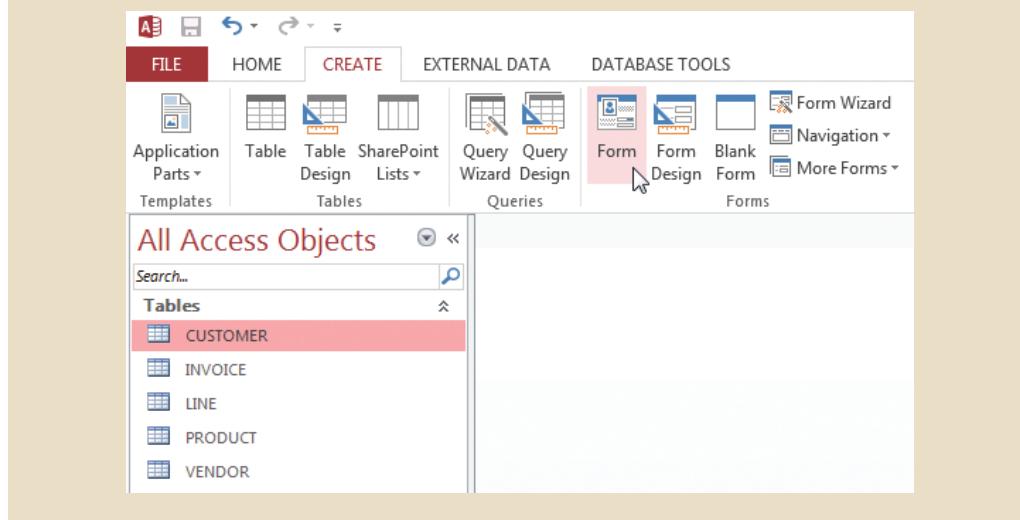


FIGURE M.69 THE ACCESS-GENERATED CUSTOMER FORM

- As you can see by looking at Figure M.69, the form opens up in **Layout View** format. (Given the work you have done with the query development, you should be familiar with the various views.)
- Since the relationship between the CUSTOMER and the INVOICE table has already been established, the form automatically generates an INVOICE subform. Select the **INVOICE** subform and press **Delete**. The selection is confirmed by the selection markers (the orange box around the subform perimeter).
- Save and name the form **frmCUSTOMER** before continuing the form design process.



Note

Here is another example of self-documentation: The **frm** prefix indicates that the object is a form and the capital letters used in the **CUSTOMER** portion indicate that the query data source was a table named CUSTOMER. If the form's data source had been a query named **qryCustomer**, the form name would have indicated the data source through the use of lowercase letters, using "caps" only to separate the components.

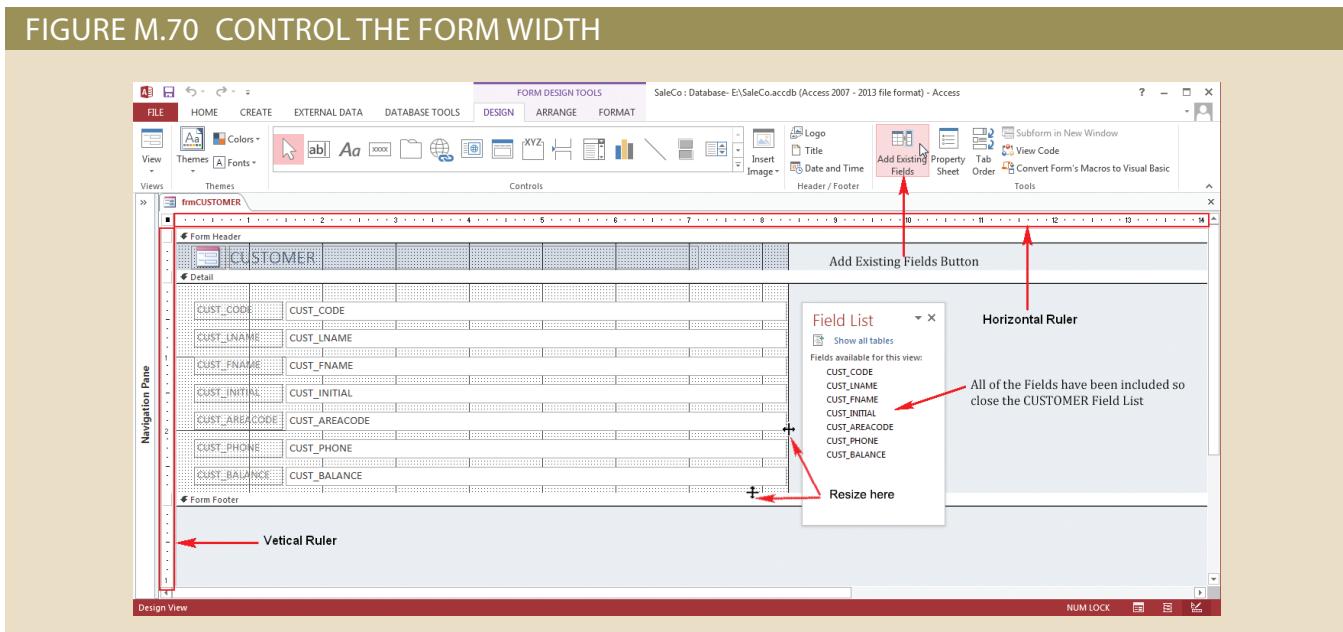
Making it easy to keep track of the many components in a set of database applications is a mark of professionalism. Nobody wins if nobody can figure out what the database objects are or what they do. Documentation conventions should be made in clear in the formal application documentation.

M-7a Editing the Form

While the **frmCUSTOMER** form presents the data properly, it is only a starting point. The form will be enhanced for functionality and appearance reasons (for example, better spacing, customizing labels, text boxes, and so on). It is important that the forms are well designed and functionally accurate to meet the end-user needs. To customize the form you use the **Design View**.

The first time you open the form in Design View format, you may see the Field List window shown in Figure M.70. All the CUSTOMER table's fields have already been included in the form, so go ahead and close this Field List. (If you later want to open this Field List box again, click the **Add Existing Fields** button shown in Figure M.70.)

FIGURE M.70 CONTROL THE FORM WIDTH



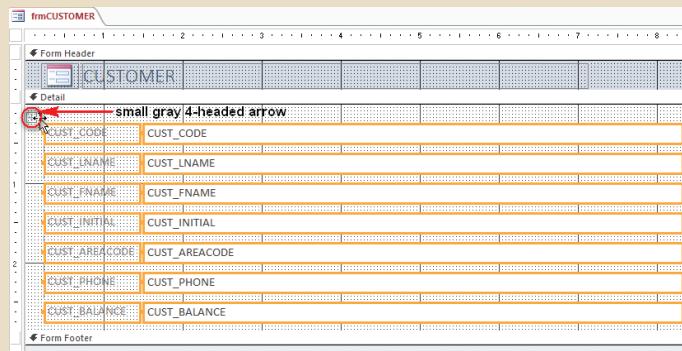
The first thing to notice is that a form has three main areas (see Figure M.70):

- Header area: section on the top of the form containing the title of the form.
- Detail area: contains the form record data—this is where the table data will appear.
- Footer area: section on the bottom of the form that contains text common to all records.

If you want to widen the form, put the cursor on the form's edge to change the cursor to the format shown in Figure M.70, and then drag the form limit to wherever you want it to be. You can control the vertical size the same way. Just put the cursor on the **top edge** of the **Form Footer** and drag up or down to suit your needs. If you put the cursor on the **bottom edge** of the **Form Footer**, you will be able to create a footer area and control its width. (Incidentally, the horizontal and vertical rulers can be used later to help you line up selected output components or to mark multiple selection on the form.)

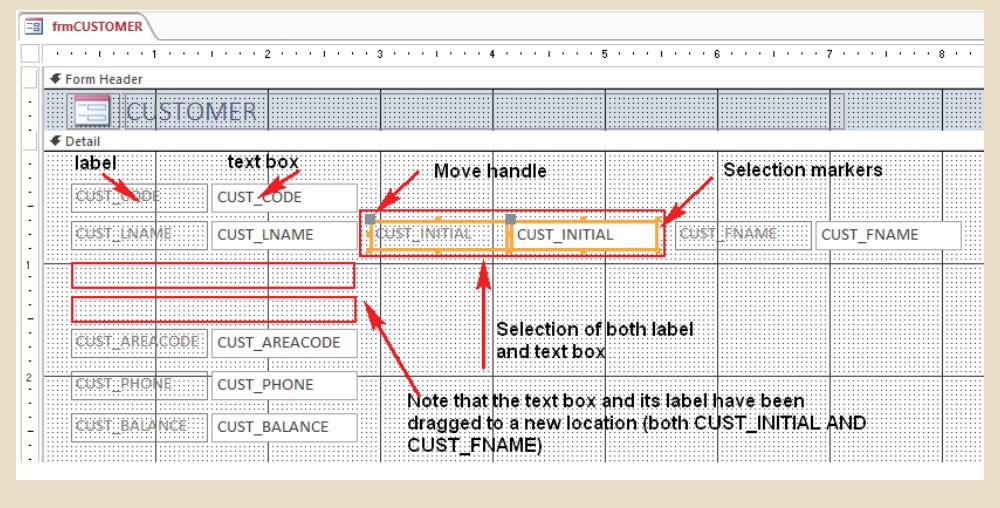
The table fields appear in the Detail area of the form, one record at a time. By default, when using the Form button on the Ribbon the fields are grouped together as a single group. This default layout arrangement for the fields is known as "Stacked." This means that all of the fields are contained in a single layout and resized or moved as one group. To remove this layout, select the one of the fields in the Detail area, and a small gray 4-headed arrow icon will appear in the top left corner. Click this **4-headed arrow** and all of the fields will automatically be selected (see Figure M.71); then select the **ARRANGE** tab and click **Remove Layout**. This allows you to now move and resize the fields individually. Figure M.71 shows how to select all the attributes when they are in the stacked arrangement.

FIGURE M.71 REMOVING THE LAYOUT



Now, you can rearrange the fields and labels in your form. Use Figure M.72 as a guide to rearrange your form fields. Notice that the controls in Figure M.72 have been resized and lined up.

FIGURE M.72 DESIGN COMPONENTS



As you examine Figure M.72, keep the following points in mind as well as the shapes from Table M.4:

- The move handle is the small dark square box situated on the top left of each control.
- Selecting the gray move handle of the text box portion selects only the text box.
- Selecting the gray move handle of the label box selects only the label box.
- To move or resize both the text box and label you will need to select them both, you can click anywhere in the center of the label or the center of the textbox to select both of them. Holding down the CTRL key will allow you to select them both as well or make multiple selections. Figure M.73 also shows other ways to select multiple components.
- To align several fields so that they are along a straight edge, highlight the attributes you wish to align; then on the **FORM DESIGN TOOLS** tab, select the **ARRANGE** tab, select the **Align** button, and then choose the option you want.

FIGURE M.73 MULTIPLE COMPONENT SELECTION

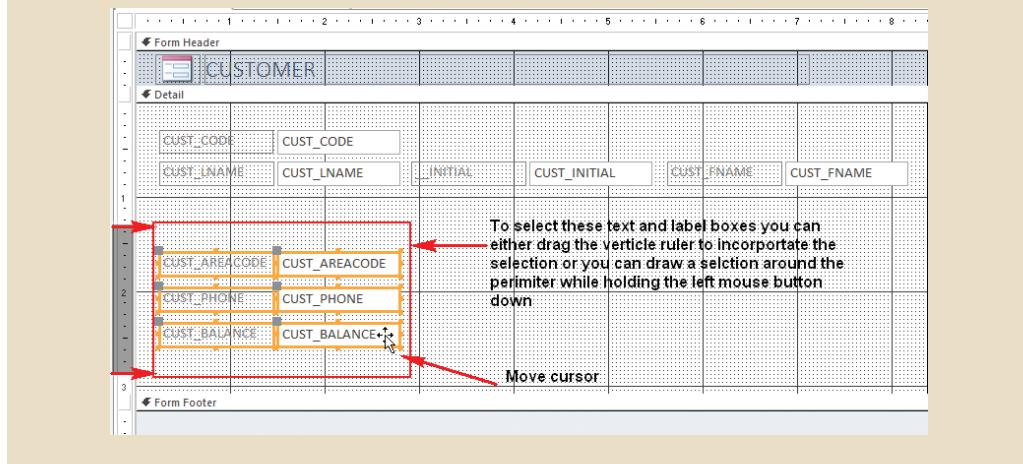


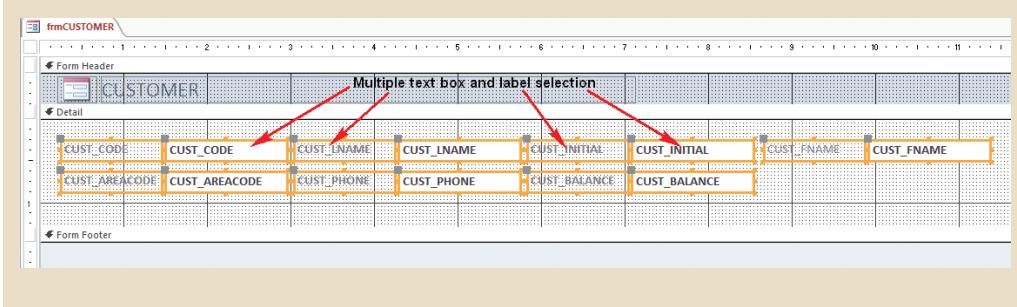
TABLE M.4

MOUSE POINTER SHAPES

SHAPE	WHEN WILL IT SHOW UP?	WHAT DOES IT DO?
	When you point to any control that is not selected on the form	A single click will select a control.
	When the upper left hand corner of a control is pointed to in design view or the middle of a control in layout view. In design view the move handles will appear for you to select the control.	When the control(s) is selected this mouse pointer will drag the control(s).
	When you point to any sizing handle of a control in form or design view	Dragging with this mouse pointer resizes the control.
	When you point to the edge of any form in Design View	Drags the extents of a form to make the form bigger or smaller.

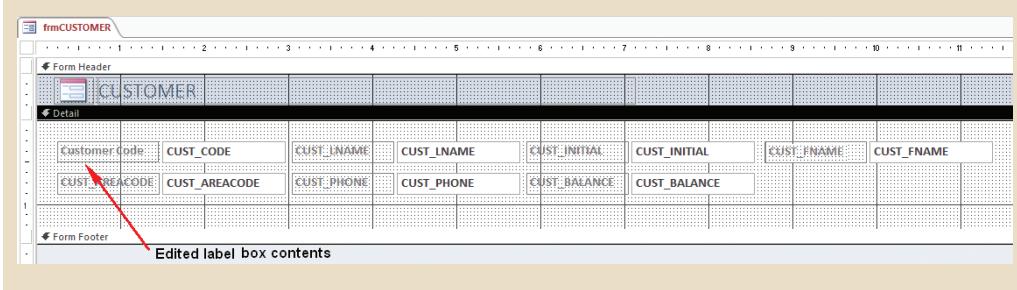
Practice moving the form components around until they match Figure M.74. Then use the same techniques that you learned earlier to change text-formatting properties and to change the font to **Bold** for both labels and text boxes, as shown in Figure M.74.

FIGURE M.74 FONT SET TO BOLD



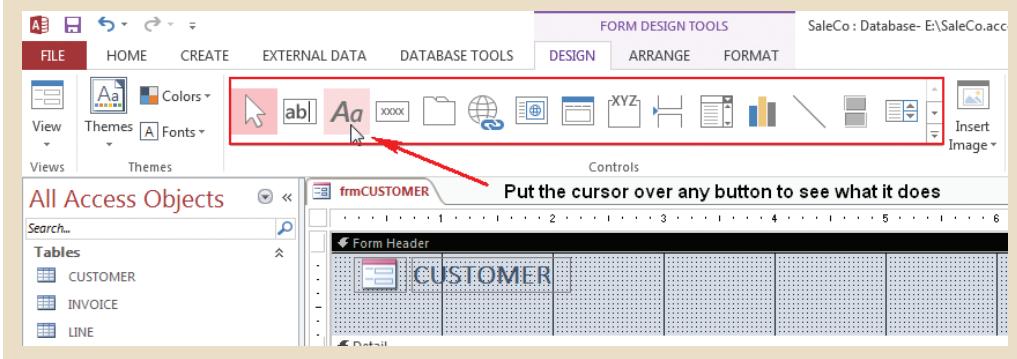
You can also change the label box contents by selecting the label box and then clicking on that selected label box to put it in edit mode. You can then edit the label text. Note that Figure M.75 shows that the CUST_CODE label was changed to **Customer Code**. (Remember that the label box width can be changed by dragging its limits.)

FIGURE M.75 EDITING THE LABEL BOXES



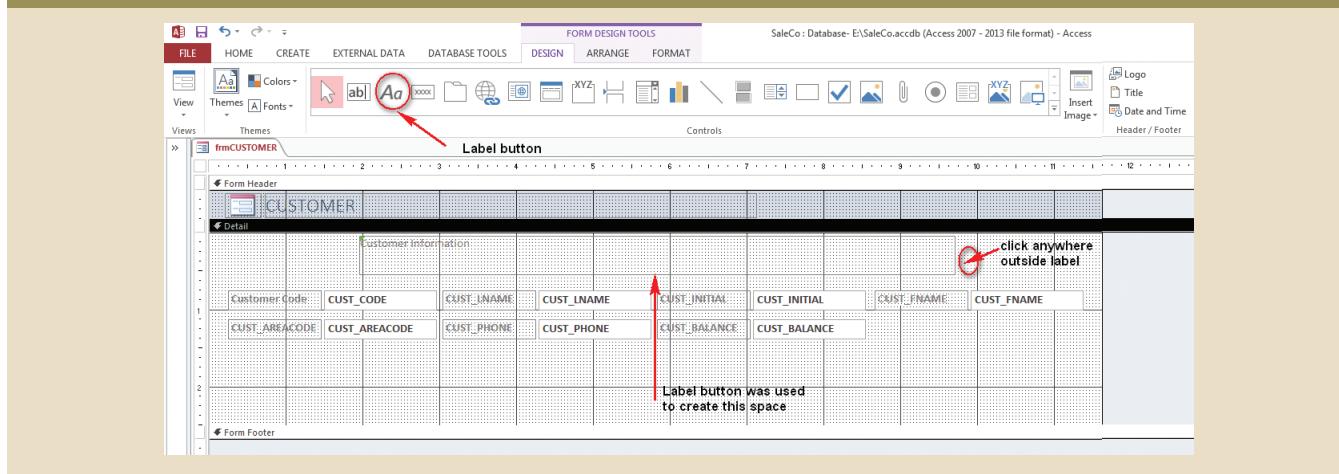
You have a large number of form design tools available. For example, you can add additional text, create boxes to delineate form components, and so on. Click the DESIGN tab to view the **Controls** options you see in Figure M.76.

FIGURE M.76 CONTROLS OPTIONS



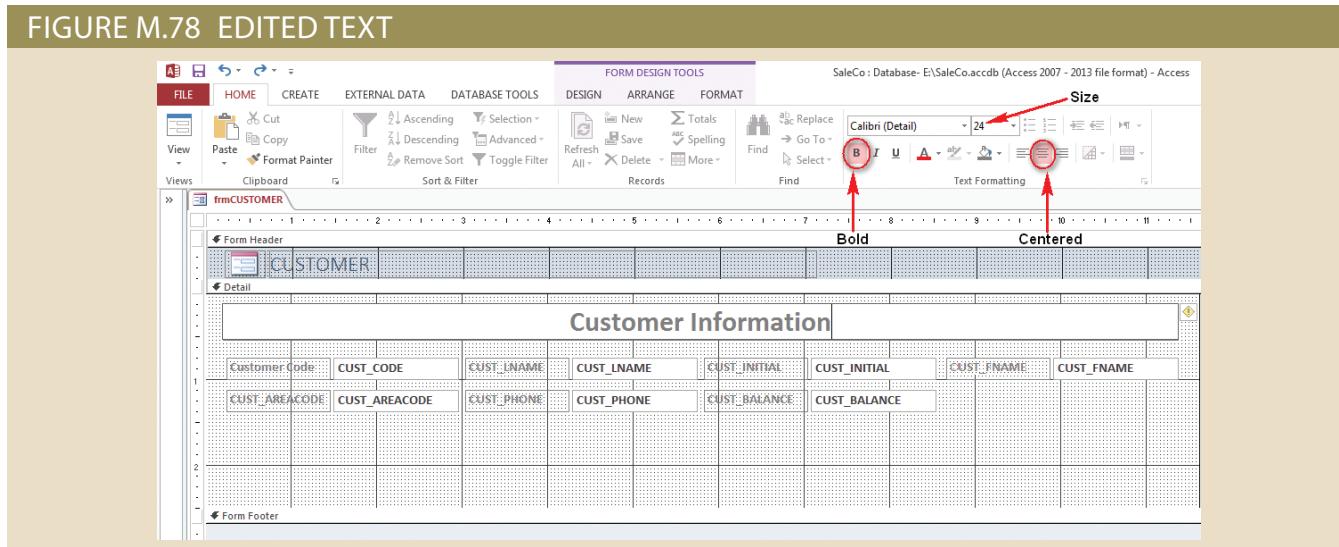
For example, you can add a label to the form. In this case, you will add a label in the Detail area, above the controls in your form. The first step is to create space for the label by selecting all the existing controls in the Detail area and moving them down about half an inch, so the bottom of the first row of controls rest on the one-inch mark. Next, click the **Label** button in the **Controls** group on the **DESIGN** tab (see Figure M.76). Then click and drag on the form Detail area to insert the label control as shown in Figure M.77. If you drag the cursor anywhere on the form, you will create a box in which you can type whatever is needed. In this example, the typed text is **Customer Information**. Click anywhere outside the label to return the cursor to its normal function as shown in Figure M.77.

FIGURE M.77 USING THE LABEL TOOL



You can edit the text as needed. Note that Figure M.78 shows that the font size was changed, the font was bolded, the text was centered, and the label box was resized.

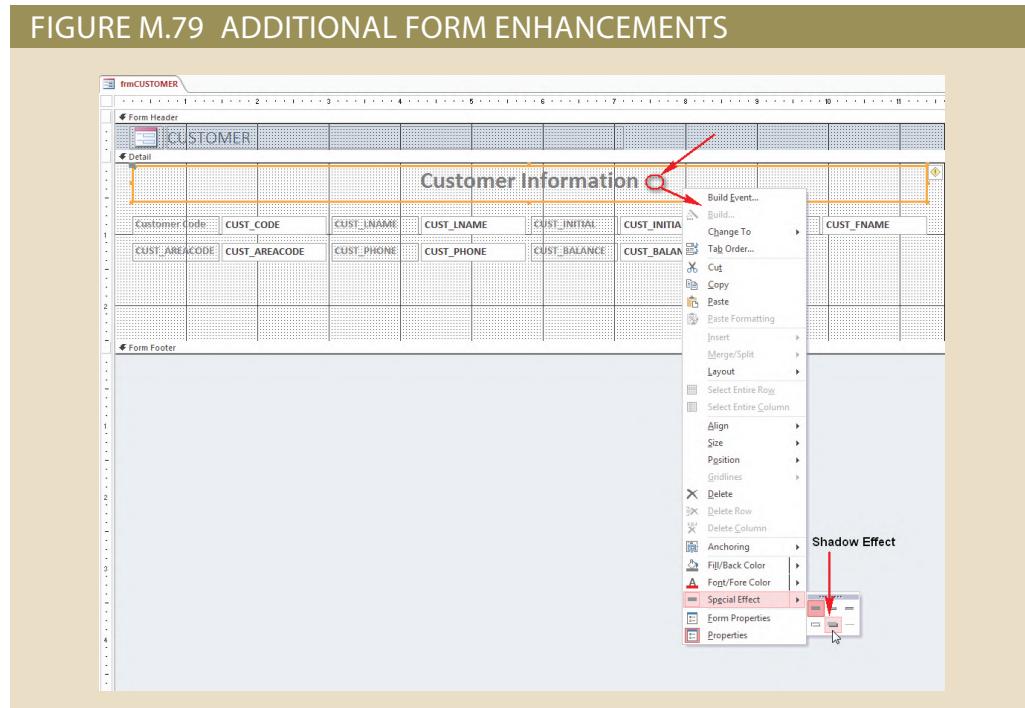
FIGURE M.78 EDITED TEXT



The form's looks can be improved by giving the labels and/or text boxes borders a certain special effect and by adding color. You can use the Rectangle tool to create logical groupings of information presented on the form. For example, let's add the Shadow Special Effect to the Customer Information label box. To accomplish this do the following:

- Select the **Customer Information** label as shown in Figure M.79. (The selection is confirmed by the selection markers—the orange box around the label perimeter.)
- Right-click the selection, the context menu appears.
- Select the **Special Effect** option, and then select the **Shadow** effect (second row, second column) option as shown in Figure M.79.

FIGURE M.79 ADDITIONAL FORM ENHANCEMENTS

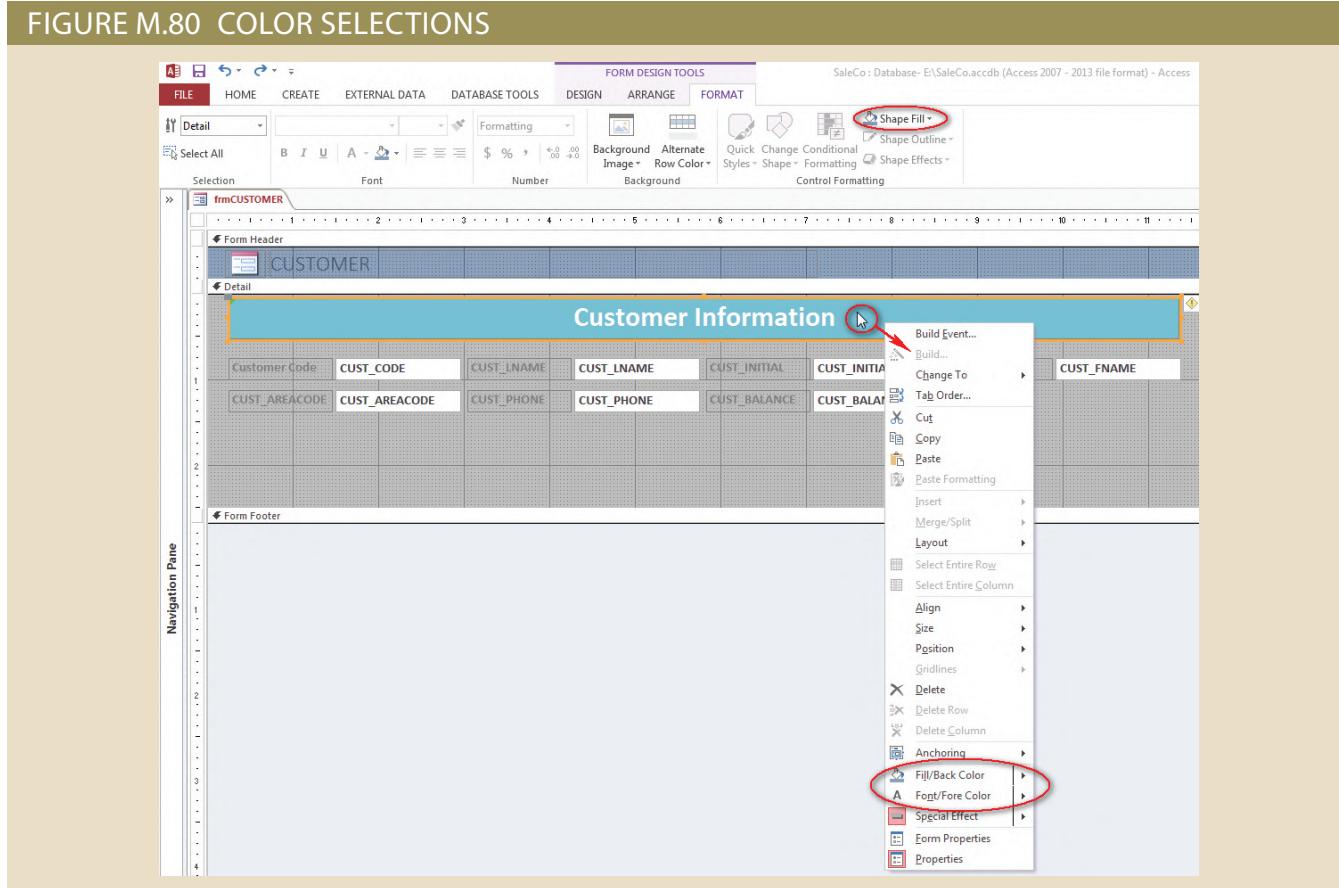


Let's take a look at just a few more form design options. For example, if you look at Figure M.80, you will see that there are quite a few color options available. To match the figure you see here, you can take the following actions:

- To make the form Detail area gray, right-click any empty portion of the Detail section of the form. In the context menu that appears, click the **Fill/Back color** option and click on a gray color square.
- To make the form Header area blue, right-click an empty portion of the header section of the form. On the context menu click the **Fill/Back color** option and click on a blue color square.
- To make the **Customer Information** label background aqua, click the label to select it. Right-click it to open the context menu, click the **Fill/Back color** option and click on an aqua color square.
- To change the font color in the label to white, make sure that the label is still selected, right-click to open a context menu, click the **Font/Fore color** option, and click on the white color square.

Note that Access keeps track of all the recently selected colors to make it easy to later match color selections of other form components.

FIGURE M.80 COLOR SELECTIONS



It is often useful to group logically similar fields together as a visual unit. You can use the Rectangle tool to draw a rectangle around such a group. To get the job done, go to Controls group under the DESIGN tab and select the **Rectangle** button. If you don't see the Rectangle button in the group, scroll to the right of the group to see the additional controls. Then drag a rectangle around the fields you want to group. The rectangle is initially clear and just shows its outline. However, you can right-click the outline of the rectangle and use the **Fill/Back color** option on the context menu to give the rectangle a color. Figure M.81 shows that the selected fill color was light gray. Unfortunately, the default setting on the Rectangle tool places the rectangle in front of the other controls, thus shading them out. However, you can use the **Position/Send to Back** option from the context menu to send the now opaque light gray rectangle to the back, thus making the fields visible again.

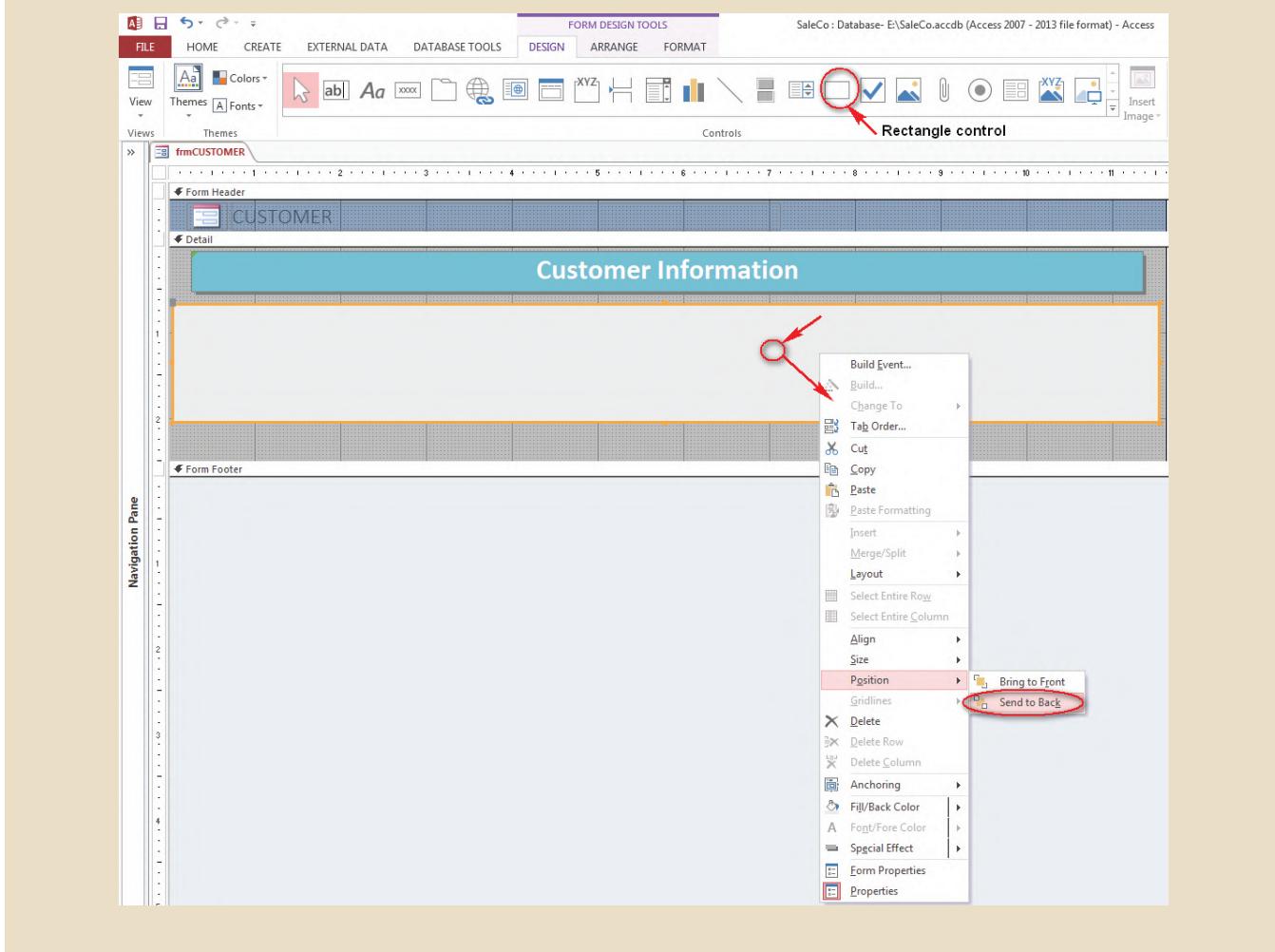


Note

You could also use the **Bring to Front** or **Send to Back** option from **ARRANGE** tab under the **FORM DESIGN TOOLS** menu to position the control where you want.

In addition, you can change the background color of any control by using the **Shape Fill** option on the **FORMAT** tab.

FIGURE M.81 USING THE RECTANGLE TOOL

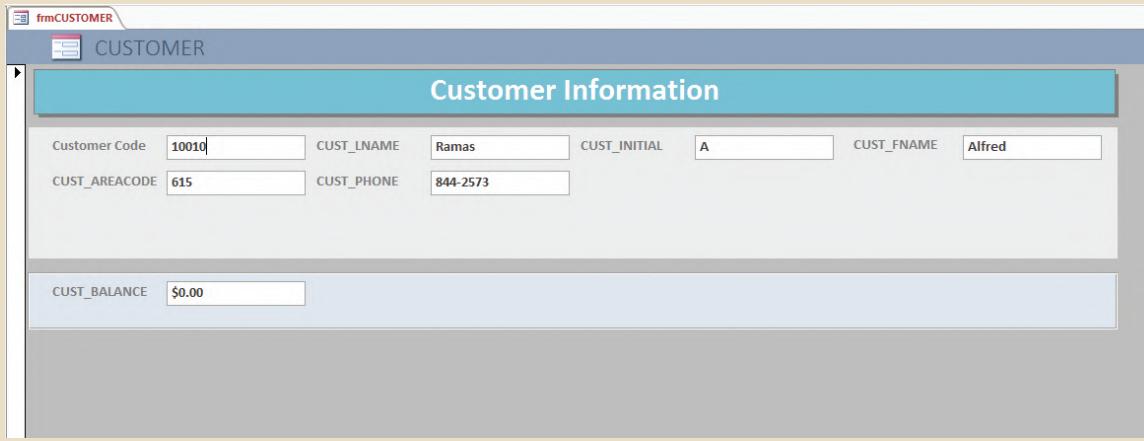


Finally, use the techniques you learned here to make a few more changes:

- Move the **CUST_BALANCE** field, add a rectangle around it, and change its color.
- Drag the **Customer Information** label limits to line up with the two rectangles on the CUSTOMER form.

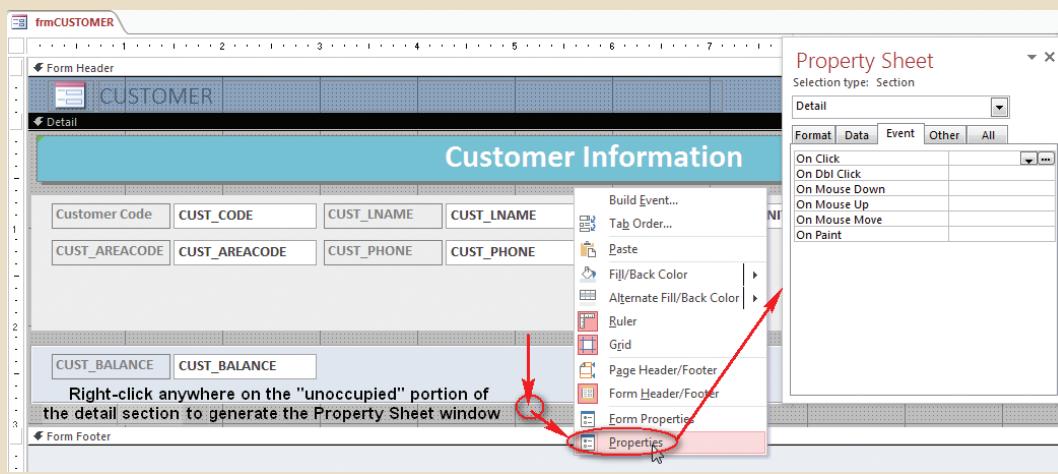
When you are done, open the form in **Form View**. Your form should match Figure M.82.

FIGURE M.82 THE FORM IN FORM VIEW



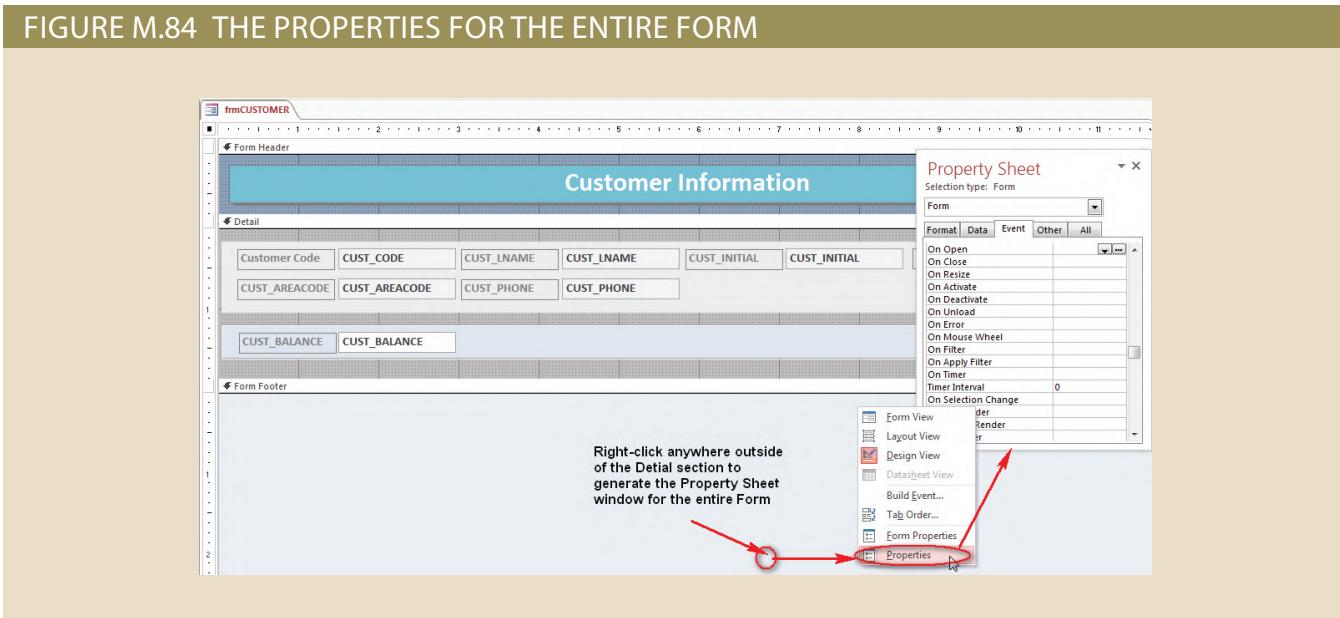
Go back to form **Design View**. Notice that right-clicking on the unoccupied portion of the Detail section of the form in Design View will generate the context menu for the Detail section (see Figure M.83). Once the Properties Sheet is open, you can generate the properties for any Access object by simply clicking on the object. Go ahead and perform this action a few times until it becomes routine.

FIGURE M.83 GENERATING THE PROPERTY SHEET



You can use the context menu to open the Property Sheet window for any Access object. For example, if you want to see the properties of just the CUST_CODE text box, click the CUST_CODE **text** box to select it and then right-click and select Properties to access the Property Sheet window. If you want to see the properties of the Customer Code: **label** box, click the CUST_CODE label to select it and then right-click it and select Properties to access the Property Sheet window. Figure M.84 shows the Property Sheet for the entire form.

FIGURE M.84 THE PROPERTIES FOR THE ENTIRE FORM

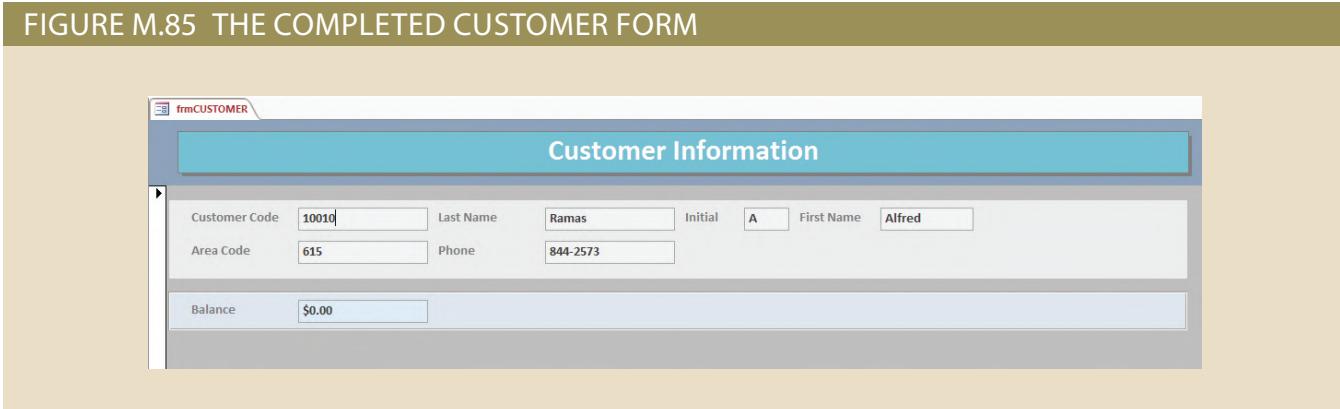


The form is essentially completed. However, you may want to make a few additional changes:

- Put the **Customer Information** label in the Header section of the form. Delete the current content of the Header section. Then, drag and drop the label box to the header. You can create the header space by simply dragging the form's Detail area border down.
- Change the labels on the Detail section, and resize the rectangles, labels, and text boxes to match Figure M.85.
- Finally, you can change the white background for the text boxes to gray using the fill color option shown earlier.
- Save the form and close it.

Figure M.85 shows the final **frmCUSTOMER** form in Form View.

FIGURE M.85 THE COMPLETED CUSTOMER FORM



M-7b Forms Based on Queries

Forms may be used to present data and/or information from multiple sources. Queries are particularly good as a basis for form design, because they can access multiple tables directly.

- Right-click **qryCustomerInvoices** and select **Copy**. Then press **Ctrl+V** to paste the query. Name the new query **qryInvoicesByCustomer**.
- Open it in **Design View**; remove CUST_LNAME, CUST_FNAME, and CUST_INITIAL fields from the query design as shown in Figure M.86. To remove the fields, hover over the top of the CUST_LNAME field; a small black arrow will appear. Once the arrow appears, click and slide the mouse over to select all three fields. Once selected they should all be highlighted black. Click the **Delete** button to delete the selected fields.
- Now we are going to build the new field called **Customer Name** using the Expression Builder as shown in Figure M.86. Note that the new field uses an expression that concatenates the customer's first name, middle initial, and last name. However, because some customers may not have an initial, a logical function must be used. The Immediate If (**IIf**) function uses the following format:

IIf(expression, action if true, action if false)

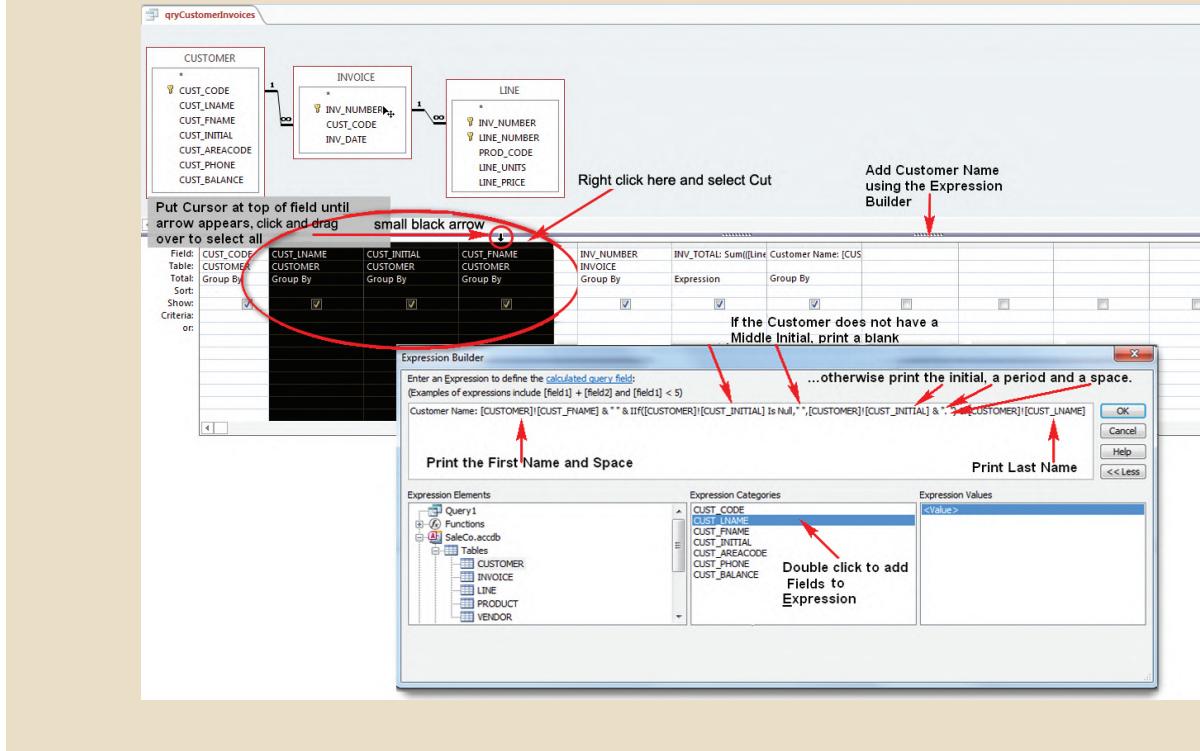
Note that the two possible actions are separated by commas.

- If there is no middle initial – that is, the **CUST_INITIAL Is Null** – a blank space must be printed after the customer's first name. If the customer has a middle initial — that is, the customer initial is not null -- the initial must be printed, followed by a period and space. In either case, the customer's last name must be printed, so the CUST_LNAME is not included in the logical **IIf** statement.
- To build this query, right-click the empty field to the right of the INV_TOTAL field and select **Build**. The Expression Builder window appears. The left side of the window contains the Expression Elements column. Here you can click the plus “+” sign to expand and drill down the elements available for you. For example, you can click the plus sign to the left of the database name to get the Tables, Queries, Forms, and Reports. Keep expanding the tables to see all available tables and then all available fields on a table.
- From the Expression Elements list, expand the tables, then expand the CUSTOMER table and double-click **CUST_FNAME**, then **CUST_INITIAL**, **CUST_INITIAL** once more, and then **CUST_LNAME** to add each attribute needed for the expression.
- Note that when you add the attributes «**Expr**» appears between them, this is not needed so just delete them when they show up.
- Finish the expression by typing the **&**, the blank spaces enclosed by quotes, and the **IIf** function components as shown in Figure M.86. When you are done with the Expression Builder, click **OK** to save the expression. The expression should read as follows:

**Customer Name: [CUSTOMER]![CUST_FNAME] & " " &
IIf([CUSTOMER]![CUST_INITIAL] Is Null," ",[CUSTOMER]![CUST_INITIAL]
& "?")& [CUSTOMER]![CUST_LNAME]**

- Leave the INV_TOTAL as an Expression and change **Customer Name** to **Group By**.
- Finally rearrange the fields so they are ordered as CUST_CODE, INV_NUMBER, Customer Name, INV_DATE, and INV_TOTAL, as shown in Figure M.87. To move the fields simply hover over the top of the field until the black arrow shows up, click, click again, and drag.

FIGURE M.86 THE DESIGN VIEW OF THE INVOICE QUERY



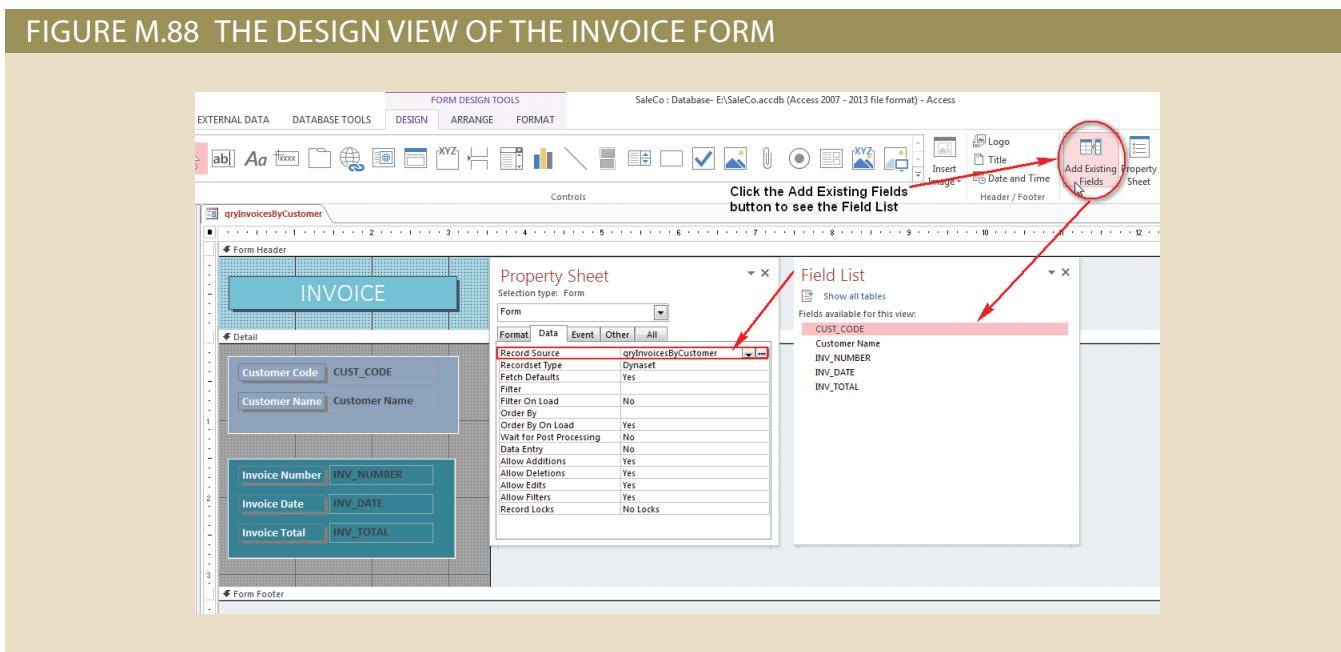
Save the query and then open it in **Datasheet View** as shown in Figure M.87.

FIGURE M.87 THE DATASHEET VIEW OF THE INVOICE QUERY

CUST_CODE	INV_NUMBER	Customer Name	INV_DATE	INV_TOTAL
10011	1002	Leona K. Dunne	16-Mar-18	\$10.78
10011	1004	Leona K. Dunne	17-Mar-18	\$37.66
10011	1008	Leona K. Dunne	17-Mar-18	\$431.08
10012	note a that just a	1003 Kathy W. Smith	16-Mar-18	\$166.16
10014	blank space is	1001 Myron Orlando	16-Mar-18	\$26.94
10014	printed	1006 Myron Orlando	17-Mar-18	\$429.66
10015	1007	Amy B. O'Brien	17-Mar-18	\$37.77
10018	1005	Anne G. Farriss	17-Mar-18	\$76.08

Next, begin the form design just as you did in the previous section. However, this time you use the **qryInvoicesByCustomer** query as the data source. Use the formatting techniques you learned in the previous section to produce the form you see in Figure M.88.

FIGURE M.88 THE DESIGN VIEW OF THE INVOICE FORM



Save the form as **frmInvoicesByCustomer**. Figure M.89 shows the final form view relative to its **Forms** list.

FIGURE M.89 THE FORM VIEW OF THE INVOICE FORM

The screenshot shows the final form view titled 'frmInvoicesByCustomer'. The title bar is red. The main area contains two sections. The top section has a light blue background and displays customer information: 'Customer Code' (10011) and 'Customer Name' (Leona K. Dunne). The bottom section has a dark grey background and displays invoice details: 'Invoice Number' (1002), 'Invoice Date' (16-Mar-18), and 'Invoice Total' (\$10.78).

M-7c Forms with Subforms (Using the Form Wizard)

Forms contribute much to the ability to display data and/or information in a meaningful and visually appealing way. However, from an information management point of view it would be very desirable to see the interaction of related data. Fortunately, information from the LINE, INVOICE, and CUSTOMER tables can be combined by showing the customer and invoice data with the related invoice lines on a single screen through a form/subform combination. In this example, the subform contains the LINE data, while the main form contains the “parent” CUSTOMER and INVOICE data. (In a 1:M relationship, the “M” side would be found in the subform and the “1” side would be seen in the main form.) Go back and study the relationships and notice that the CUSTOMER table is related to the INVOICE table via CUST_CODE and the INVOICE table is related to the LINE table via INV_NUMBER. These relationships must be established for the form/subform to work properly.

In this section you will learn how to generate one form as a subform to another using the Form Wizard. You will also learn some additional techniques that will turn out to be very useful when you try to control input via a form.



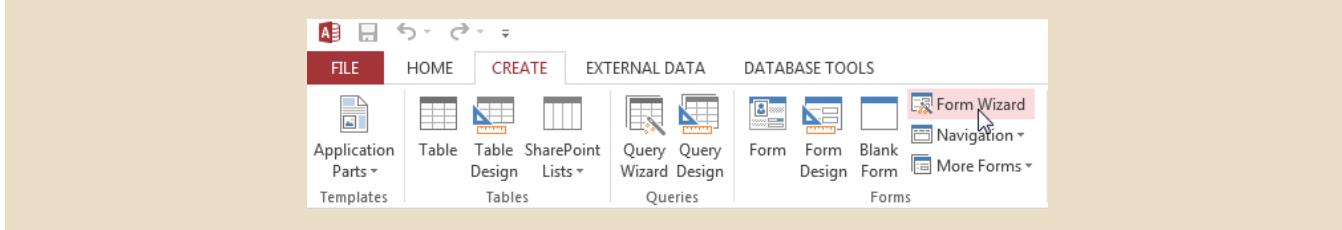
Note

Queries and Reports also have wizards that you can use to quickly generate what you are trying to represent. They all present you with a series of prompts for options that you will need to select in order to customize your query, form, or report.

Let's go ahead and build the **Invoice** form with a line subform using the wizard.

Start by clicking the **Form Wizard** button on the **CREATE** tab as shown in Figure M.90.

FIGURE M.90 SELECTING THE FORM WIZARD OPTION



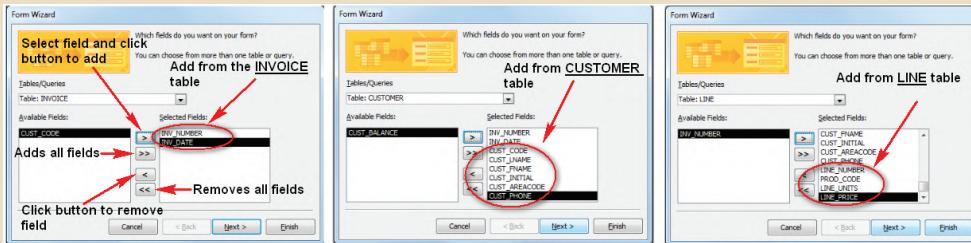
The Form Wizard will take you through a series of four prompts

1. Which fields do you want on your form? (This is shown in Figure M.91.)

Add the following fields in the following order as shown in Figure M.91:

- Select the **INVOICE** table then select the **INV_NUMBER** and **INV_DATE** fields.
- Select the **CUSTOMER** table then select the **CUST_CODE**, **CUST_LNAME**, **CUST_FNAME**, **CUST_INITIAL**, **CUST_AREACODE**, and **CUST_PHONE** fields.
- Select the **LINE** table then select the **LINE_NUMBER**, **PROD_CODE**, **LINE_UNITS**, and **LINE_PRICE** fields.
- Click the **Next** button.

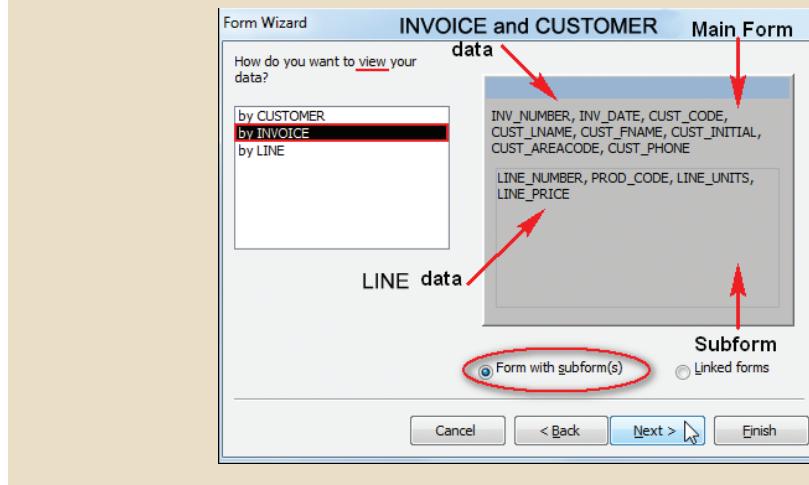
FIGURE M.91 ADDING FIELDS TO THE FORM WIZARD



2. How do you want to view your data?

Select **by INVOICE**, and choose the **Form with subform(s)** option as shown in Figure M.92. This tells the wizard that you want the main form to contain the INVOICE and CUSTOMER data while the subform will contain the LINE data. Due to the fact that we added INVOICE, CUSTOMER, and LINE in that particular order, the Form Wizard knows by default that the main form will contain the INVOICE and CUSTOMER data and the subform will contain the LINE items related to that particular invoice number. If we had added them in a different order then Figure M.92 would look different, so the order is important when using the Form Wizard. Therefore each invoice number with the related customer information is shown in one record with the line items pertaining to that one invoice number. If a customer has multiple invoices then the customer will be shown in multiple records with the corresponding line items. Click **Next**.

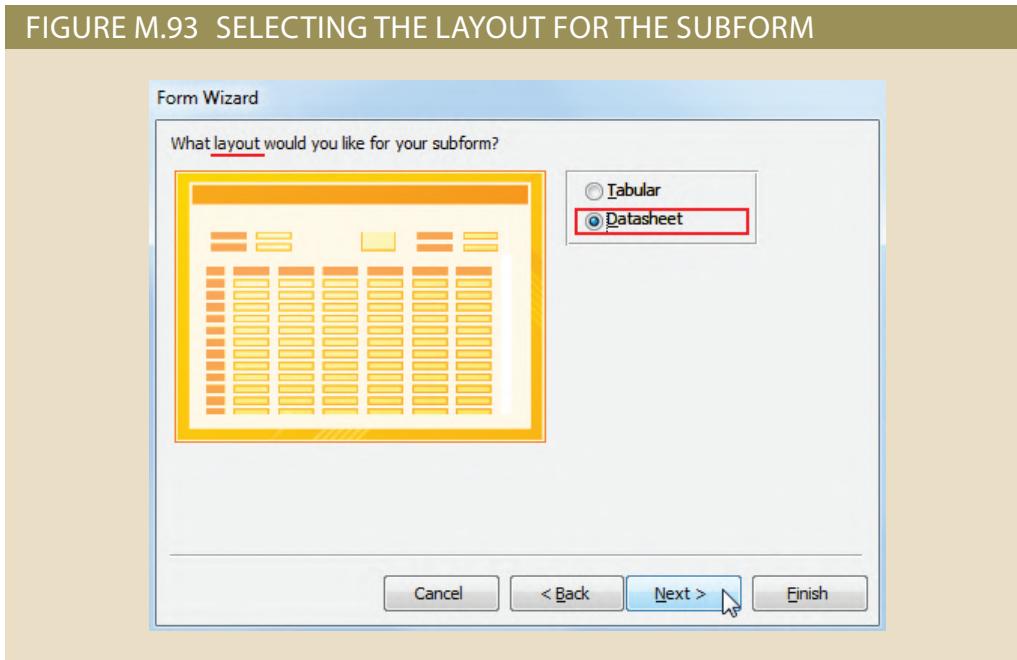
FIGURE M.92 SELECTING THE VIEW FORMAT FOR THE FORM



3. What layout would you like for your subform?

Choose the **Datasheet** option as shown in Figure M.93. This will show the subform in Datasheet View which shows all the line items related to the invoice (it looks similar to a query). Click **Next**.

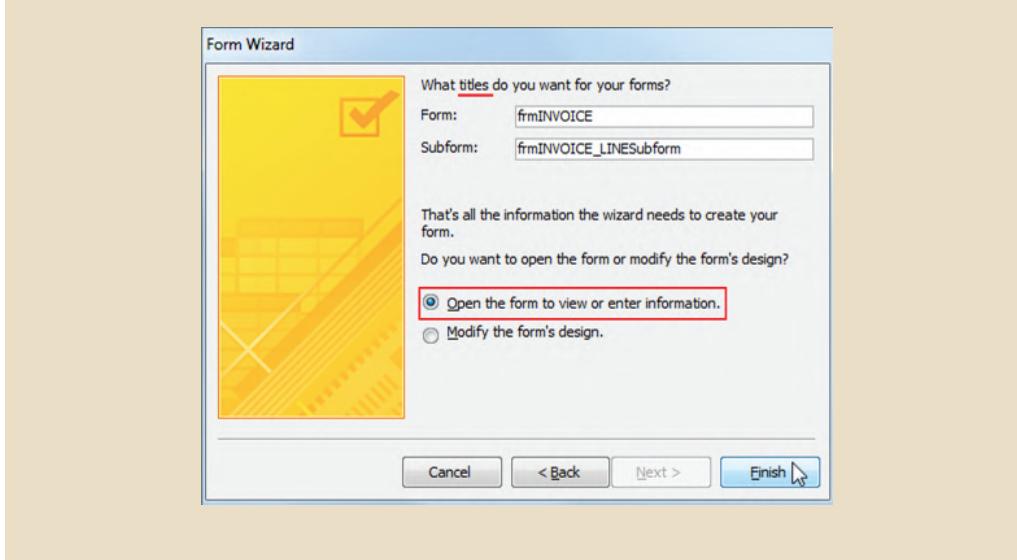
FIGURE M.93 SELECTING THE LAYOUT FOR THE SUBFORM



4. What titles do you want for your forms?

Use the proper naming conventions and name the form **frmINVOICE** and the subform **frmINVOICE_LINESubform** as shown in Figure M.94. Select the **Open the form to view or enter information** option. Click **Finish**.

FIGURE M.94 NAME THE FORM AND SUBFORM

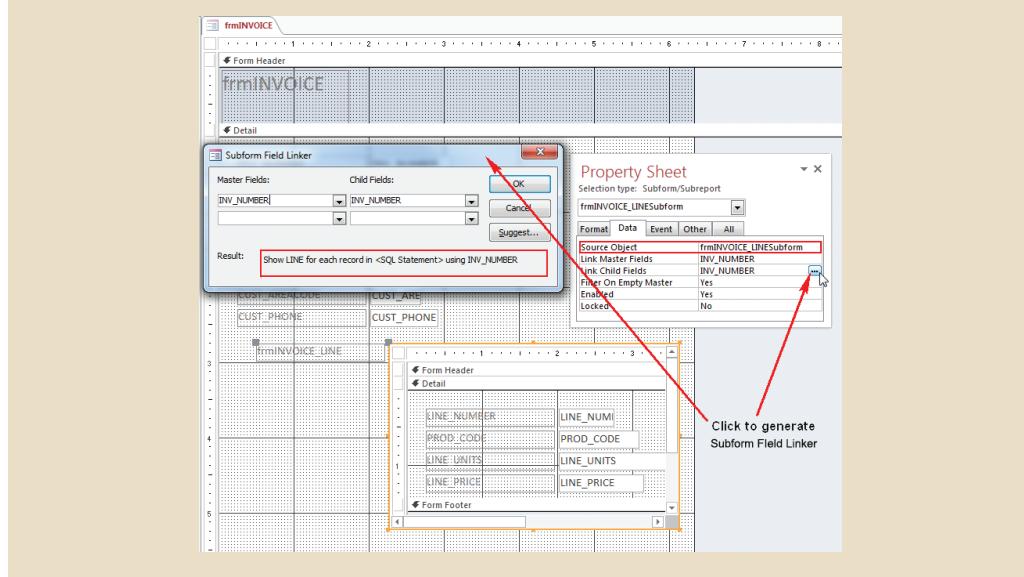


As soon as you finish the Form Wizard prompts, the form and subform appear in Form View. Switch to **Design View** and you will see the generated label for both the form and subform indicating which is which.

Select the subform (an orange border will surround the subform). Right-click on the orange border and select **Properties**. The Property Sheet window for the subform will open. Under the **Data** tab, you will see the **Link Master Field** and **Link Child Fields** properties. These properties tell MS Access how your main form and subform are related (see Figure M.95). In this case, the default values are correct. However, you can change these

settings anytime. For example, click the three dots on the right ([...]) of **Link Child Fields** to produce the Subform Field Linker dialog box you see in Figure M.95. Note that the (correct) link is made through the INV_NUMBER, because the INV_NUMBER is the foreign key (FK) in the LINE table that links the LINE table to the INVOICE table. Because the relationship has already been established and was established correctly, this selection is correct, so just click **OK** to accept it. You are done; that's all there is to it. Close the Property Sheet.

FIGURE M.95 SETTING THE LINK



Next we are going to resize the subform so that we can add a new field. This is the same procedure we used to resize the other two forms; you are just performing it on the subform now. The selection box is the size that the subform will appear in relation to the form and within that selection you can resize the form to fit the new selection size. With the subform still selected, drag the Detail area down just below the 2" mark on the vertical ruler (this is the vertical ruler for the subform, not the entire form). Next drag the entire selection box down just past the 2-1/2" mark. Drag the subform footer down about $\frac{1}{2}$ " as well. You may also need to increase the extents of the original form. Figure M.96 shows the appropriate pointers to use to achieve the proper size. The size of your subform should look similar to Figure M.96.

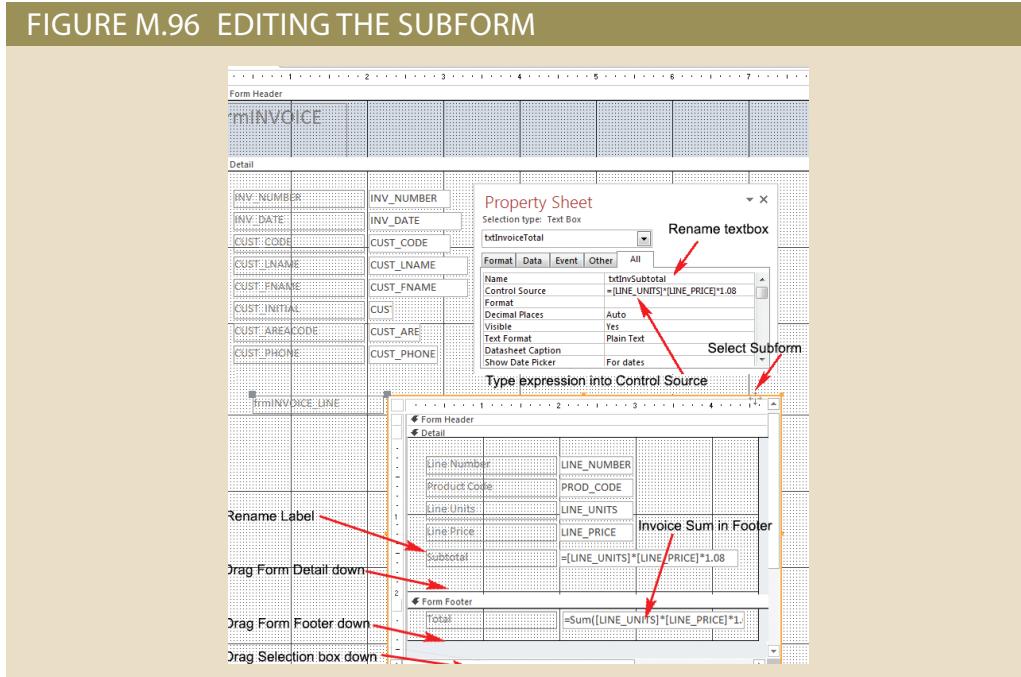
Now we will learn how to use text boxes to add calculations to the form. This is similar to the Expression Builder in query design but instead you will access the Expression Builder through the Control Source on the Property Sheet. This process is demonstrated in Figure M.96.

- Click the **DESIGN** tab, click the **Text Box** button in the **Controls** group, and then click under the LINE_PRICE located inside the subform and draw a new text box. You will see its automatically generated label. Select the label and click the **Property Sheet** button. Click the **All** tab and rename the label to **lblSubtotal** and type **Subtotal** in the Caption box.
- Select the text box. The Property Sheet shows the properties of the text box. Click the **All** tab, and rename the text box **txtInvSubtotal**.
- Click the **Control Source** property. Click the three dots to the right of **Control Source** and build or type the expression **=[LINE_UNITS]*[LINE_PRICE]*1.08**. If you know

the names of the fields and what you are trying to calculate you can simply type it into **Control Source** or the text box itself.

- Create a text box in the footer of the subform. Use the proper naming conventions to give the label a good name and change the caption to **Total**. Name the text box **txtInvTotal**. Datasheet View does not show subform footers in form view so this text box will be hidden. It is important that it goes in the footer of the subform because the subform contains the LINE_UNITS and LINE_PRICE fields. If you were to put this text box in the footer of the main form an error would occur because it will not recognize where the fields are coming from.
- The txtInvTotal text box will compute the total value of the invoice. In the Control Source for this control, enter **=SUM([LINE_UNITS]*[LINE_PRICE]*1.08)**. This will add up all the invoice lines for the invoice lines displayed on the subform. Because the subform displays the lines for one and only one given invoice, the txtInvTotal control represents the total for that given invoice.
- In the main form, create a text box in the bottom of the Main Form just below the subform. Type **=[frmINVOICE_LINESubform].[Form]![txtInvTotal]** into the **Control Source**. This text box will essentially call the value that is stored in the txtInvTotal text box that is located on the INVOICE_LINE subform. It will reflect the Total Invoice for each individual record. Therefore, the label for this text box will use the caption: Invoice Total (see Figure M.97).
- Go to the property sheet for the entire form and click the **Data** tab. Under **Record Source** click the three dots on the right. This will bring up the query that generates the output of the form. Change the **Sort to Ascending** for **INV_NUMBER**. Close and accept the changes.
- Format the form and subform fields the same way you formatted the previous ones.
- Notice that the subform in Datasheet View does not need to be formatted. However, the fields must be presented in a given order. If you need to change the order of fields, switch to Layout View, and change the order of the fields in the subform by selecting and dragging the fields to their proper place. Click the Save button on the Quick Access section of the title bar. This will save the form and the subform field layouts.

FIGURE M.96 EDITING THE SUBFORM



Switch to Layout View or Design View. Delete the frmINVOICE_LINE label and resize the selection box of the subform so that all of the fields are visible. Resize the columns to display the entire headers; modify the layout of the main form fields and labels to match Figure M.97. Note also that there is no INV_NUMBER field in the subform. Why? Is there a need to have the INV_NUMBER field on the subform? The answer is No. When using a main form/subform layout, MS Access automatically updates the linking field value in the subform.



Note

You can only resize the field widths on a datasheet subform in Layout View.

FIGURE M.97 THE FORM-SUBFORM IN LAYOUT VIEW

The screenshot shows the frmINVOICE form in Layout View. The main form has fields for Invoice Number (1001), Invoice Date (16-Mar-14), Customer Code (10014), Last Name (Orlando), Initial (M), First Name (Myron), Area Code (615), and Phone (222-1672). Below these is a subform titled "1st of 2 Invoice Lines" showing two rows of data:

Line Number	Product Code	Line Units	Line Price	Subtotal
1	1 13-02/P2	1	\$14.99	\$16.19
2	2 23109-HB	1	\$9.95	\$10.75
0		0	\$0.00	\$0.00

A tooltip says "Double click to auto-size column header". A red arrow points to the "1st of 2 Invoice Lines" label. Another red arrow points to a "Resize selection box" on the right side of the subform. At the bottom of the subform, there is a footer message: "1st invoice has 2 Lines".

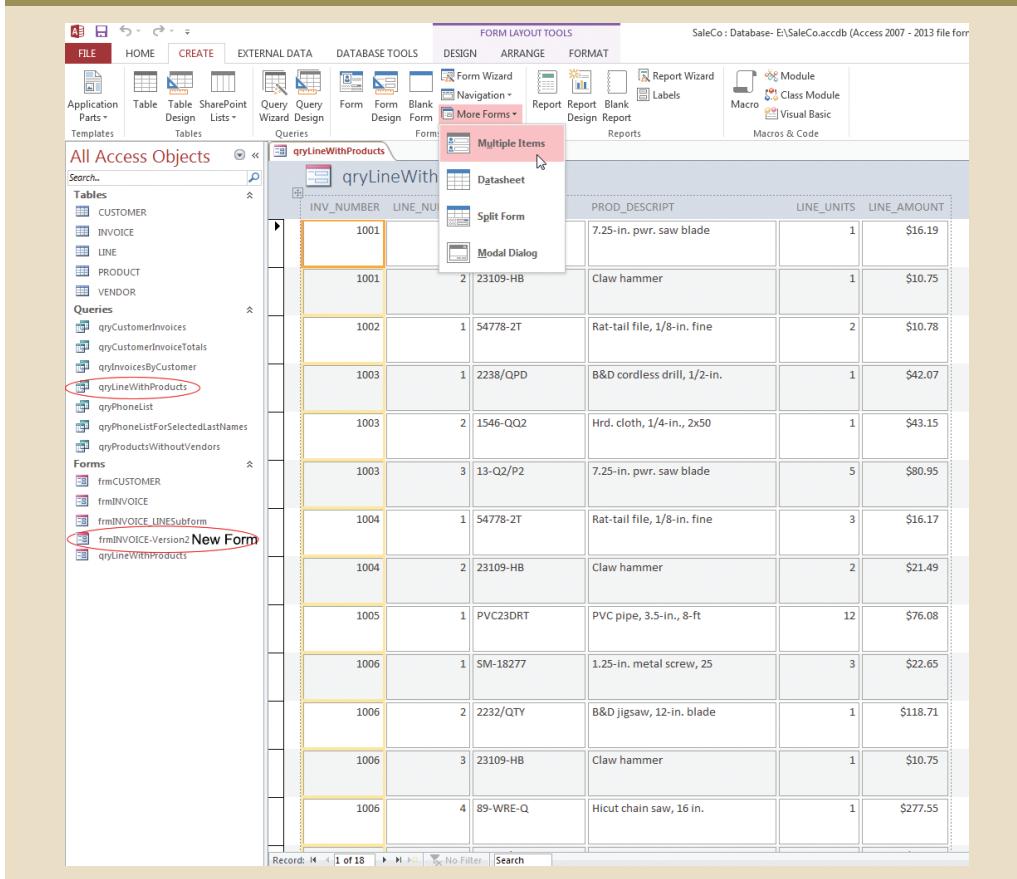
At the bottom of the main form, the status bar shows "Record: 1 of 2" and "Invoice Total: \$26.94". At the very bottom of the screen, another status bar shows "Record: 1 of 8 invoices".

Thus far, you saw how forms could be used as subforms on other forms using data from tables. However, you can also create subforms based on queries. If you already have several calculated or concatenated fields in your query, then it would be easier to use the query for the subform instead of the table.

First make a copy of **frmINVOICE** and name it **frmINVOICE-Version2** so that you can replace the subform that was generated from a table with the new one you are going to make from a query. Import the **qryLineWithProducts** query from the **Ch07_SaleCo** database stored in the student folder using the same technique described in Section M-3.

Select **qryLineWithProducts** from the Navigation Pane, click the **CREATE** tab, and under the Forms group, click **More Forms** and select **Multiple Items**. You will see the output shown in Figure M.98. The **Multiple Items** form produces a form with a tabular output. We have seen the stacked output which was produced by default when we created the frmCUSTOMER and the datasheet output which we chose for our frmINVOICE_LINESubform.

FIGURE M.98 FORM WITH TABULAR OUTPUT



Naturally, you can modify the wizard-produced form, using the design techniques you learned earlier in this section. Save the form as **frmLineWithProducts-TabularView**.

Next, go to **frmINVOICE-Version2** and substitute the **frmINVOICE_LINESubform** with the new form by simply changing the Source Object to **frmLineWithProducts-TabularView** as shown in Figure M.99.

You will need to recreate the text box in the subform footer. Copy over the **txtInvTotal** text box and label from **frmINVOICE_LINESubform**. Then paste it in the new subform footer. The formula is the same and it should work just fine. Subform footers appear in tabular view so the text box and label need to have the Visible property set to **No** as shown in Figure M.99.

Format the form as you did the previous forms, and display the form in **Form View** as shown in Figure M.100. Compare this output with the one shown in Figure M.97. The first figure does not contain the Product Description because we did not use the **PRODUCT** table in the initial Form Wizard.

FIGURE M.99 THE EDITED FORM WITH SUBFORM

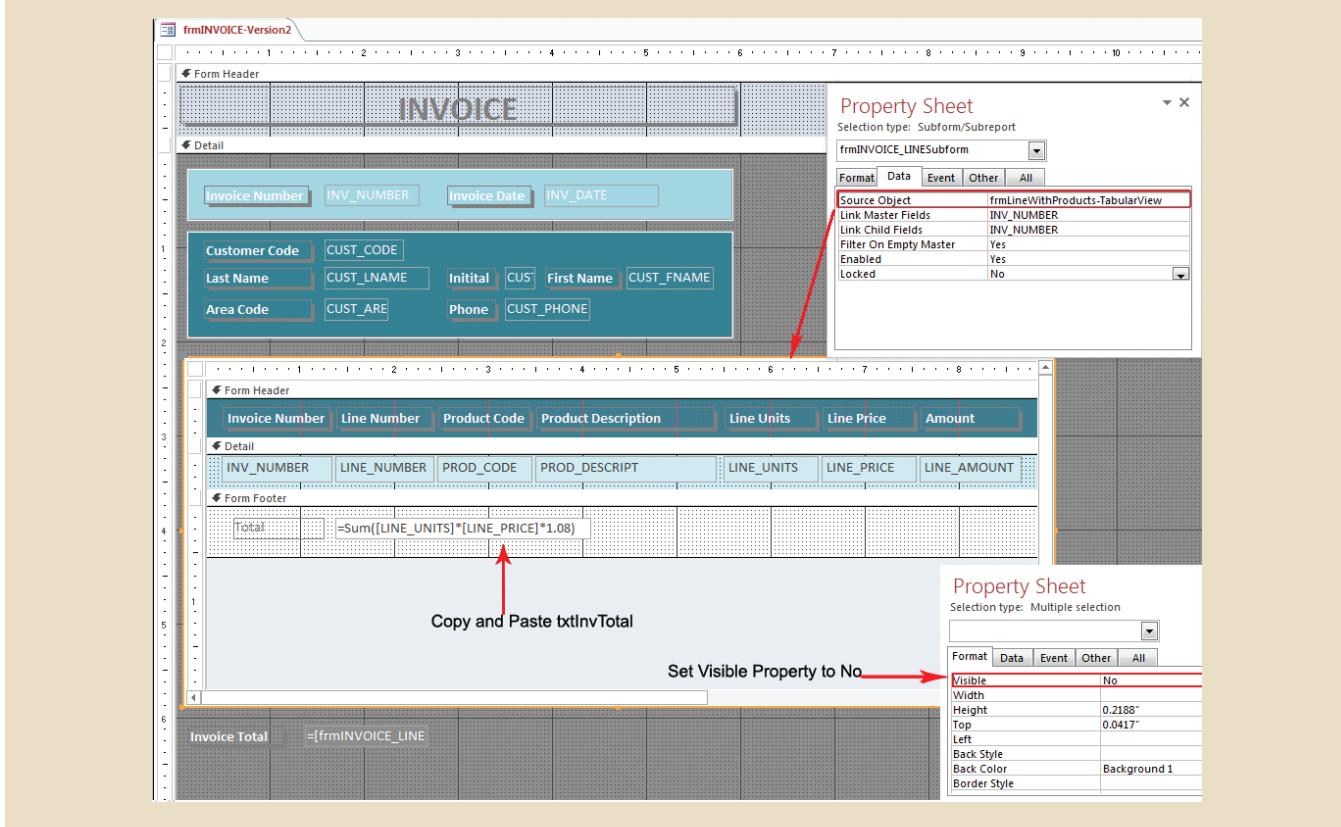


FIGURE M.100 EDITED FORM WITH SUBFORM OUTPUT

M-7d Controlling Input with Combo Boxes

In a real-world environment, the end user is likely to use a scanner to enter product codes and prices. However, when you are prototyping database applications, you probably don't use such "final product" options. So how do you control end user input when it involves something as convoluted as a product code? The answer is simple—just design the form that uses such inputs to include **combo boxes**. A combo box is simply a text box that lists the available input options from which the end user may choose. To create a simple combo box you must define five properties:

- **Row Source** indicates how the list of values in the drop down list is formed and how the list of values that will populate the combo box drop down list is generated. Generally, the list is populated by a query or table rows, but could also be a comma separated list of values. This property works together with the **Row Source Type** property. Therefore, here you indicate either a query or a table name or a list of values.
- **Bound Column** indicates which of the columns in the combo box drop down list contains the value that will be stored in the table's field (indicated in the Control Source property). Remember, this control represents a field in a table; when the user selects a row from the combo box drop down list, one of the column values in the selected row will be stored in this field. This is indicated by the position of the column in the drop down query, for example: 1 will be the first column, 2 will be the second, etc. The Bound Column defaults to the first column.
- **Column Count** indicates how many columns are in the drop down list. This property works together with the Column Widths property explained next. By default only the first column values are displayed on the combo box drop down list.
- **Column Widths** specify the width of each of the columns you indicated in the column count property. You must specify the width (in inches) for each column separated by commas. Here you can make any column "invisible," by indicating a width of 0".
- **List Width** represents the total width of the drop down list. Normally, the sum of all column widths. This width could be wider than the width of the combo box control in the form.



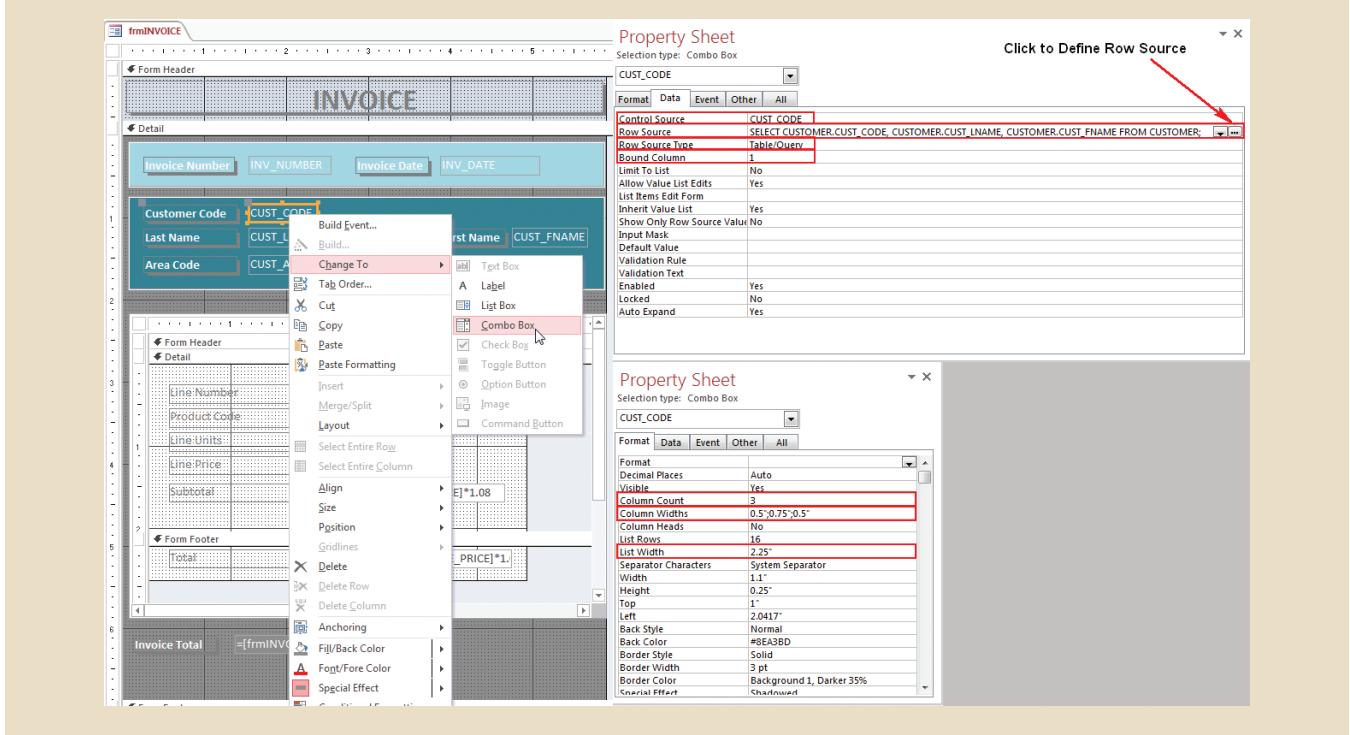
Note

The combo box control will always show a value in the text box. This value is generally the first column of the row source query that is visible and has a width greater than 0. For example, when you click a combo box drop down arrow, the drop down list will show a list of rows, generally from a query or table. The total list width is determined by the List Width property and it could be wider than the combo box text box itself. Once the user selects one row from the drop down list, the first visible column (determined by the Column Widths property) is what is displayed on the combo box text box. However, remember, what is stored in the combo box control source is the value in the column indicated by the Bound Column property.

Microsoft Access makes it easy to convert a text box to a combo box. To illustrate how it is done, open the **frmINVOICE** in Design View (see Figure M.101) and follow the procedure below:

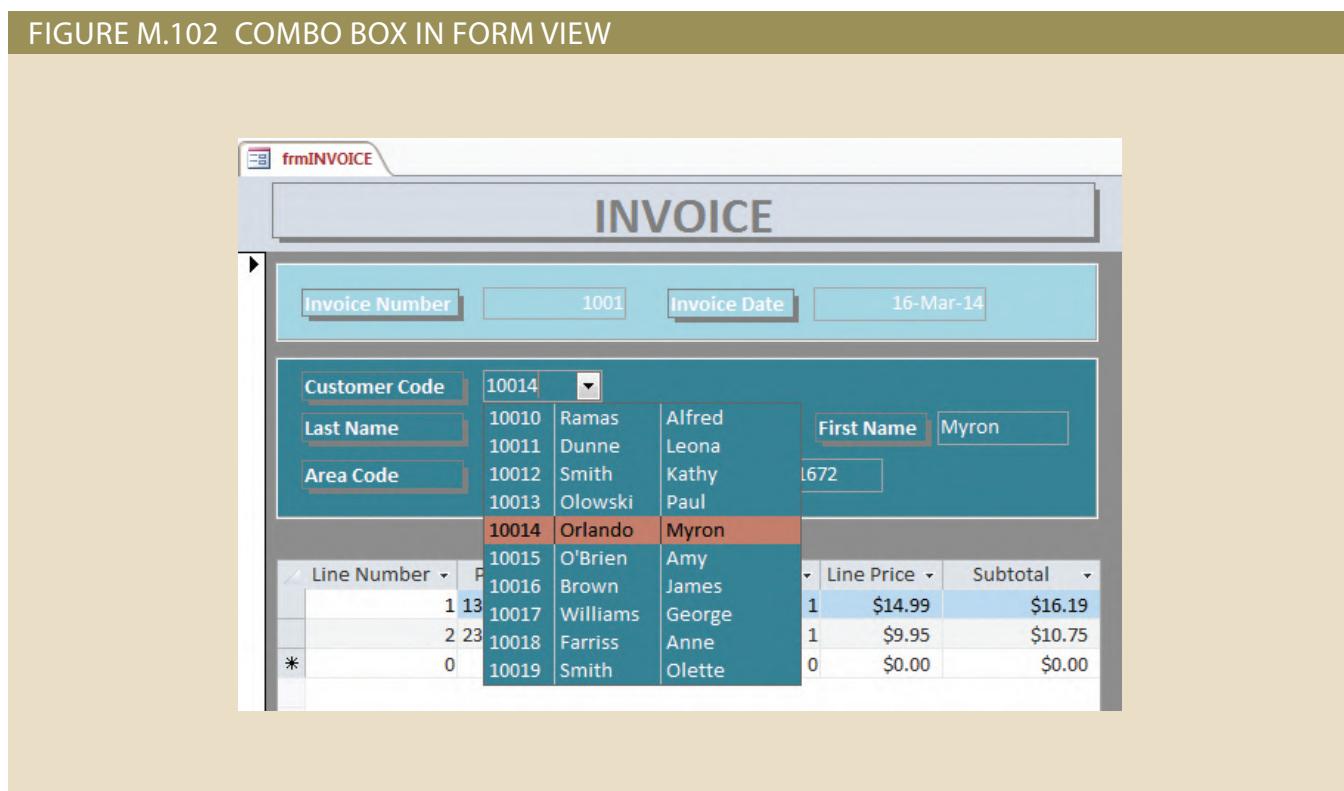
- Right-click the CUST_CODE text box control and select **Change To... Combo Box**.
- Open the Property Sheet for the new Combo Box and select the **Data** tab.
- Click the **Row Source** property, then click the three dots to the right to generate a query design window.
- Add the CUSTOMER table to the QBE data source.
- Add CUST_CODE, CUST_LNAME, CUST, FNAME to the QBE data manipulation grid.
- Close the Query and click **Yes** to save changes.
- From the Form Design View select the Property Sheet for the CUST_CODE Combo Box.
- Note that under the **Data** tab the **Bound Column** property is set to 1. The Bound Column indicates which field from the Row Source query/table will be stored in the Control Source field. Because the Control Source field is CUST_CODE in the INVOICE table, we need to store only valid customer codes. The CUST_CODE field in the Row Source query is the first field; therefore, the **Bound Column** is column 1.
- Select the **Format** tab and change the **Column Count** to 3. (This allows for all three fields to be displayed in the drop down list. By default, the column count is set to 1, which will only display the CUST_CODE column.)
- Type **0.5;0.75;0.5** into the **Column Widths** property. This sets the width for fields 1, 2, and 3.
- Type **2.25** into the **List Width** Property. (This sets the width for the entire drop down list, generally it needs to be .5" larger than the sum of the 3 fields.)

FIGURE M.101 CREATING A COMBO BOX



Save the form. Then open the **frmINVOICE** in its Form View. If you now click the combo box drop-down arrow, you will see the available entries in Figure M.102. Note that you can select a customer based on their code, first, or last name.

FIGURE M.102 COMBO BOX IN FORM VIEW

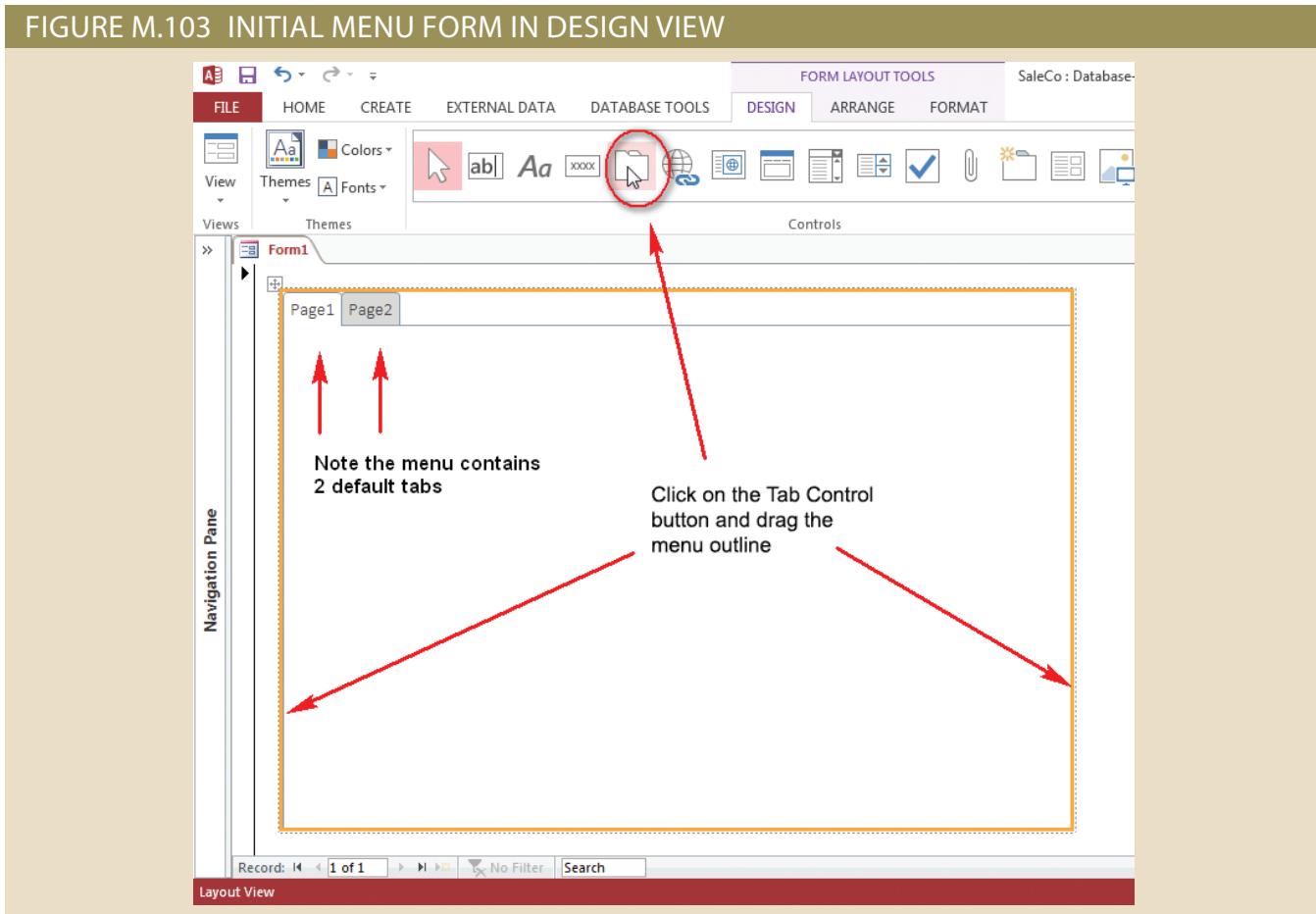


M-7e Menus

If you want to make queries, forms, and reports easily available, a menu is a good way to get the job done. A menu is a form containing buttons to other objects (forms, queries or reports) and it is not bound to any table or query. Creating menus is easy—just click the **CREATE** tab, and then click the **Blank Form** button under the **Forms** group. You will see an empty form in **Layout View** in which to create the menu.

Next, under the **FORM LAYOUT TOOLS**, select **DESIGN**, and under **Controls**, select the **Tab Control** button shown in Figure M.103—note that the cursor changes shape when you do that—and draw the menu outline by dragging the cursor outline on the form. Figure M.103 shows the result of dragging the menu cursor to create a large Tab control on the menu form. Note also that the initial use of the Tab control automatically generates two tabbed menu pages, **Page1** and **Page2**. Right-click on an empty space on **Page1** and select **Insert Page**. This will generate **Page3**. Use the Property Sheet to rename **Page1** to **Queries**, **Page2** to **Forms**, and **Page3** to **Reports**.

FIGURE M.103 INITIAL MENU FORM IN DESIGN VIEW



You are now ready to place a few command buttons on the menu pages. Such buttons will be used to let the end user open a particular object—in this case, a query, a form, or a report. (You will learn about report generation in Section M-8.) Clicking such a button will generate a command that will execute the selected option.



Note

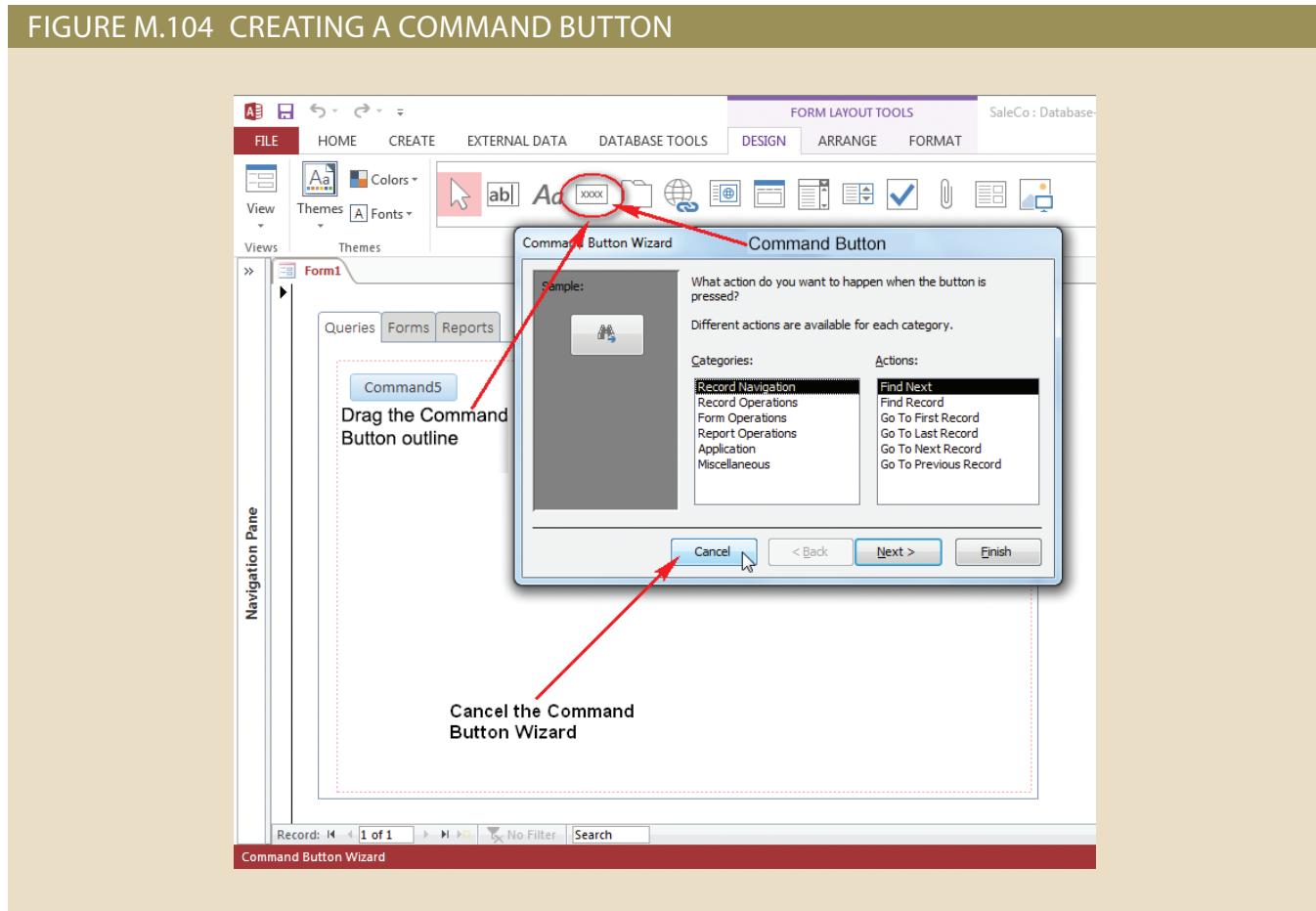
If you do not pay attention to the page you are supposed to be on, you may wind up placing command buttons on all pages simultaneously or you may place a queries command button on a forms or reports page. Therefore, if you intend to produce one or more command buttons on the Queries page, make sure that you select the tab for that page when you are in Design View or Layout View.

Figure M.104 shows how a command button is created on the Queries page. Make sure that the Queries page of the menu form has been selected. Then do the following:

- Click the **Command** button control on the Ribbon.
- Drag the Command button outline onto the page.
- The Command Button Wizard appears.
- Because here you are just creating some buttons, click **Cancel**.

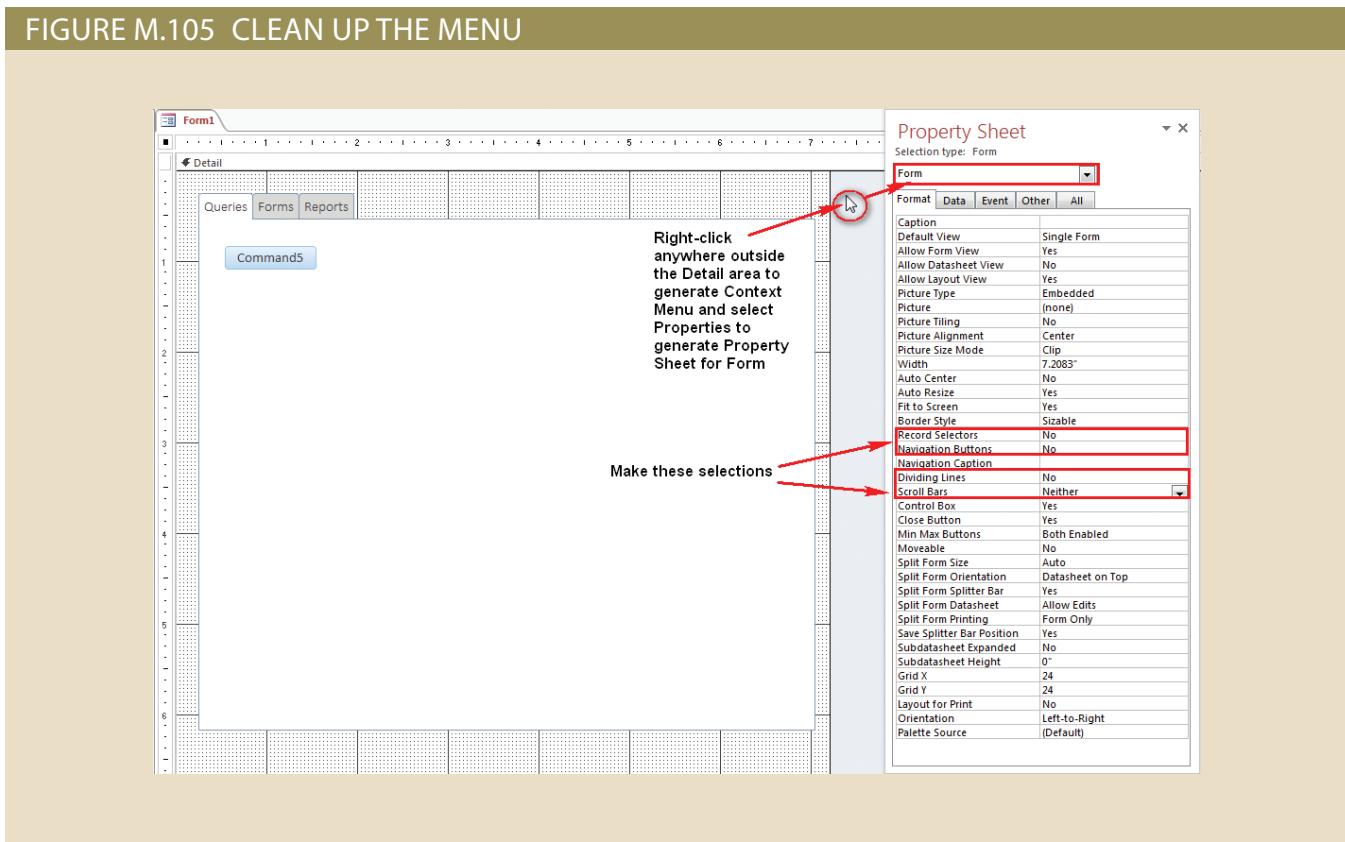
As you can see from Figure M.104, the Command Button Wizard is very easy and self-explanatory. For example, you can follow the prompts and create a button to open a form by clicking Form Operations, Open Form, and then select the form you want to open. It's that easy! To open a query, just click Miscellaneous, then click Run Query, and select the query you want to open. (You will learn how to create and use macros in Section M-9.)

FIGURE M.104 CREATING A COMMAND BUTTON



As you can see in Figure M.104, the menu has a few unwanted properties such as record selector border. Switch to **Design View** to “clean” the menu form by performing the actions indicated in Figure M.105. Save the menu form as **frmMainMenu**.

FIGURE M.105 CLEAN UP THE MENU

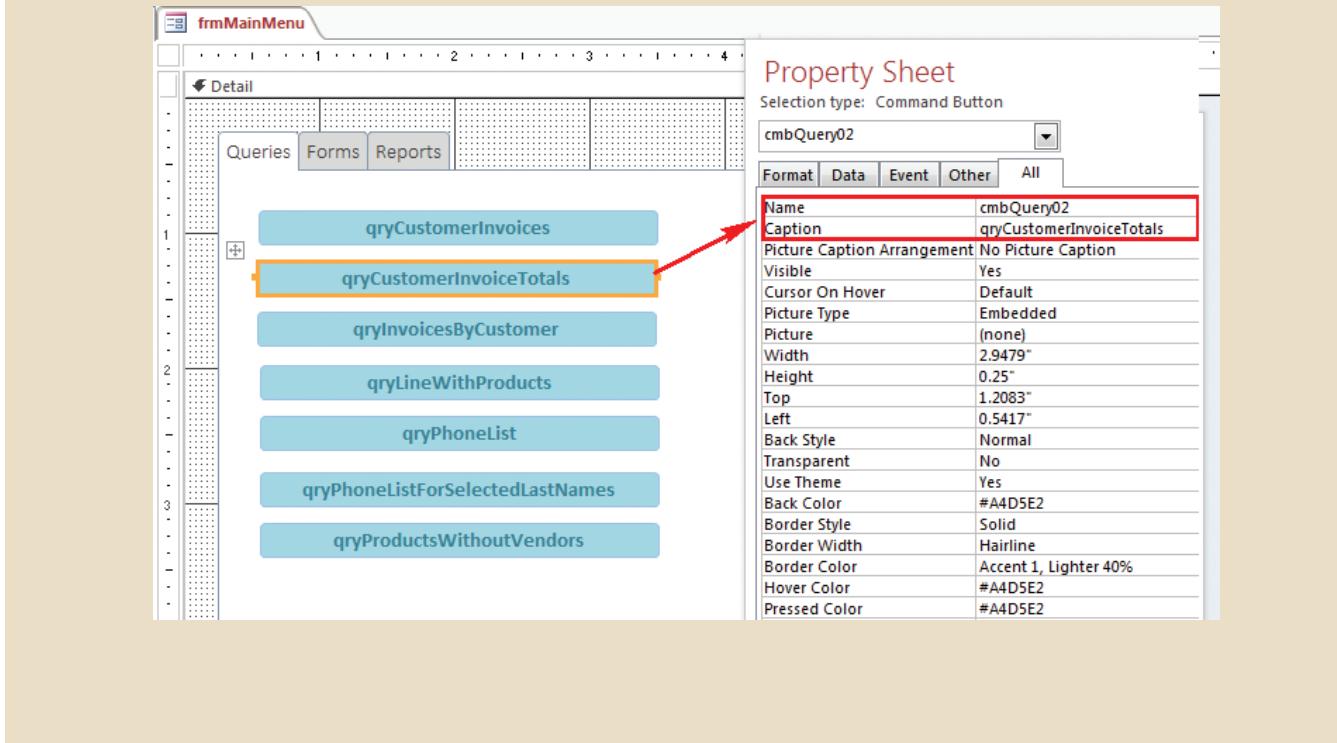


To edit the first command button's name and caption, stay in Design View, right-click the command button, and open the Property Sheet for that button. Change the name to **cmbQuery01** and the caption to **qryCustomerInvoices**.

Change the command button's text to boldface and change the size and location of the command button by dragging its limits. Note that this command button occurs on the Queries page only; the remaining two pages are still blank. (Go ahead and click from tab to tab to see the results.)

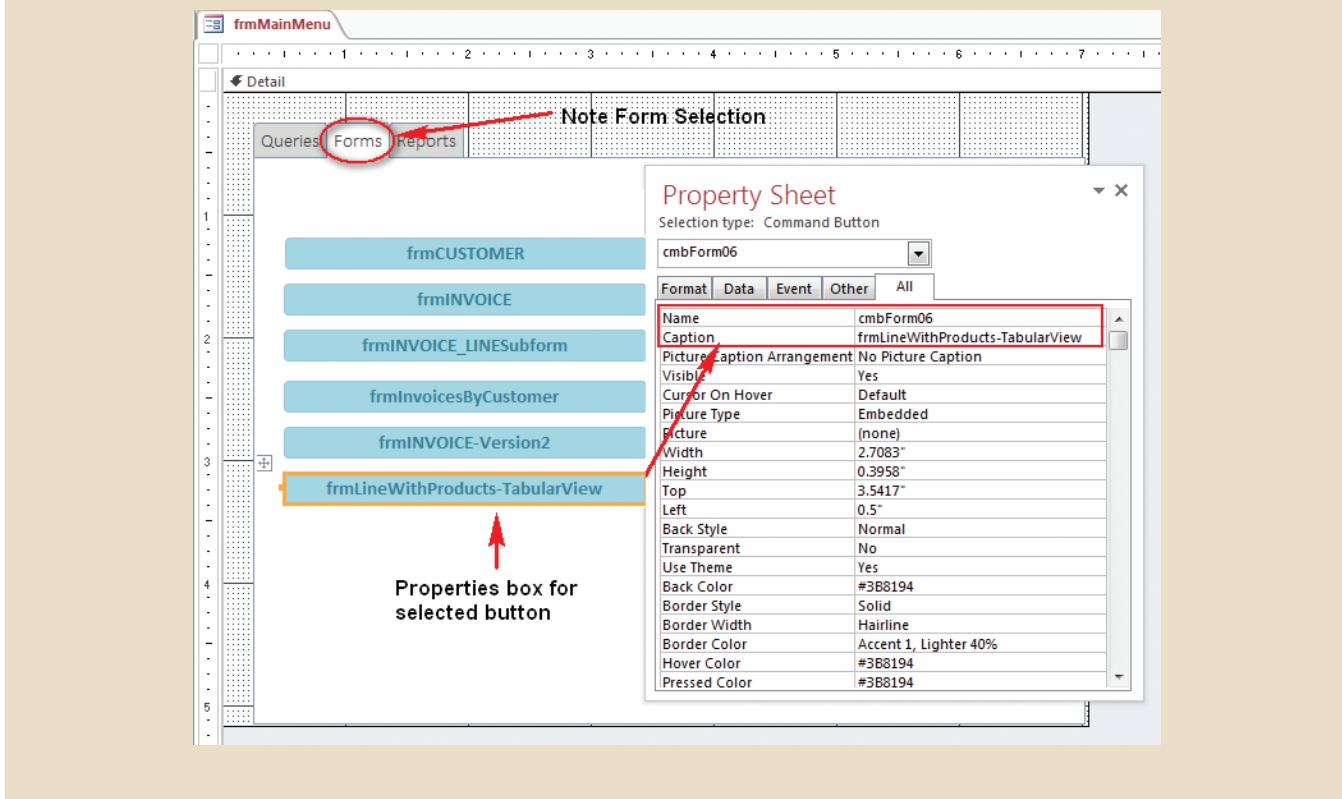
Let's create few more buttons for the **Queries** page. The easiest way to do this is to use a procedure you should know if you are familiar with Windows. That is, select the object in Design View and then use the copy/paste routine to make and place copies. Make sure that that the orange selection box is on the Queries page and not the **entire tab control** (or that the Property Sheet indicates the Queries page). This ensures that the buttons will only be copied to the Queries page. Drag the buttons to their intended positions. Note that the copy/paste routine ensures that all the buttons have the same properties. When you have made and placed the button copies, edit each to match the results in Figure M.106. Don't forget to save your efforts from time to time. You can go to the **FORMAT** tab on the Ribbon and use the existing options to customize the look of the buttons further. Continue to name the buttons as shown in Figure M.106.

FIGURE M.106 MULTIPLE QUERIES PAGE COMMAND BUTTONS



Next, copy the buttons you see here and paste them into the **Forms** page; then edit the buttons to match the forms that will be included. When you are done, your page in Design View should match Figure M.107.

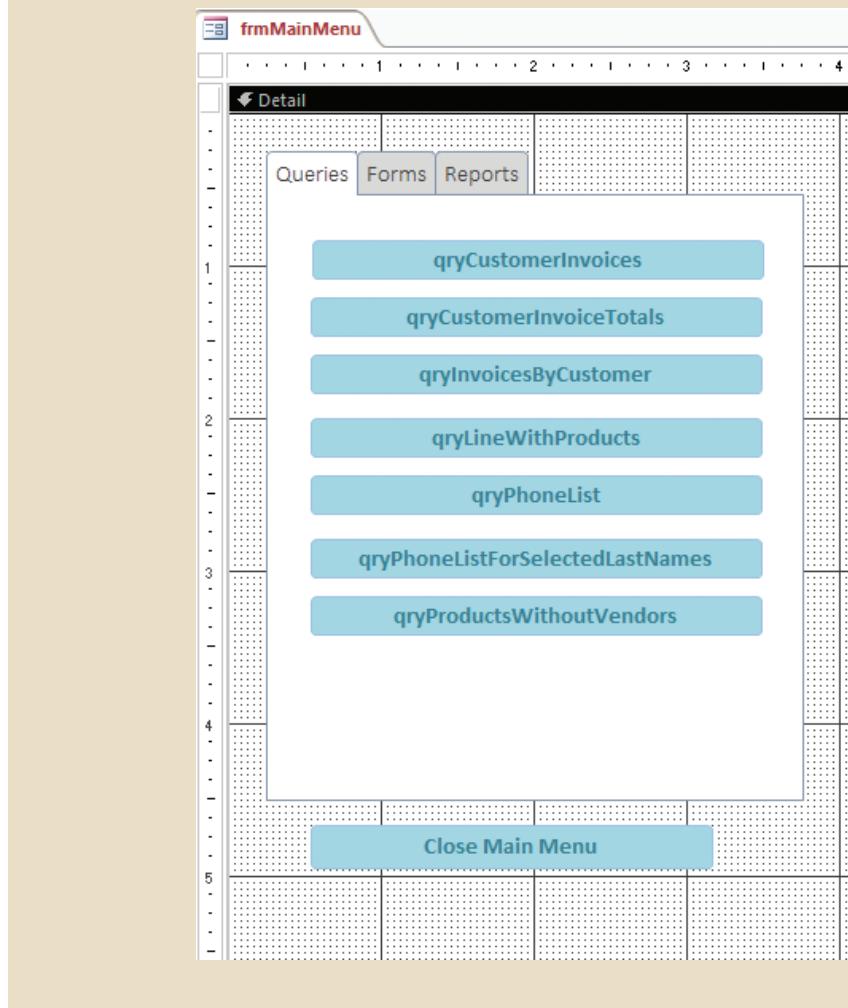
FIGURE M.107 FORM PAGE COMMAND BUTTONS



You have not yet created any reports, so leave the **Reports** page blank. (You will learn how to create reports in Section M-8.)

Next, let's create a **Close Main Menu** command button. Figure M.108 shows that a single button has been created outside of the tabs and therefore, this will make the button accessible from all of the tabs. In this case, when the **Command Button Wizard** appears, you would select **Form Operations**, **Close Form**. Notice also that the form has been resized. Save and close the form.

FIGURE M.108 CLOSE MAIN MENU BUTTON



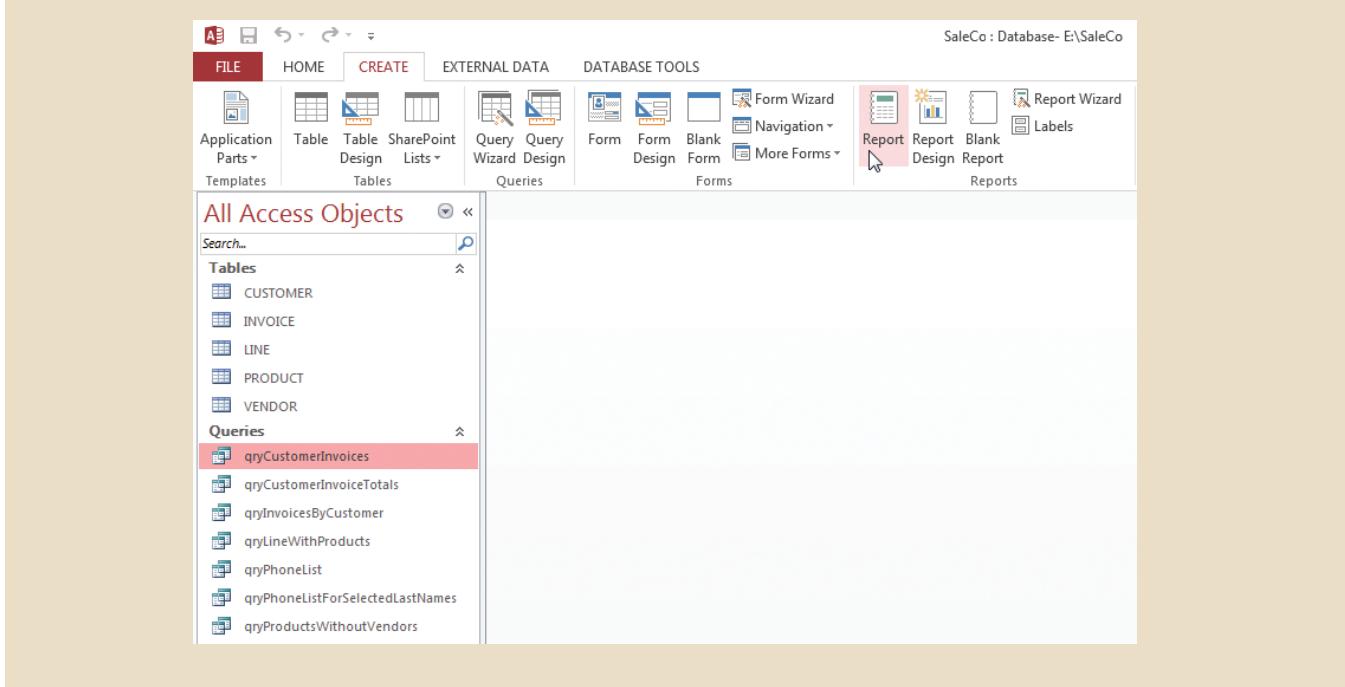
M-8 Reports

Although reports often contain the same information as forms, they do have several advantages. First, it is much easier to show multiple-record information in reports than in forms. Second, given their layout, reports are much easier to print than forms. In addition, if you have a lot of numeric data to present, the Access report format enables you to produce subtotals and grand totals as the report is generated.

Reports can be based on tables and/or queries. The simplest and most efficient way to create a report is to follow the steps outlined below (also shown in Figure M.109). The steps to quickly create a form are the following:

- Select the data source for the report in this case, the query named **qryCustomerInvoices**.
- Click **CREATE** on the Ribbon, then click the **Report** button.

FIGURE M.109 DESIGNING A NEW REPORT



As soon as you click the **Report** button shown in Figure M.109, Access will automatically generate the report and show it in Layout View. You can switch to Report View by clicking **VIEW** and selecting **Report View** on the Ribbon. See Figure M.110.

FIGURE M.110 NEW REPORT OUTPUT

CUST_CODE	CUST_LNAME	CUST_FNAME	CUST_INITIAL	INV_NUMBER	INV_DATE	INV_TOTAL
10011	Dunne	Leona	K	1002	16-Mar-18	\$10.78
10011	Dunne	Leona	K	1004	17-Mar-18	\$37.66
10011	Dunne	Leona	K	1008	17-Mar-18	\$431.08
10012	Smith	Kathy	W	1003	16-Mar-18	\$166.16
10014	Orlando	Myron		1001	16-Mar-18	\$26.94
10014	Orlando	Myron		1006	17-Mar-18	\$429.66
10015	O'Brien	Amy	B	1007	17-Mar-18	\$37.77
10018	Farriss	Anne	G	1005	17-Mar-18	\$76.08

8

Page 1 of 1

Although the report output shown in Figure M.110 is already quite usable, you could add functionality and better formatting using the now-familiar design tools. However, before you start editing the report, save the report as **rptCustomerInvoices**.

To make this report more useful, we will modify the report to include invoice subtotals per customer. In this case, to do subtotals by customer, we need to group the report by customer code and then show invoice subtotals *for each customer*. To achieve this goal follow the steps described next.

- Switch the report to **Layout View**.
- The REPORT LAYOUT TOOLS menu appears on the Ribbon.
- Select and delete the **Record Count** text box under the CUST_CODE column.
- Click the first **CUST_CODE** value on the report area. The entire column of customer code values is highlighted with an orange border.
- Right-click the column value and select **Group On CUST_CODE** from the context menu. See Figure M.111. This option will create a grouping by CUST_CODE and will add a header area at the beginning of each group.
- Move the **CUST_LNAME**, **CUST_INITIAL**, and **CUST_FNAME** fields from the detail area to the CUST_CODE group header area.

Next, we need to add the subtotals for the invoices:

- Click the **first INV_TOTAL value** on the report area. The entire column of invoice total values is highlighted with an orange border.
- Right-click the column value, select **Total INV_TOTAL** from the context menu and then **SUM**. This option automatically adds a footer area with subtotal line after the last invoice line for each customer.

Finally, we are going to add a sort order by INV_NUMBER (within each customer grouping):

- Click the first INV_NUMBER value on the report area. The entire column of invoice number values is highlighted with an orange border.
- Right-click the column value, and select the option **Sort Smallest to Largest** from the context menu.

FIGURE M.111 ADDITION OF A GROUP HEADER

The screenshot shows a Microsoft Access report titled "qryCustomerInvoices". The report displays data from a table with columns: CUST_CODE, CUST_LNAME, CUST_FNAME, CUST_INITIAL, INV_NUMBER, INV_DATE, and INV_TOTAL. The data is grouped by CUST_CODE. A context menu is open over the first row (CUST_CODE 10011), with the "Group On CUST_CODE" option highlighted. Other options in the menu include Cut, Copy, Paste, Paste Formatting, Insert, Merge/Split, Layout, Select Entire Row, Select Entire Column, Total CUST_CODE, Sort A to Z, Sort Z to A, Clear filter from CUST_CODE, Text Filters, Equals "10011", Does Not Equal "10011", Contains "10011", Does Not Contain "10011", Delete, Delete Row, Delete Column, Change To, Position, Gridlines, Anchoring, Report Properties, and Properties. At the bottom of the report, there are buttons for "Add a group" and "Add a sort". The status bar at the bottom right shows "Page 1 of 1".

CUST_CODE	CUST_LNAME	CUST_FNAME	CUST_INITIAL	INV_NUMBER	INV_DATE	INV_TOTAL
10011	Dunn	Leona	K	1002	16-Mar-18	\$10.78
10011	Dunn	Leona	K	1004	17-Mar-18	\$37.66
10011	Dunn	Leona	K	1008	17-Mar-18	\$431.08
10012	Kathy		W	1003	16-Mar-18	\$166.16
10014	Myron			1001	16-Mar-18	\$26.94
10014	Myron			1006	17-Mar-18	\$429.66
10015	Amy		B	1007	17-Mar-18	\$37.77
10018	Anne		G	1005	17-Mar-18	\$76.08

Now that you have added grouping and totals to the report, the updated report now has CUST_CODE header and footer areas. To complete the report, go to **Design View** and do the following:

- Add a **Customer Total:** label to the subtotal in the CUST_CODE footer and a **Grand Total:** label to the Report Footer area. See Figure M.112.
- Move the page number components from the Page Footer area to the Report Header area. See Figure M.112.

Save the report.

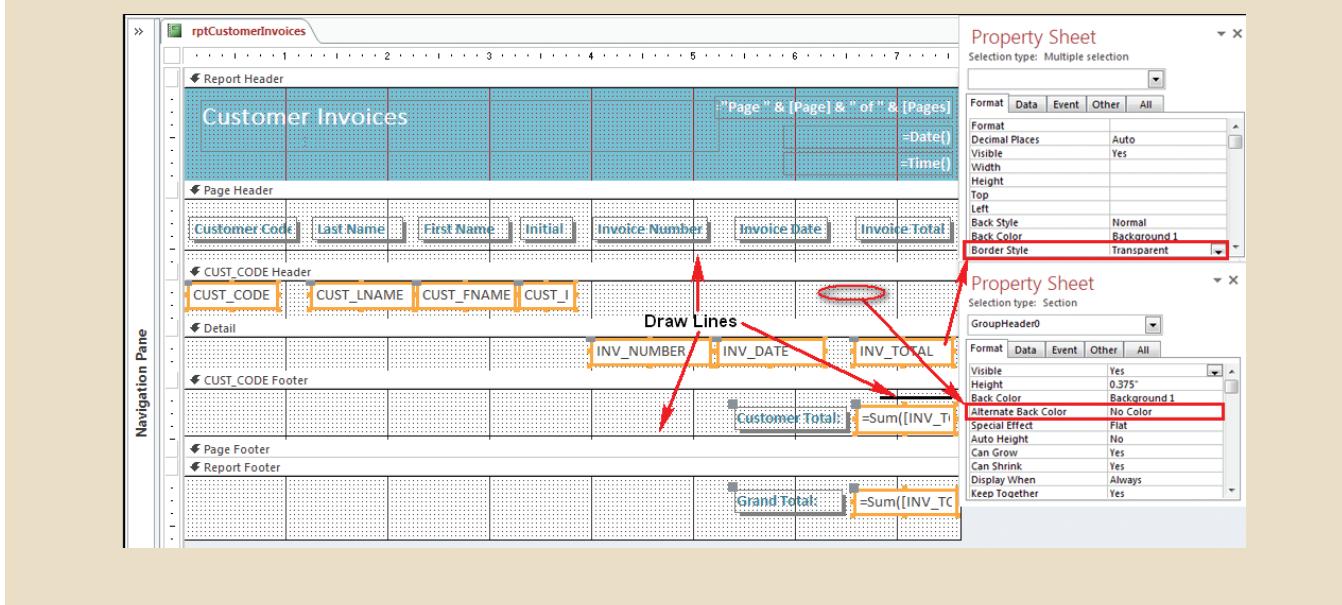
FIGURE M.112 SUBTOTALS AND TOTALS

Customer Code	Last Name	First Name	Initial	Invoice Number	Invoice Date	Invoice Total
10011			K	1002	16-Mar-18	\$10.78
				1004	17-Mar-18	\$37.66
				1008	17-Mar-18	\$431.08
					Customer Total:	\$479.52
10012			W	1003	16-Mar-18	\$166.16
					Customer Total:	\$166.16
10014				1001	16-Mar-18	\$26.94
				1006	17-Mar-18	\$429.66
					Customer Total:	\$456.59
10015				1007	17-Mar-18	\$37.77
					Customer Total:	\$37.77
10018			G	1005	17-Mar-18	\$76.08
					Customer Total:	\$76.08
					Grand Total:	\$1,216.11

As you examine Figure M.112, note the creation of the (computed) customer total text box and especially note its computation through the SUM function. The **same function is used to compute the grand total**. Aside from the fact that the two new text boxes have different labels, the important difference between the two text boxes (customer total and grand total) is their **location**. The SUM function in the text box located in the Report Footer area will add up all the invoice totals for all customers. The SUM function in the text box located at the CUST_CODE Footer area will add up all the invoice totals but only for each individual group of customers. That is, it will add up all the invoices for customer 10011, then 10012, etc.

Next, use Figure M.113 as a guide to reduce spacing in the Page Footer area to change the labels and formatting. Note that the customer code header, detail, and customer code footer have different colored backgrounds. You can select these portions in Design View and change the **Alternate Back Color** property to **No Color**. Note that the text boxes also have a border around them. Select the text boxes and change the **Border Style** to **Transparent**. Rename the fields and change the font colors to match Figure M.113.

FIGURE M.113 ADDITIONAL EDITS



The line tool on the **DESIGN** tab can be used to create lines to visually separate certain parts of the report as shown in Figure M.113. Change the thickness of the line in the CUST_CODE footer by using the following path: **FORMAT** tab on the Ribbon and then **Shape Outline, Line Thickness, 2 pt**. Figure M.114 shows the report output in Report View.

FIGURE M.114 COMPLETED REPORT OUTPUT

rptCustomerInvoices

Customer Invoices

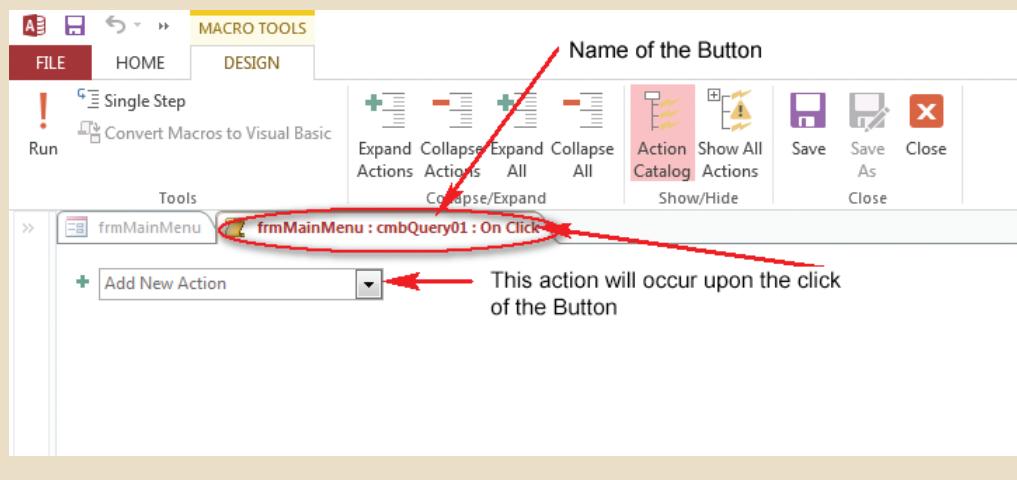
Page 1 of 1
Friday, October 13, 2017
11:33:07 AM

Customer Code	Last Name	First Name	Initial	Invoice Number	Invoice Date	Invoice Total
10011	Dunne	Leona	K			
				1002	16-Mar-18	\$10.78
				1004	17-Mar-18	\$37.66
				1008	17-Mar-18	\$431.08
				Customer Total:		\$479.52
10012	Smith	Kathy	W			
				1003	16-Mar-18	\$166.16
				Customer Total:		\$166.16
10014	Orlando	Myron				
				1001	16-Mar-18	\$26.94
				1006	17-Mar-18	\$429.66
				Customer Total:		\$456.59
10015	O'Brien	Amy	B			
				1007	17-Mar-18	\$37.77
				Customer Total:		\$37.77
10018	Farris	Anne	G			
				1005	17-Mar-18	\$76.08
				Customer Total:		\$76.08
				Grand Total:		\$1,216.11

M-9 Macros

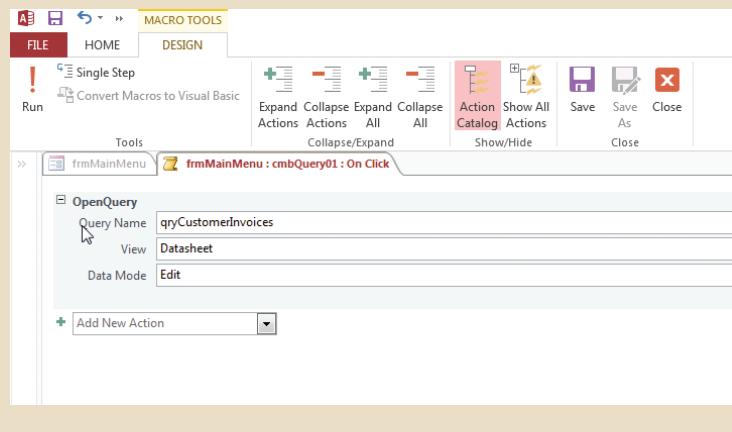
Macros are code sets that let you perform a wide variety of actions that range from opening and closing forms, queries, and reports to calculating and inserting values on a form. Open **frmMainMenu** in Design View. Right-click the **qryCustomerInvoices** button, select **Build Event**, and then select **Macro Builder** and click **OK**. This will take you to the screen shown in Figure M.115.

FIGURE M.115 DESIGN VIEW OF A MACRO



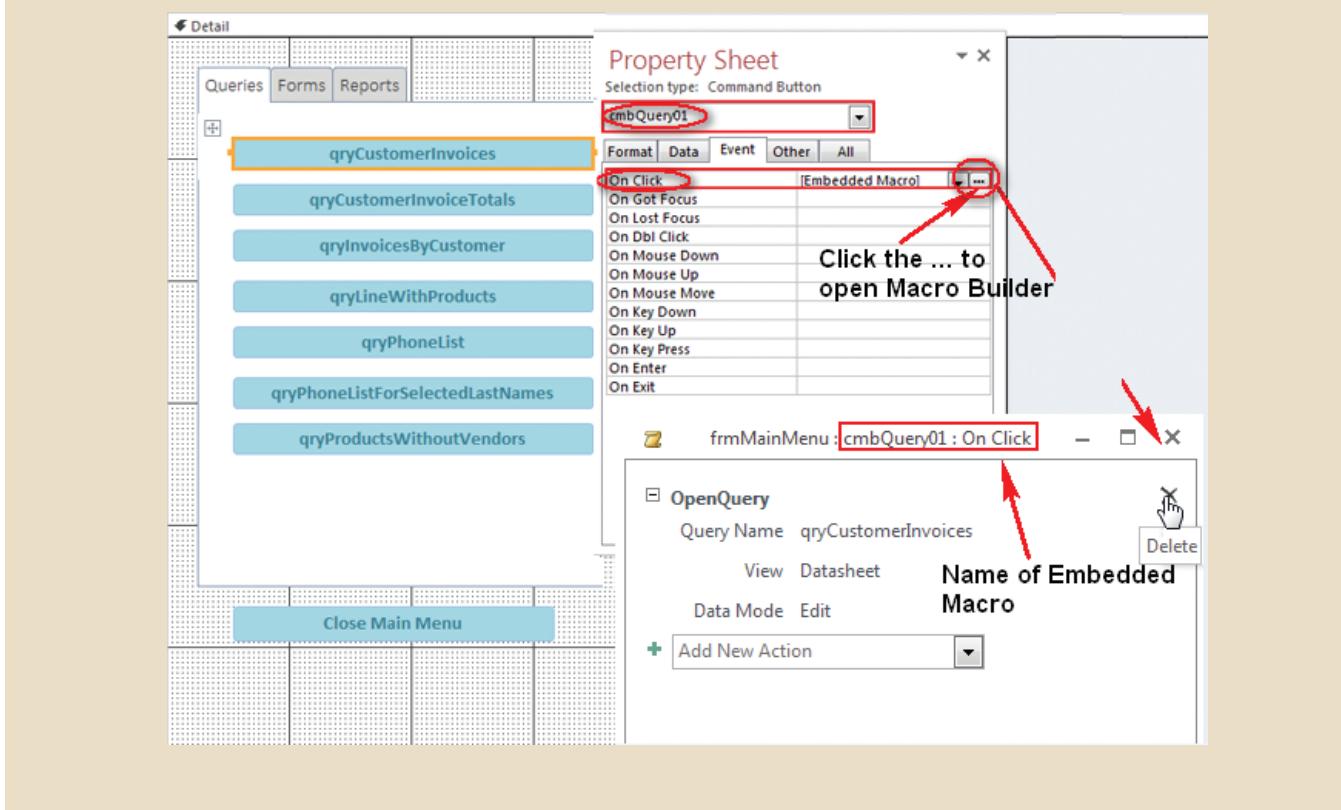
Select the **Add New Action** drop-down and select **OpenQuery**. Type in the query name that you wish for the button to open, or select it from the drop-down list. For this button it is **qryCustomerInvoices**. Leave the defaults that are set for View (Datasheet) and Data Mode (Edit) as they are. This is shown in Figure M.116. Exit the macro and click **Yes** to save changes. You do not have to give the macro a name because it is an embedded macro that is associated with the cmdQuery01 button.

FIGURE M.116 ADDING AN ACTION TO THE MACRO



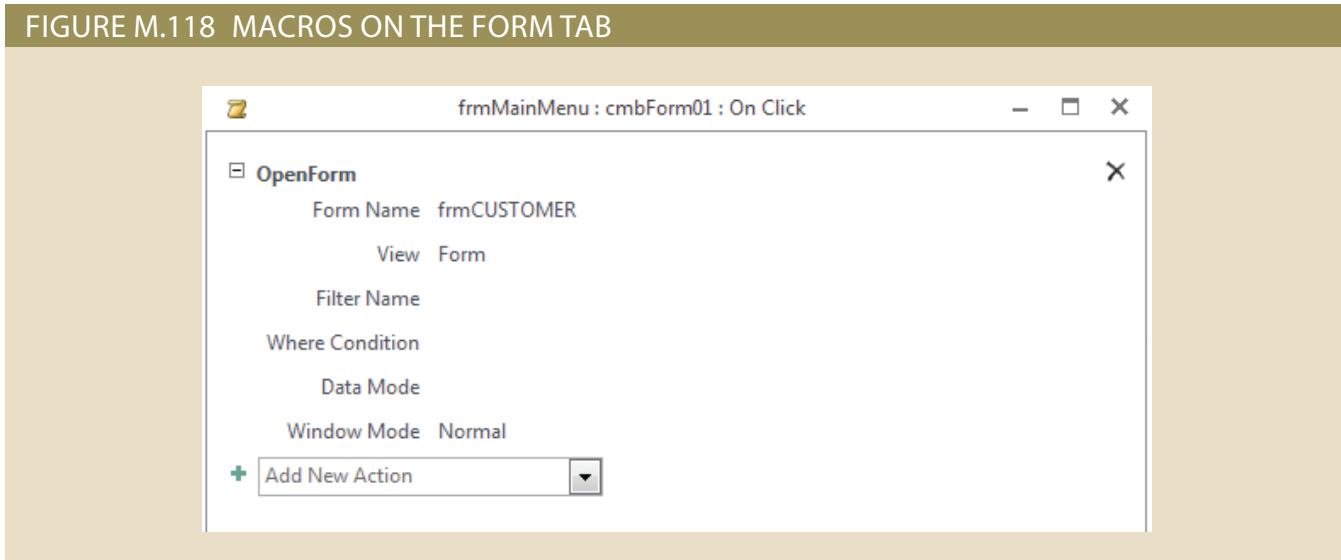
If you wish to edit the macro, open the Property Sheet for the selected button, click the **Event** tab, and then click the ellipses (...) button for the **On Click** property. This will take you to the Macro Builder. To delete an action from the Macro Builder, hover your mouse over the far right corner of Macro Builder screen near the action you want to delete and select the **X** that shows up in the corner. This will remove the current action. Figure M.117 demonstrates how to delete actions from a macro. You can practice deleting the macro but make sure not to save upon closing (we don't want to actually delete the macro).

FIGURE M.117 EDITING THE MACRO



Switch to the **Forms** tab so that you can build the macros for each of the buttons on the form. Select **Add New Action** then **OpenForm**. Select the appropriate form name for each button and leave all the other default settings as they are. This is shown in Figure M.118.

FIGURE M.118 MACROS ON THE FORM TAB



Create the rest of the macros for all of the buttons on the Form tab. Go ahead and create one last button on the Report tab with an associated macro to open the report we created in Section M-8.

Return to the Main Menu form in Form View and test all of your buttons.

M-9 Conclusion

Only a few examples are shown in this tutorial. The objective is not to develop full-blown applications, but to show you some examples of what can be done in the Microsoft Access environment. Once you have seen those examples, you have a foundation on which to build greater expertise. Keep in mind that Access is a superb prototyping tool, but it is not capable of serving the full database and information needs of even medium-sized organizations, let alone large ones. Products such as Microsoft's SQL Server, IBM's DB2, or Oracle are better candidates for such environments. Nevertheless, given its ability to let you develop superb prototypes, Access has earned a place of honor in the ranks of database professionals.

Appendix N

Creating a New Database Using Oracle 12c

Preview

Although the general database creation format tends to be generic, its execution tends to be DBMS-specific. The leading RDBMS vendors offer the DBA the option to create databases manually, using SQL commands or using a GUI-based process. Which option is selected depends on the DBA's sense of control and style.

Using the Oracle Database Configuration Assistant, it is simple to create a database. The DBA uses a wizard interface to answer a series of questions to establish the parameters for the database to be created. Other than the name of the database and passwords for administrator accounts, in most cases, the wizard will suggest common options and default values that produce a well-configured database. Figures N.1 through N.17 show you how to create a database with the help of the Oracle Database Configuration Assistant.

Data Files and Available Formats

MS Access **Oracle** **MS SQL** **My SQL**

MS Access **Oracle** **MS SQL** **My SQL**

There are no data files for this appendix.

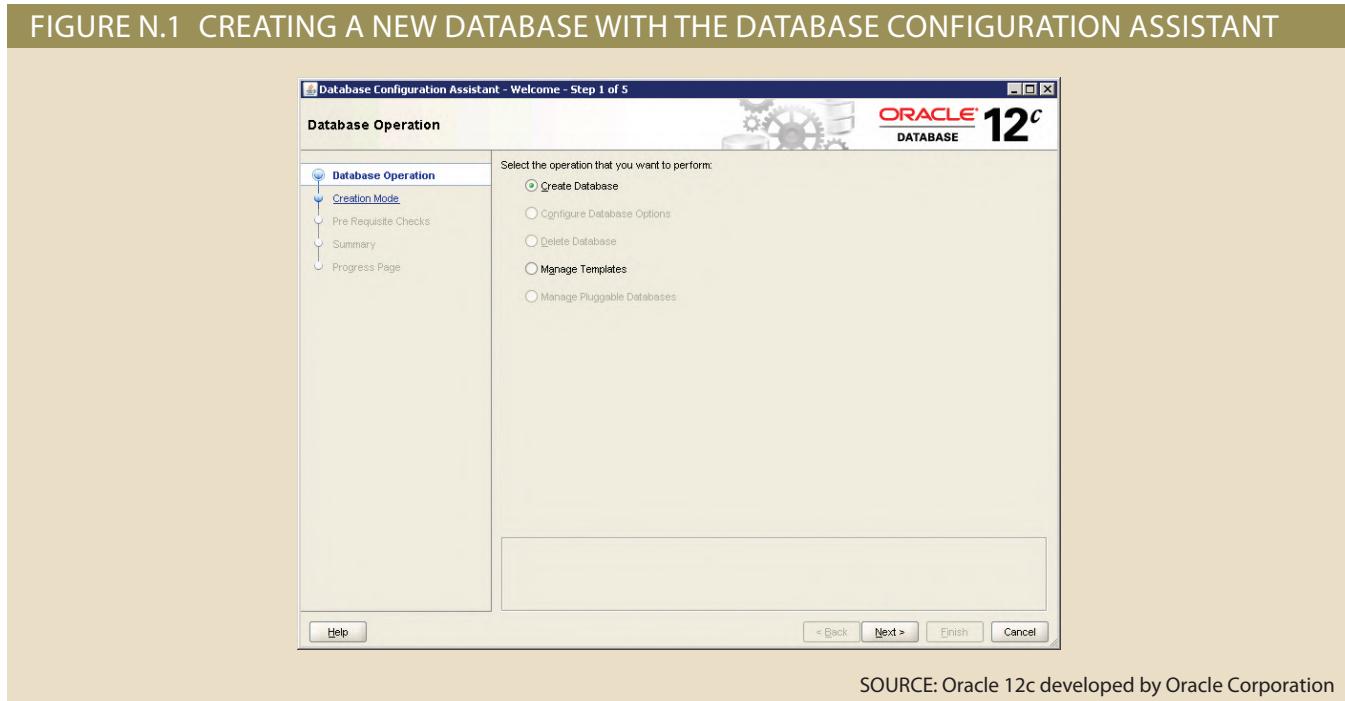
Data Files Available on cengagebrain.com

N-1 Creating a Standalone Database

Creating a single instance database in Oracle 12c is the equivalent of creating a database in Oracle 11g. It creates a standalone database that uses its own set of memory structures and background processes. Oracle has simplified the creation of a standalone database by improving default value selections so that the database can be created by specifying just the global database name and providing appropriate passwords.

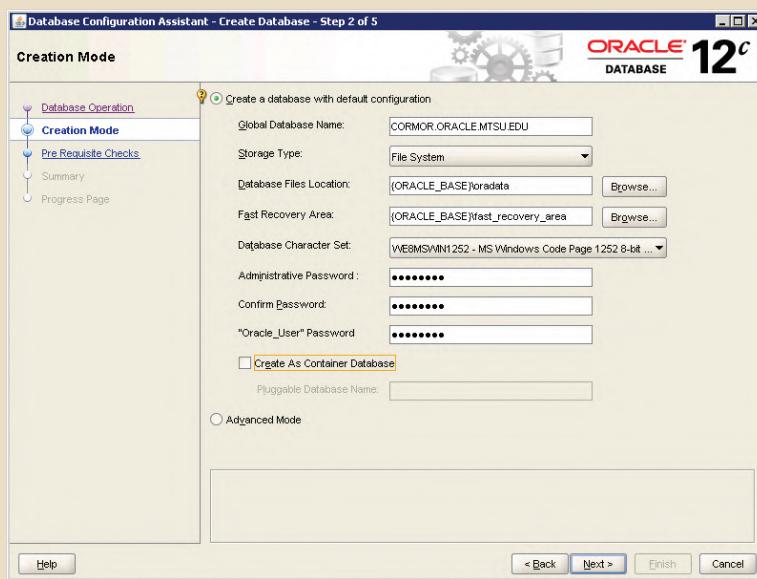
After confirming the database options selected in Figures N.1 through N.3, the database creation process starts as shown in Figure N.4. This process creates the database structure, including the necessary data dictionary tables, the administrator user accounts, and other supporting processes required by the DBMS to manage the database. Figure N.5 illustrates the completion of the database creation process. Depending on the capabilities of the computing platform on which the database is being installed, the configuration options chosen, and whether or not sample schemas are to be loaded into the database, the creation process can be lengthy.

FIGURE N.1 CREATING A NEW DATABASE WITH THE DATABASE CONFIGURATION ASSISTANT



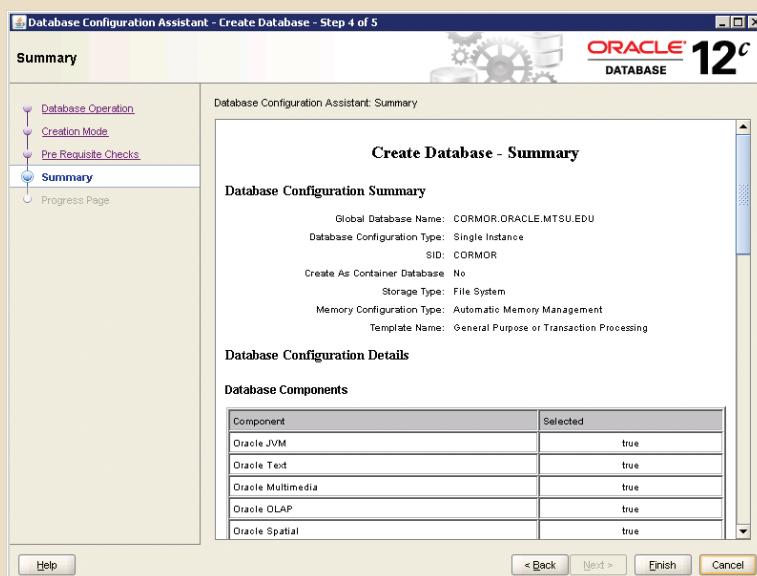
SOURCE: Oracle 12c developed by Oracle Corporation

FIGURE N.2 NAMING THE DATABASE AND SETTING PASSWORDS



SOURCE: Oracle 12c developed by Oracle Corporation

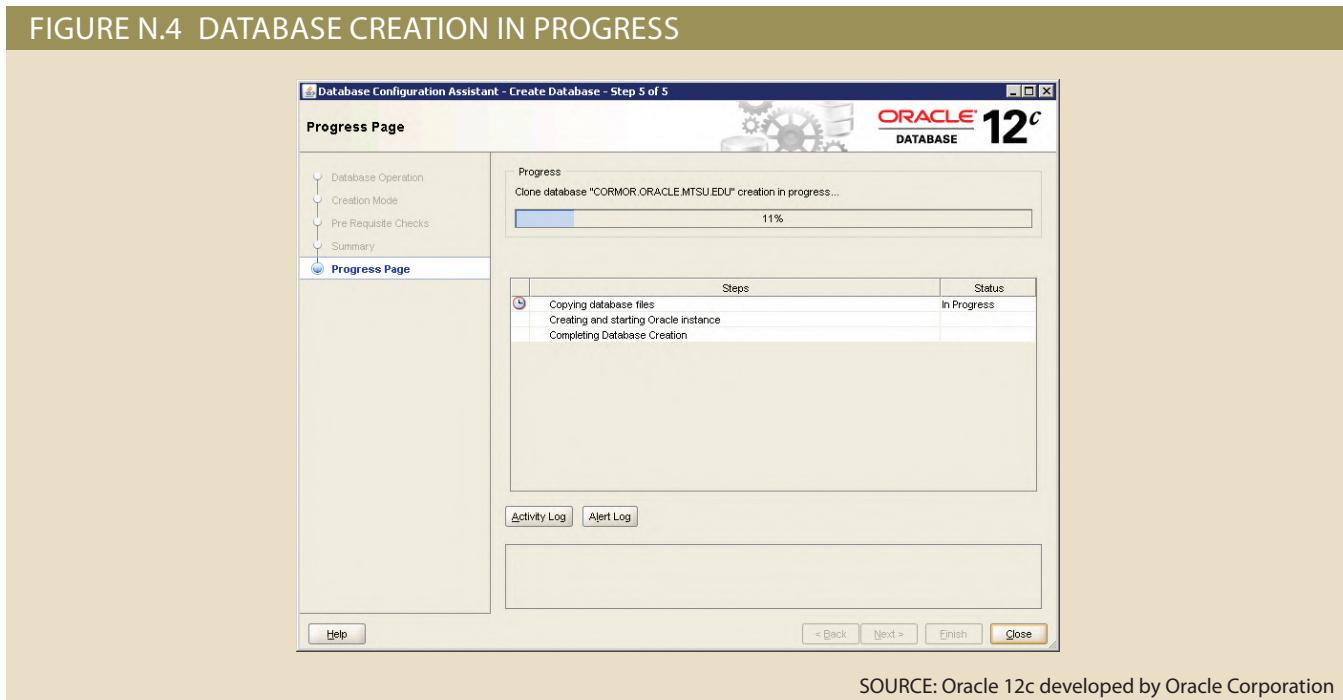
FIGURE N.3 REVIEW THE CREATION SUMMARY



SOURCE: Oracle 12c developed by Oracle Corporation

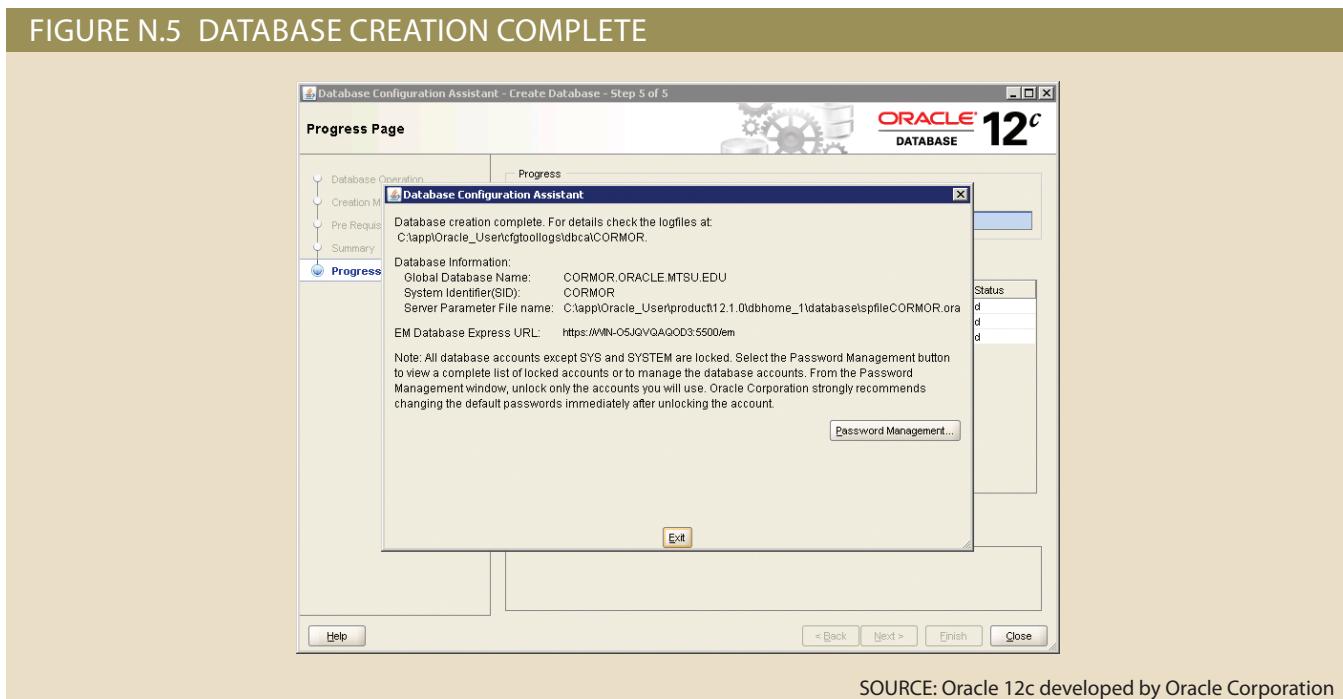
N-4 Appendix N

FIGURE N.4 DATABASE CREATION IN PROGRESS



SOURCE: Oracle 12c developed by Oracle Corporation

FIGURE N.5 DATABASE CREATION COMPLETE



SOURCE: Oracle 12c developed by Oracle Corporation

N-2 Creating a Multitenant Database

Oracle 12c introduced the concept of multitenant databases. Simplified, a multitenant database is a database that contains other databases. In a **multitenant database** environment, a container database is created that contains one or more pluggable databases. A **pluggable database** is simply a database that is contained in a database container. It looks and acts just like a standalone database to applications and users. These databases are called pluggable because they can be unplugged from one container database and easily plugged into another container database with relative ease. This makes it easy for a DBA to move a database from one server to another. Imagine, for example, a database used to support income tax preparation for an accounting firm. During most of the year, the database may get little use. As the income tax due date approaches, the database may become extremely busy. As a pluggable database, the DBA could keep the database plugged into a container on a low-cost, less powerful server most of the year, and then move it to a powerful server during its peak usage times. All of these changes in location would be transparent to the users and applications, and would require little effort for the DBA.

Another advantage of multitenant databases is that they consolidate database infrastructure. All of the pluggable databases in a container share the same memory structures and background processes. If a company has 50 standalone databases running on a single server, then each database must be managed independently in terms of creating backups, applying patches, monitoring performance, and so on. Further, each standalone database would be running its own set of processes to manage the database. With the multitenant database, all pluggable databases in the container can be managed simultaneously through the container database. Patches, backups, and performance statistics are all performed and aggregated through the container. Applying a patch to the container database applies the patch to all of the pluggable databases in that container. Thus, the DBA's task is greatly simplified and server resources are more efficiently utilized.

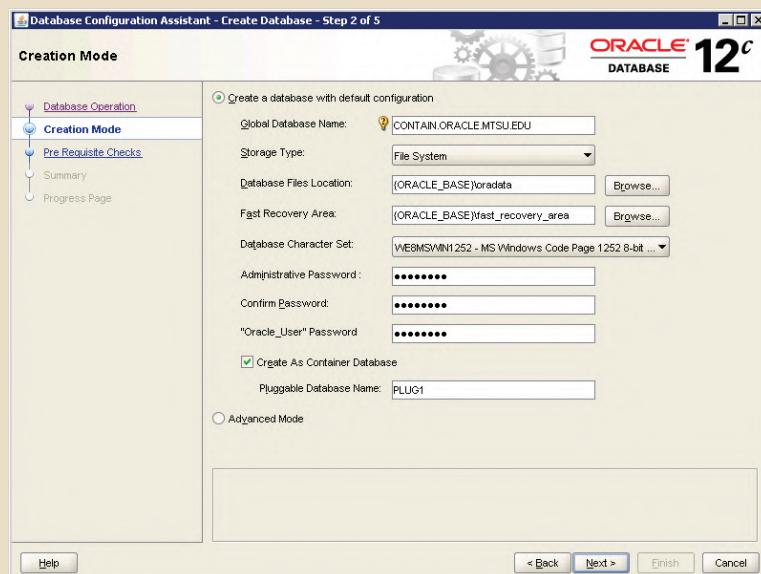
multitenant database

A database environment in which a container database can hold other databases.

pluggable database

In a multitenant database environment, a database that can be contained within a container database.

FIGURE N.6 CREATING A CONTAINER DATABASE FOR MULTITENANT DATABASES



SOURCE: Oracle 12c developed by Oracle Corporation

Notice that in Figure N.6 the container is created with one pluggable database, PLUG1, already installed.

N-6 Appendix N

Figures N.7 and N.8 show the creation of the container database and the first pluggable database. Figure N.9 shows the successful creation of both databases—the container database, CONTAIN, and the pluggable database, PLUG1.

FIGURE N.7 CONTAINER SUMMARY

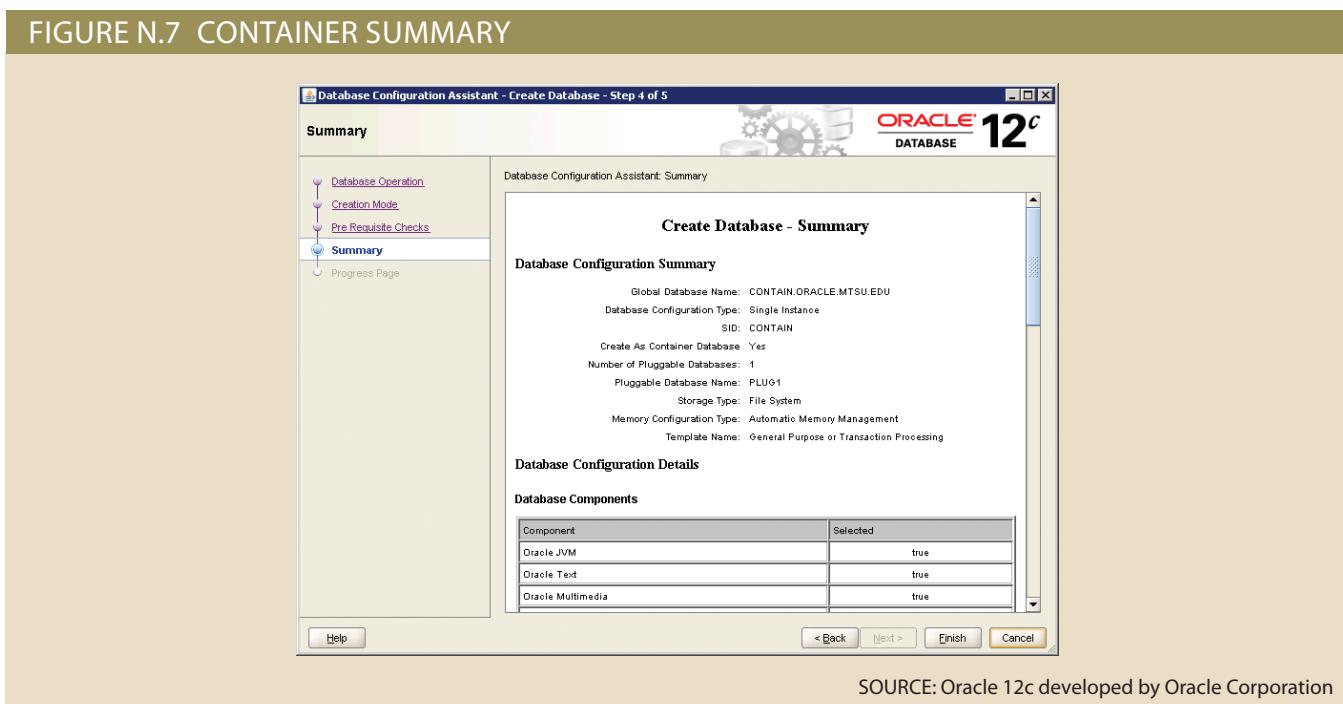


FIGURE N.8 CONTAINER AND PLUGGABLE DATABASE CREATION PROGRESS

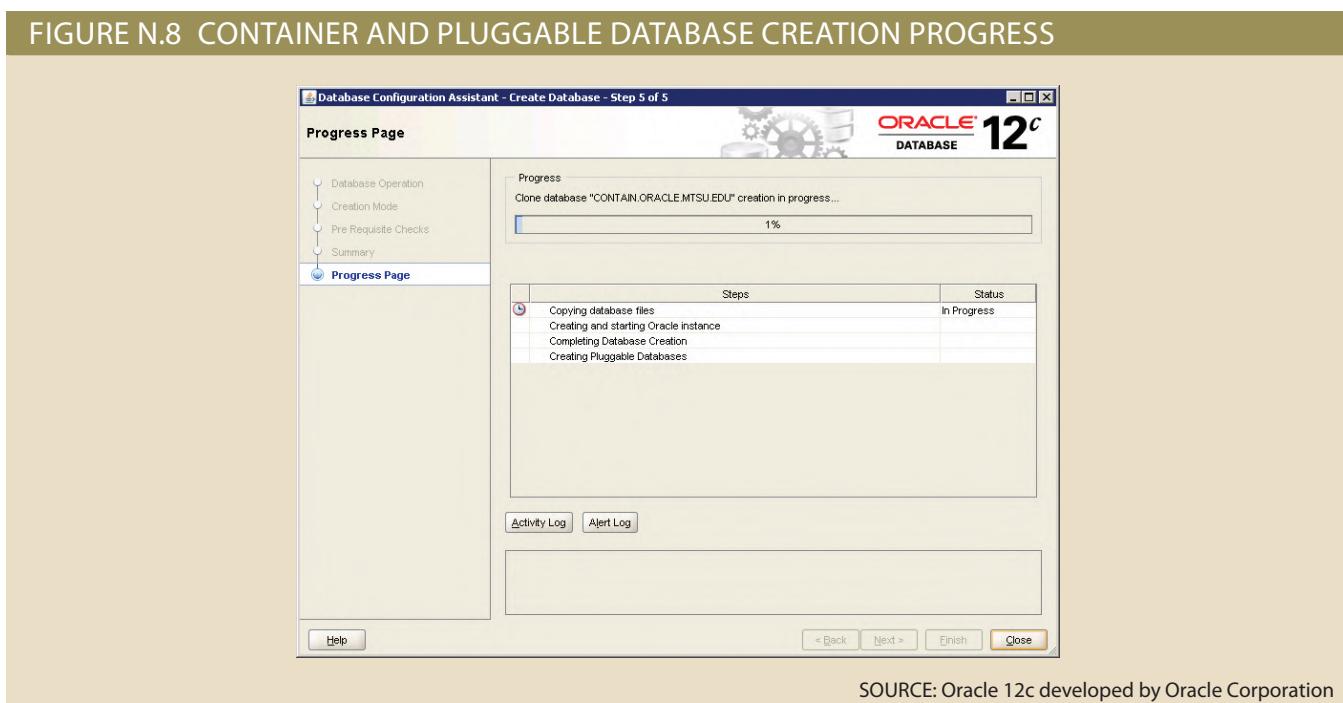
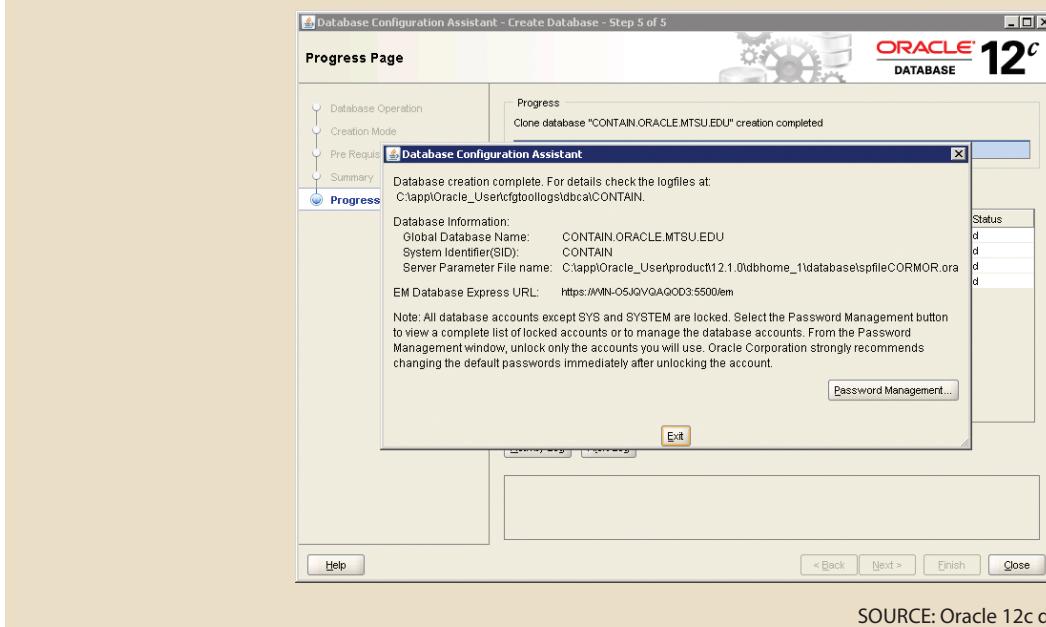


FIGURE N.9 CONTAINER AND PLUGGABLE DATABASES CREATION COMPLETE



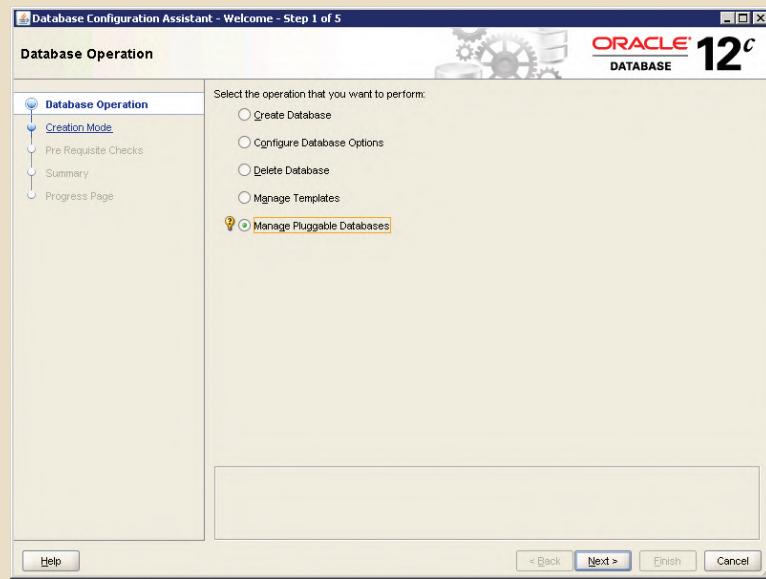
SOURCE: Oracle 12c developed by Oracle Corporation

N-3 Managing Pluggable Databases

Pluggable databases can be managed through the Database Configuration Assistant (DBCA). Using the DBCA, new pluggable databases can be created or deleted, and existing pluggable databases can be unplugged from a container or plugged into a container. Figures N.10 through N.17 illustrate the creation of a new pluggable database, PLUG2, in the CONTAIN container database created previously.

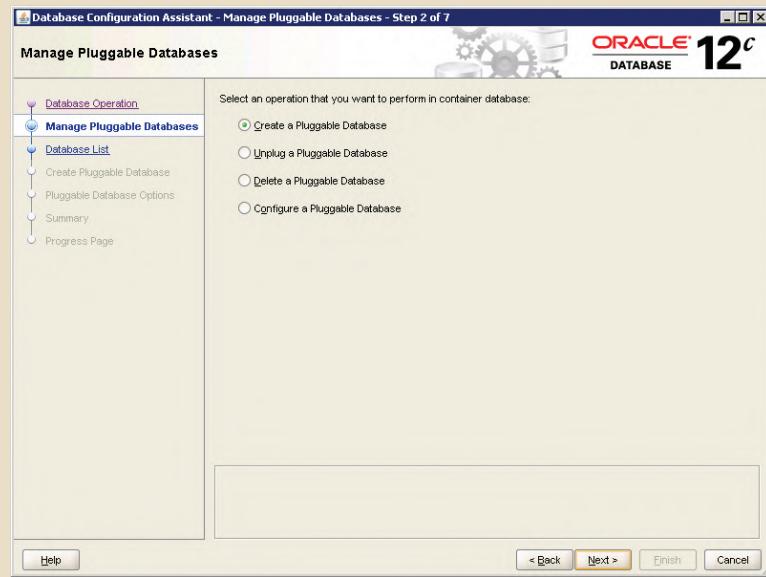
N-8 Appendix N

FIGURE N.10 MANAGING PLUGGABLE DATABASES



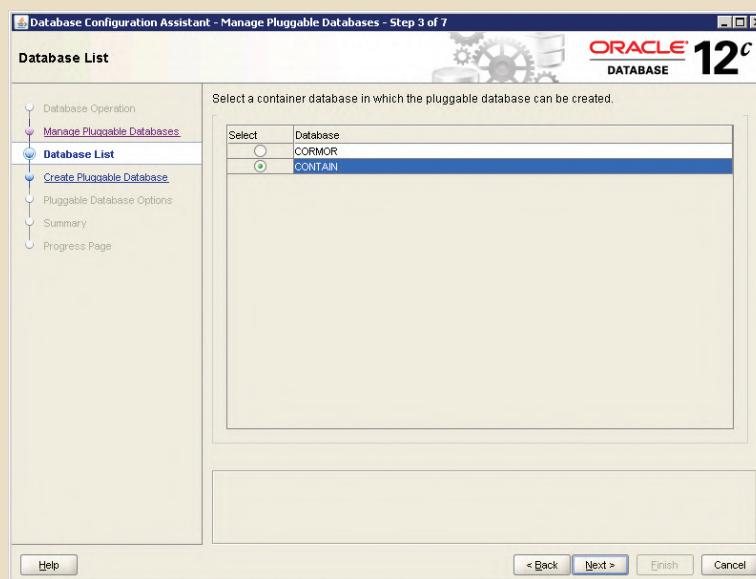
SOURCE: Oracle 12c developed by Oracle Corporation

FIGURE N.11 CREATING A NEW PLUGGABLE DATABASE



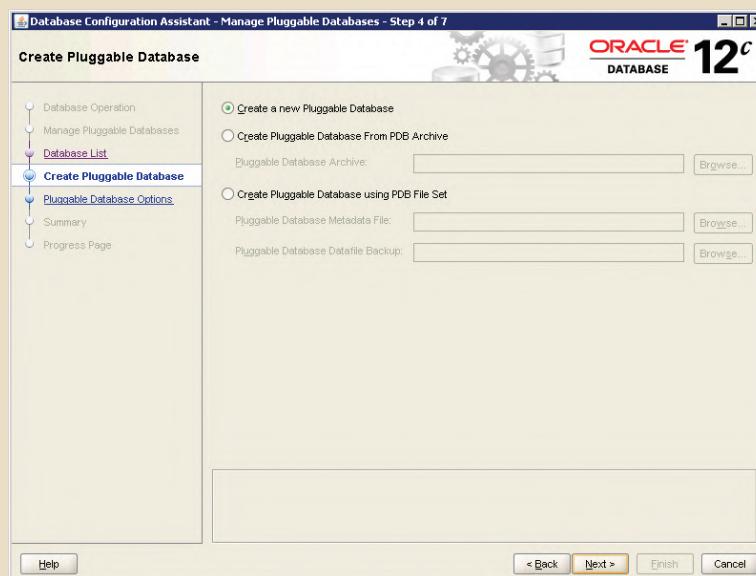
SOURCE: Oracle 12c developed by Oracle Corporation

FIGURE N.12 SPECIFYING A CONTAINER DATABASE



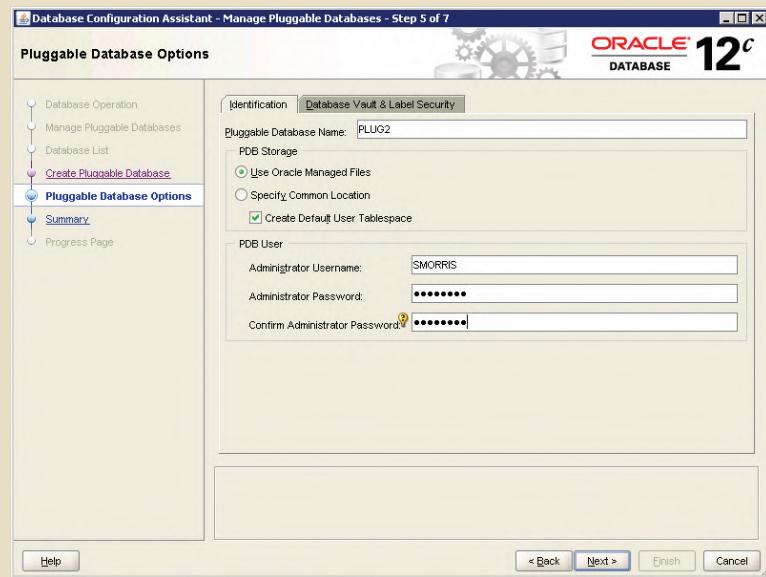
SOURCE: Oracle 12c developed by Oracle Corporation

FIGURE N.13 CHOOSING A CREATION SOURCE



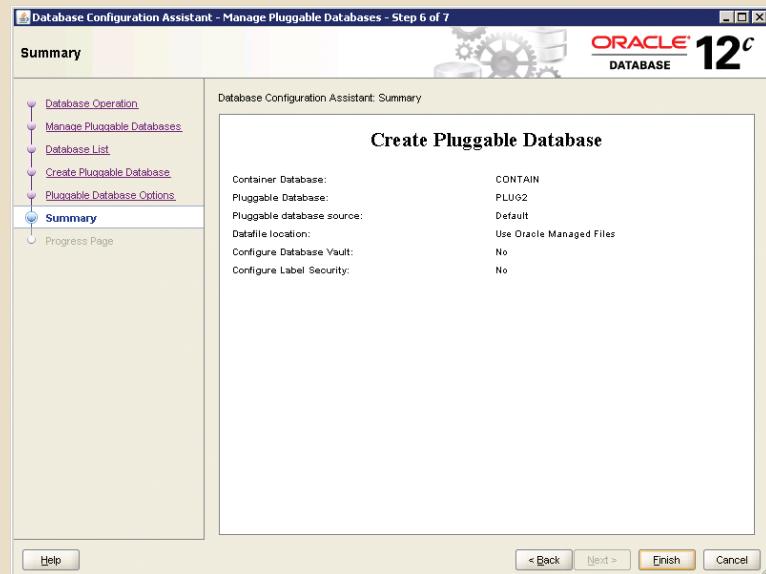
SOURCE: Oracle 12c developed by Oracle Corporation

FIGURE N.14 NAMING THE NEW DATABASE AND SPECIFYING AN ADMINISTRATOR



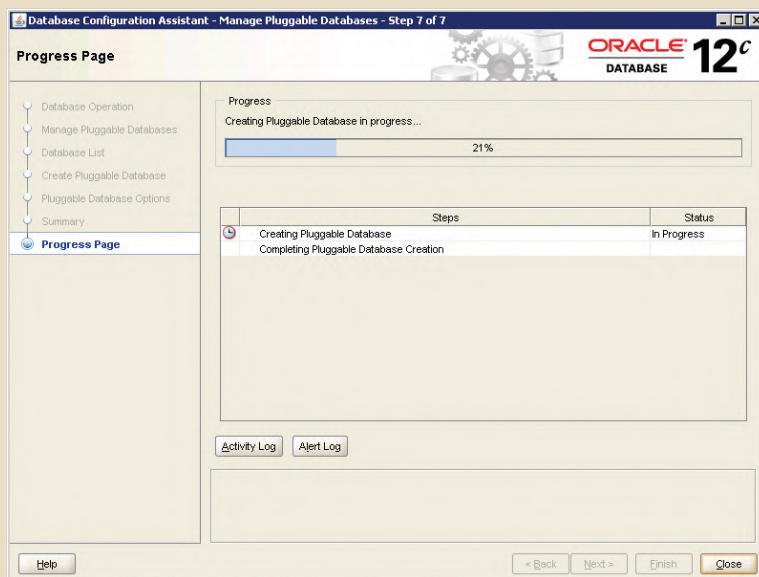
SOURCE: Oracle 12c developed by Oracle Corporation

FIGURE N.15 REVIEW THE PLUGGABLE DATABASE OPTIONS



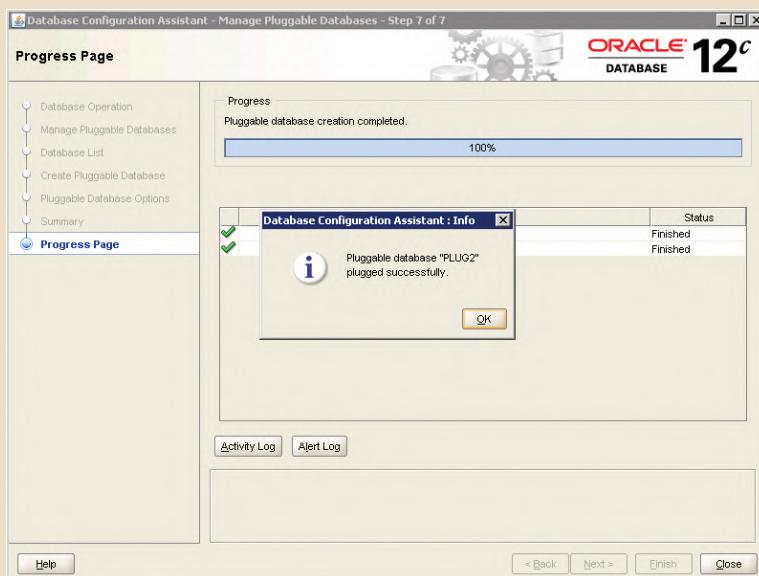
SOURCE: Oracle 12c developed by Oracle Corporation

FIGURE N.16 NEW PLUGGABLE DATABASE CREATION



SOURCE: Oracle 12c developed by Oracle Corporation

FIGURE N.17 NEW PLUGGABLE DATABASE COMPLETE



SOURCE: Oracle 12c developed by Oracle Corporation

Creation of the new pluggable database is much faster than the creation of a stand-alone or container database. The pluggable database uses the memory structures and background processes of the container database, so there are fewer components to create and few processes and services to start on the database server.

Key Terms

multitenant database, N-5

pluggable database, N-5

Appendix O

Data Warehouse Implementation Factors

Preview

Organization-wide information system development is subject to many constraints. Some of the constraints are based on available funding. Others are a function of management's view of the role played by an IS department and of the extent and depth of the information requirements. Add the constraints imposed by corporate culture, and you understand why no single formula can describe perfect data warehouse development. Rather than proposing a single data warehouse design and implementation methodology here, this appendix identifies a few common factors to consider in implementing data warehousing.

Data Files and Available Formats

MS Access **Oracle** **MS SQL** **My SQL**

MS Access **Oracle** **MS SQL** **My SQL**

There are no data files for this appendix.

Data Files Available on cengagebrain.com

0-1 Remember the Data Warehouse Is an Active Decision Support Framework

Perhaps the first thing to remember is that a data warehouse is not a static database. Instead, it is a dynamic framework for decision support, that is almost by definition a work in progress. Because the data warehouse is the foundation of a modern BI environment, designing and implementing a data warehouse means that you are involved in designing and implementing a complete database system development infrastructure for company-wide decision support. Although it is easy to focus on the data warehouse database as the central BI data repository, you must remember that the decision support infrastructure includes hardware, software, people, and procedures, as well as data. The argument that the data warehouse is the only *critical* component of BI success is as misleading as the argument that a human being needs only a heart or a brain to function. The data warehouse is a critical component of a modern BI environment, but it is certainly not the only critical component. Therefore, its design and implementation must be examined in light of the entire infrastructure.

0-2 Be Aware of Organizational Components and Solicit User Involvement

When you design a data warehouse, you are given an opportunity to help develop an integrated model of the data that are considered to be essential to the organization, from both end-user and business perspectives. Data warehouse data cross departmental lines and geographical boundaries. Because the data warehouse represents an attempt to model all of the organization's data, you are likely to discover that organizational components (divisions, departments, support groups, and so on) often have conflicting goals, and it certainly will be easy to find data inconsistencies and damaging redundancies. Also, because information is power, the control of data sources and uses is likely to trigger turf battles, end-user resistance, and power struggles at all levels. Building the perfect data warehouse is not just a matter of knowing how to create a star schema; it requires managerial skills to deal with conflict resolution, mediation, and arbitration. In short, the designer must:

- Involve end users in the process.
- Secure end users' commitment from the beginning.
- Solicit continuous end-user feedback.
- Manage end-user expectations.
- Establish procedures for conflict resolution.

0-3 Satisfy the Trilogy: Data, Analysis, and Users

Great managerial skills are not, of course, solely sufficient. The technical aspects of the data warehouse must be addressed as well. The old adage of input-process-output repeats itself here. The data warehouse designer must satisfy:

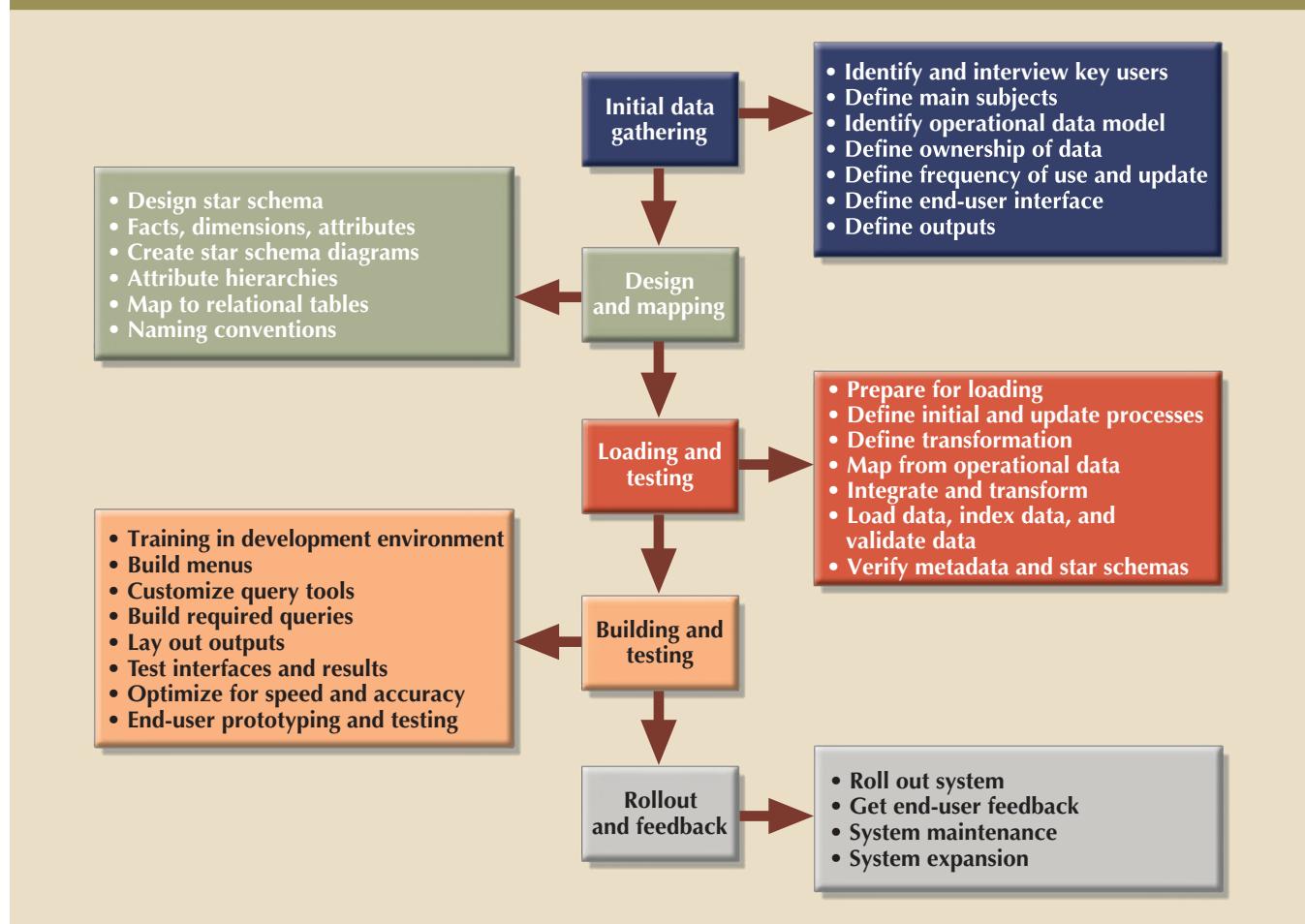
- Data integration and loading criteria.
- Data analysis capabilities with acceptable query performance.
- End-user data analysis needs.

The foremost technical concern in implementing a data warehouse is to provide end-user decision support with advanced data analysis capabilities—at the right moment, in the right format, with the right data, and at the right cost.

O-4 Apply Database Design Procedures

You learned about the database life cycle and the database design process in Chapter 9, Database Design, so perhaps it is wise to review the traditional database design procedures. These design procedures must then be adapted to fit data warehouse requirements. If you remember that the data warehouse derives its data from operational databases, you will understand why a solid foundation in operational database design is important. (It's difficult to produce good data warehouse data when the operational database data are corrupted.) Figure O.1 depicts a simplified process for implementing the data warehouse.

FIGURE O.1 DATA WAREHOUSE DESIGN AND IMPLEMENTATION ROAD MAP



As noted, developing a data warehouse is a company-wide effort that requires many resources: human, financial, and technical. Providing company-wide decision support requires a sound architecture based on a mix of people skills, technology, and managerial procedures that is often difficult to find and implement. For example:

- The sheer and often mind-boggling quantity of decision support data is likely to require the latest hardware and software—that is, advanced computers with multiple processors, advanced database systems, and large-capacity storage units. In the not-too-distant past, those requirements usually prompted the use of a mainframe-based system. Today's client/server technology offers many other choices to implement a data warehouse.
- Very detailed procedures are necessary to orchestrate the flow of data from the operational databases to the data warehouse. Data flow control includes data extraction, validation, and integration.
- To implement and support the data warehouse architecture, you also need people with advanced database design, software integration, and management skills.

Appendix P

Working with MongoDB

Preview

This appendix gives you some step-by-step illustrations of using MongoDB, a popular document database. Among the NoSQL databases currently available, MongoDB has been one of the most successful in penetrating the database market. Therefore, learning the basics of working with MongoDB can be quite useful for database professionals.

Data Files and Available Formats

File name	Format/Description
Ch14_FACT.json	JavaScript Object Notation file (also used in Chapter 14)

Data Files Available on cengagebrain.com



Note

MongoDB is a product of MongoDB, Inc. In this appendix, we use the Community Server v.3.4.6 edition, which is open source and available free of charge from MongoDB, Inc. New versions are released regularly. This version of MongoDB is available from the MongoDB website for Windows, MacOS, and Linux.

P-1 Introduction to MongoDB

The name, MongoDB, comes from the word *humongous* as its developers intended their new product to support extremely large data sets. It is designed for:

- High availability
- High scalability
- High performance

As a document database, MongoDB is schema-less and aggregate aware. Recall from Chapter 14, Big Data and NoSQL, that being schema-less means that all documents are not required to conform to the same structure, and the structure of documents does not have to be declared ahead of time. Aggregate aware means that the documents encapsulate all relevant data related to a central entity within the same document. Data is stored in documents, documents of a similar type are stored in collections, and related collections are stored in a database. Documents are formatted using BSON for storage, which is a lightweight, binary representation of JSON, but with support for added features like a greater range of data types. To the users, the documents appear as JSON files, which makes them easy to read and easy to manipulate in a variety of programming languages. Understanding the basic structure of a JSON document is essential for working with MongoDB.

P-2 JSON Documents

JavaScript Object Notation (JSON) is a data interchange format that represents data as a logical object. Objects are enclosed in curly brackets {} that contain key-value pairs. When discussing JSON documents, the key-value pairs are typically written in the format *key:value*, so we will follow that convention in the discussion of MongoDB. In different circumstances, you may see the terminology *key:value*, *tag:value*, *field:value*, *id:value*, or *property:value* used, but they all mean the same thing, so do not allow these changes in terminology to throw you off. This variation in terminology is a testament to the wide range of contexts and programming languages that have adopted the JSON format.

A single JSON object can contain many *key:value* pairs separated by commas. A simple JSON document to store data on a book might look like this:

```
{_id: 101, title: "Database Systems"}
```

This document contains two *key:value* pairs:

- *_id* is a key with 101 as the associated value
- *title* is a key with "Database Systems" as the associated value

The *value* component may have multiple values that would be appropriate for a given key. In the previous example, adding a *key:value* pair for authors could have the values

“Coronel” and “Morris.” When there are multiple values for a single key, an array is used. Arrays in JSON are placed inside square brackets []. For example, the above document could be expanded to:

```
{_id: 101, title: "Database Systems", author: ["Coronel", "Morris"]}
```

When JSON documents are intended to be read by humans, they are often displayed with each *key:value* pair on a separate line to improve readability, such as:

```
{
  _id: 101,
  title: "Database Systems",
  author: ["Coronel", "Morris"]
}
```

Objects can also have other objects embedded as a value. When a JSON document has an embedded object, the embedded object is often referred to as a **subdocument**. Consider another simple document with data about a publisher that is related to the book in the previous example.

```
{
  Name: "Cengage",
  Address: "500 Topbooks Avenue",
  City: "Boston",
  State: "MA"
}
```

The book document and the publisher document are related. Remember, document databases are aggregate aware, so documents are typically arranged around a central entity. If we are constructing documents that have a book as the central entity, then we may wish to include data about the publisher inside that document. To include the publisher information with the book, we could embed the publisher as a subdocument inside the book document, as follows:

```
{
  _id: 101,
  title: "Database Systems",
  author: ["Coronel", "Morris"],
  publisher: {
    name: "Cengage",
    street: "500 Topbooks Avenue",
    city: "Boston",
    state: "MA"
  }
}
```

In this case, we have avoided a situation that would have required a join in a relational environment. In a relational environment, we would have used a BOOK table and a PUBLISHER table with a 1:M relationship. In essence, we have pre-joined the

subdocument

A JSON document that is embedded as an object within another JSON document.

book and publisher data into a single document. Although this increases redundancy, NoSQL databases often sacrifice redundancy to improve scalability. Remember, with document databases, we are attempting to avoid the need for joins, making documents independent of each other so they can be easily scaled out to many computers in a cluster.

In addition to pre-joining data, subdocuments can be useful when a value comprises smaller meaningful components. In the previous example, the street, city, and state pairs all work together to provide an address. Therefore, an address object could be created that includes these properties:

```
{
  _id: 101,
  title: "Database Systems",
  author: ["Coronel", "Morris"],
  publisher: {
    name: "Cengage",
    address: {
      street: "500 Topbooks Avenue",
      city: "Boston",
      state: "MA"
    }
  }
}
```

This document is organized around a book. The book document has four *key:value* pairs. The third *key:value* pair (author) has an array of multiple values. The fourth pair (publisher) has an object as the value. The publisher object is composed of two *key:value* pairs, the second of which (address) is an object that is composed of three *key:value* pairs. The JSON format allows an indefinite number of objects to be embedded inside each other, and it is possible to even have arrays of objects. For example, the author array in our example could be expanded as:

```
{
  _id: 101,
  title: "Database Systems",
  author: [
    {
      name: "Coronel",
      email: "ccoronel@mtsu.edu",
      phone: "6155551212"
    },
    {
      name: "Morris",
      email: "smorris@mtsu.edu",
      office: "301 Codd Hall"
    }
  ]
}
```

```

    }
],
publisher: {
    name: "Cengage",
    address: {
        street: "500 Topbooks Avenue",
        city: "Boston",
        state: "MA"
    }
}
}

```

JSON format is very flexible and very powerful. It can represent the data in many different ways so that programmers can find the representation that best suits the needs of each individual application. JSON is well-suited to schema-less data models because there is no requirement for each document in a collection to have the same *key:value* pairs. Notice, for example, that there are differences in the *key:value* pairs for the two author objects within the author array in our illustration.

As you work to get comfortable with document databases, drawing connections to the relational topics that you are already familiar with can be helpful. A collection in MongoDB can be thought of as being similar to a table in a relational database in that they both contain data about a given topic. Obviously, the data in a collection is aggregate aware so it is a very different set of data than would appear in the relational table. A document is like a row of data in a relational table. The *key:value* pairs can be thought of as being like a column and a value in the column. Because each document (row) can have different *key:value* pairs, it is like allowing each row in a table to have different columns (thus, document databases are schema-less). An array in a *key:value* pair is the document database way of handling multivalued attributes. Embedded objects in a document can correspond to composite attributes or to pre-joined tables. The use of arrays and embedded objects allow the document database to be aggregate aware and avoid using multiple tables that would require joins to combine for reporting purposes.

P-3 Creating Databases and Collections in MongoDB

Just as relational databases can be accessed through a wide range of application programming languages such as JavaScript, Java, Python, and PHP, so can MongoDB. MongoDB also provides a built-in MongoDB shell program that provides an interface for working directly with the data. Using the shell, data is queried using Mongo Query Language (MQL). MQL is based on JavaScript, and many of the programming features of JavaScript, such as IF statements, variable declaration and manipulation, and looping, are available directly in MongoDB. Although full JavaScript programming is beyond the scope of this book, it is important to consider basic data manipulation in MQL.

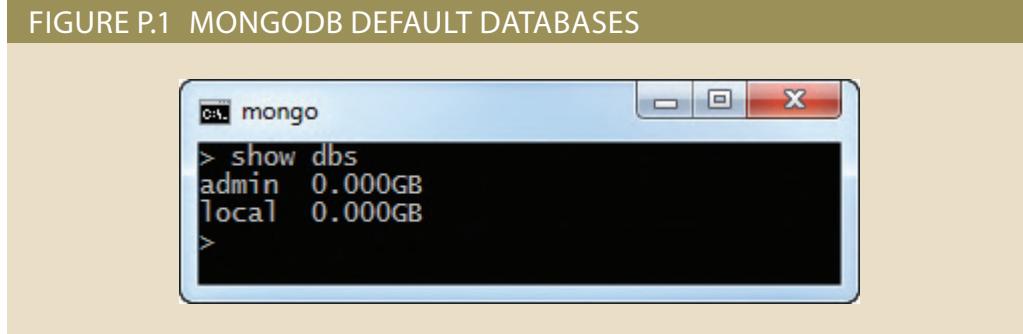
MongoDB databases comprise collections of documents. Each MongoDB server can host many databases. Within the MongoDB server (also called the host), each database is a type of object. As we know from the previous discussion of JSON format, objects can contain other objects. A database object contains collections. Collections are also

objects. Collection objects contain document objects. In addition to holding data content, an object can also have **methods**, which are programmed functions for manipulating the object. When connected to the MongoDB server, the first task is to specify with which database object you want to work. A list of the databases available on the server can be retrieved with the command:

```
show dbs
```

By default, a new installation of MongoDB includes an admin database and a local database as shown in Figure P.1. The admin database is used to record data about database administration issues such as users, roles, and privileges for the databases hosted on this server. The local database is used for storing data about the server's start-up process and the server's role in sharding operations. Recall that aggregate aware databases try to break up the data into pieces called shards. As a database grows in volume, MongoDB may adjust the way shards are distributed among the cluster nodes to balance the workload. Unless you are administering the MongoDB server, you are not likely to need to worry about the admin and local databases, as they should never be used for storing end user data.

FIGURE P.1 MONGODB DEFAULT DATABASES



All data manipulation commands in MongoDB must be directed to a particular database. Creating a new database in MongoDB is as easy as issuing the *use* command. The *use* command informs the server which database is to be the target of the commands that follow. If there is a database with the name specified, then that database will be used for the subsequent commands. If there is not a database with that name, then one is created automatically. For example, the following command creates a database named *demo* that we will use for some of the later code examples:

```
use demo
```

This command creates a database named “*demo*” and sets a special MongoDB variable named *db* to have the value “*demo*”. In subsequent commands, we will use the variable *db* to specify the database that is targeted by our commands. The name of the database that is being used can be displayed using the **getName()** method on the *db* variable, as follows:

```
db.getName()
```

method

A programmed function within an object used to manipulate the data in that same object.

getName()

A MongoDB method for returning the name of the current database.



Note

Notice that the *show dbs* command will not include the new database in the list of available databases until the database contains at least one collection.

Although there are conceptual similarities between collections and relational tables, the differences become much more obvious when creating a collection. Creating a table in a relational database requires specifying a great deal of metadata. The name and data type for each attribute, any column constraints, primary key constraints, foreign key constraints, and check constraints are all specified at the time of table creation. Because document databases are schema-less with no predefined structure, none of these parameters are required. Creating a collection requires only specifying the database it belongs to and the name of the collection. Using the `createCollection()` method with the `db` variable creates a collection with the specified name. The following command creates a “newproducts” collection inside the previously defined demo database:

```
db.createCollection("newproducts")
```

To display the collections that exist in a database, the following command can be used, as shown in Figure P.2:

```
show collections
```

FIGURE P.2 MONGODB COLLECTIONS

The screenshot shows a Windows-style application window titled "mongo". Inside the window, the mongo shell is running. The user has entered the following commands:

```
> use demo
switched to db demo
> db.createCollection("newproducts")
{ "ok" : 1 }
> show collections
newproducts
>
```

Now that the demo database has at least one collection, `demo` shows as one of the available databases returned by the `show dbs` command. Although it was not necessary to specify parameters when creating the collection, there are a few options that are available. For instance, there is an `autoIndexID` option that acts as a surrogate key. When `autoIndexID` is set to true for a collection, the DBMS automatically adds a “`_id`” field to each document to act as a unique identifier and automatically create an index on that field to speed retrievals. `AutoIndexID` is set to true by default.

Using collection creation options, it is also possible to create a **capped collection** in MongoDB. A capped collection is a MongoDB collection that can only grow to a specified maximum size; then documents are automatically deleted from it. When a capped collection reaches its maximum size and new documents are added, the oldest documents are automatically deleted to make room for the new documents. Capped collections are often used for log files where only a limited number of recent actions are of interest. Capped collections can be limited by total storage size and/or by number of documents. For example, the following command creates a collection named `userlog` that is capped to 1,000 documents and 1 megabyte in size:

```
db.createCollection("userlog", {capped: true, size: 1048576, max: 1000})
```

createCollection()

A MongoDB method for creating a collection inside a database.

capped collection

A MongoDB collection that has a specified limit on the size and/or number of documents within the collection. Older documents are automatically deleted to make room for new documents once the limit is reached.

As you might be able to see, most MQL commands borrow their syntax from JavaScript and the JSON format.

P-4 Renaming and Dropping Collections

Occasionally, it might be necessary to rename collections within MongoDB. To rename a collection you use the `renameCollection()` method of the collection, and provide the new name for the collection as a parameter. For example, the following command renames the `newproducts` collection to `products`:

```
db.newproducts.renameCollection("products")
```

Using the `show collections` command, you can see that the collection name has changed (see Figure P.3).

FIGURE P.3 RENAMING A MONGODB COLLECTION



Dropping a collection removes all documents in the collection and deletes any indexes that have been created with that collection. Dropping the collection is done with the `drop()` method of the collection. The `drop()` method does not require any options or parameters. It drops whichever collection that it was called with. For example, the following command drops the `userlog` collection that was created earlier:

```
db.userlog.drop()
```

P-5 Inserting Documents in MongoDB

Once a collection has been created, documents can be added to it. The `insert()` method of the collection is used to add new documents to the collection. The syntax for the `insert()` method in MongoDB is as follows:

```
db.<collection name>.insert({document})
```

Consider the following JSON document:

```
{name: "standard desk chair",
  price: 150,
  brand: "CheapCo",
  type: "chair"}
```

renameCollection()
A MongoDB method to change the name of an existing collection within a database.

drop()
A MongoDB method to drop a collection and all of its documents from a database.

insert()
A MongoDB method to add new documents to an existing collection.

In order to insert the previous document into the products collection that we created earlier, we call the `insert()` method of the collection and provide the document as a parameter, as follows:

```
db.products.insert ({name: "standard desk chair",
    price: 150,
    brand: "CheapCo",
    type: "chair"})
```

Retrieving documents in MongoDB will be explored in greater detail later in this chapter; however, for now we can retrieve the documents in our collection using the `find()` method. Adding the `pretty()` method, discussed later, improves the readability of the returned document. The following command displays all of the documents in the product collection, as shown in Figure P.4:

```
db.products.find()
```

FIGURE P.4 RETRIEVING A MONGODB DOCUMENT

The screenshot shows a Windows-style application window titled 'mongo'. Inside, the mongo shell is running a command: `> db.products.find().pretty()`. The output is a single document represented in a pretty-printed JSON format:

```
{
  "_id": ObjectId("598e01613ae3ad8abf1b8300"),
  "name": "standard desk chair",
  "price": 150,
  "brand": "CheapCo",
  "type": "chair"
}
```

Notice that each document has a key named `_id` that contains an `ObjectId`. As discussed previously, the default for new collections is to have the `autoIndexID` property set to true so the `_id` column is generated automatically when documents are inserted. The `_id` values generated in your collections will differ from the ones shown in the figures because characteristics of the host computer become encoded as part of the `_id` field, but they will always be unique within your collection.

Next, let's insert another document in the same collection, but with a slightly different structure. Remember, document databases are schema-less so all of the documents in a collection are not required to have the same structure.

```
db.products.insert({name: "cushioned desk chair",
    brand: "RoughRider",
    type: "chair",
    keywords: ["chair", "office", "cushioned"],
    price: 300})
```

The `insert()` method can be used with JSON documents of any complexity. The previous example includes an array of values for the `keywords` key.

find()

A MongoDB method to retrieve documents from a collection.

P-6 Updating Documents in MongoDB

There are many options when updating documents in MongoDB. At the most basic level, updates can either be **operator updates** or **replacement updates**. An operator update makes changes to some of the content of a document, but leaves the remainder of the document unchanged. This is similar to an update in a relational database. A replacement update completely removes the original document and replaces it with the new document while preserving only the value of the `_id` primary key. Both types of updates are performed with the `update()` method of the collection. The syntax for the `update()` method is:

```
db.<collection name>.update({<query>}, {<change>}, <upsert>, <multi>)
```

The `update()` method can have four parameters. First is a *query* object that is used to determine which documents to update. This is like the WHERE clause of a SQL UPDATE command. Only documents that match the query are updated.

The second parameter is a *change* object that specifies what changes to make to the document. If a replacement update is being performed, then the change object will be the new version of the document to be stored. If an operator update is being performed, then the change object will contain the functions that specify what changes should be made to the document.

The third parameter is the *upsert* flag. The `upsert` flag is Boolean, that is, it is either true or false. By default, `upsert` is false. If `upsert` is false, then if no documents match the criteria in the `query` object, then no changes will be made to the collection. On the other hand, if `upsert` is set to true, then if no documents match the criteria in the `query` object, then a new document will be inserted with the data in the `change` object. Upsert combines an update and an insert, hence the name. If there is a match, it performs an update; if there is not a match, it performs an insert.

The fourth parameter is the *multi* flag. Like `upsert`, `multi` is Boolean and false by default. If `multi` is false, then only the first document found to match the `query` object will be updated. If `multi` is set to true, then all documents found to match the `query` object will be updated. Both the `upsert` and `multi` flags can be omitted to apply the default values of false for those flags.

P-6a Replacement Updates

The following command performs a replacement update on the “standard desk chair” document that was first inserted into the `products` collection:

```
db.products.update({name: "standard desk chair"},  
                  {name: "regular desk chair"},  
                  false,  
                  false)
```

Looking at the first document returned in Figure P.5 shows that the entire document for “standard desk chair” has been replaced with a document that contains only the `name` property. Comparing with Figure P.4 shows that the `_id` remains the same for the document. In the previous code, the `query` object states that the update should match documents that have the value “standard desk chair” for the key `name`. The `change` object indicates that the new version of the document should contain only a `name` key with the value “regular desk chair”. Upset and `multi` flags are both false, so a new document will not be created if the `query` object does not match any existing documents, and only the first document found to match will be updated.

operator update

In MongoDB, an update that changes one or more values inside a document while leaving the remainder of the document unchanged.

replacement update

In MongoDB, an update that completely replaces an existing document with a new document while keeping only the `_id` (primary key) unchanged.

update()

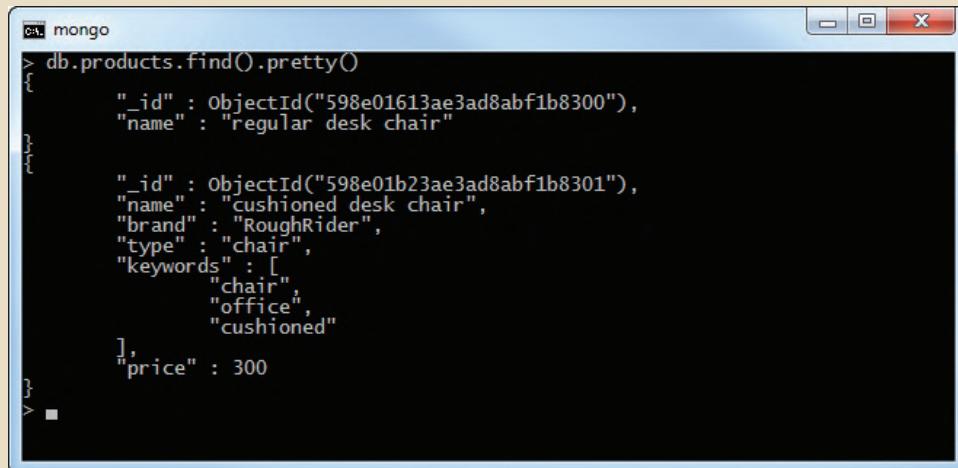
A MongoDB method for changing the contents of a document.

upsert

In MongoDB, a flag for use with updates that combines an insert operation and an update operation. When set to true, if no matching document is found for the criteria given, then a new document that matches the specified values will be created. If a matching document is found, then the existing document will be updated with the specified values.

The change object can include multiple *key:value* pairs, including complex values such as arrays and embedded objects. The following command replaces the same document with a new version:

FIGURE P.5 A REPLACEMENT UPDATE



A screenshot of a Windows-style application window titled "mongo". Inside the window, the mongo shell is running a command: "db.products.find().pretty()". The output shows two documents. The first document has fields: "_id": ObjectId("598e01613ae3ad8abf1b8300"), "name": "regular desk chair". The second document has fields: "_id": ObjectId("598e01b23ae3ad8abf1b8301"), "name": "cushioned desk chair", "brand": "RoughRider", "type": "chair", "keywords": ["chair", "office", "cushioned"], "price": 300.

```
db.products.update({name: "regular desk chair"},  
    {name: "basic desk chair", price: 100, brand: "RoughRider", type: "chair"})
```

Note that because both the upsert and multi flags are omitted, the default value of false is used for both.

P-6b Operator Updates with \$set, \$unset, and \$inc

Operator updates follow the same syntax as replacement updates, except in the change object. Operator updates use one or more update functions, or operators, in the change object. Common update operators are shown in Table P.1. Notice that MongoDB operators start with a \$, and each operator can accept an object as a value. The syntax for an operator is:

<operator>: {<object>}

TABLE P.1

MONGODB COMMON UPDATE OPERATORS

OPERATOR	DESCRIPTION
\$addToSet	Adds a value to an array if the value does not already exist in the array
\$each	Modifies \$push and \$addToSet to allow adding an array of values to an array
\$inc	Increments or decrements a value by the amount specified
\$pull	Removes all occurrences of a value from an array
\$pullAll	Removes all occurrences of an array of values from an array
\$push	Adds a value to an array even if adding the value creates duplicates
\$rename	Changes the value of the key portion of a key:value pair
\$set	Places a value in a key:value pair
\$unset	Removes a key:value pair

One of the most commonly used update operators is the **\$set** operator. The \$set operator works much like the SET clause of a SQL UPDATE command. With \$set, the programmer specifies the key:value pair to change in the object parameter. For example, the name of the “basic desk chair” from the previous example can be changed using the \$set operator. Because this is an operator update instead of a replacement update, the other key:value pairs in the document are not affected.

```
db.products.update({name: "basic desk chair"},  
{$set:  
  {name: "standard desk chair"}  
})
```

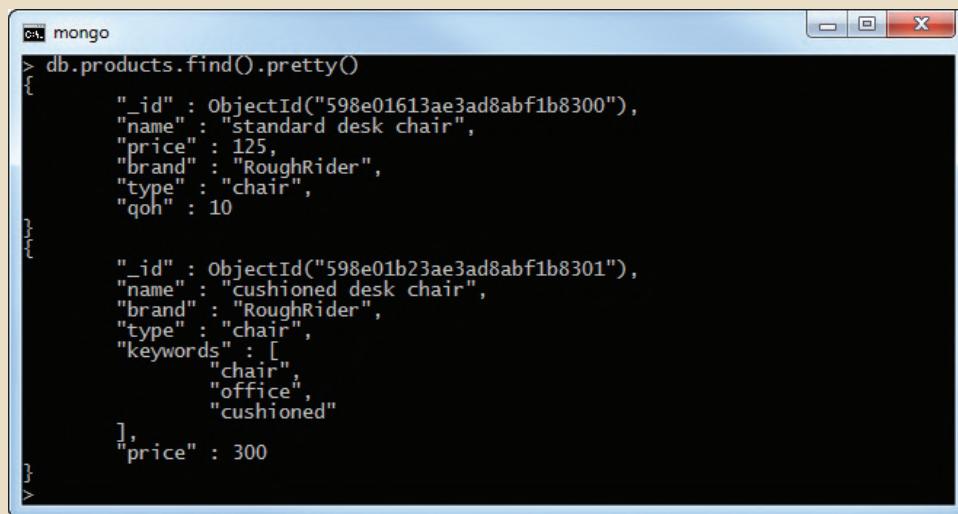
If the key specified in the \$set operator’s object parameter exists, then it is updated. If it does not exist, then the key is added to the document with the value specified. The following command is used to change the price to 125 and add quantity on hand (qoh) to the document. Figure P.6 shows the cumulative effect of these updates.

```
db.products.update({name: "standard desk chair"},  
{$set:  
  {price: 125, qoh: 10}  
})
```

\$set

A MongoDB operator used with the update() method to specify new values for the listed key:value pairs in a document.

FIGURE P.6 CUMULATIVE EFFECT OF UPDATES WITH \$SET



```
mongo
> db.products.find().pretty()
{
    "_id" : ObjectId("598e01613ae3ad8abf1b8300"),
    "name" : "standard desk chair",
    "price" : 125,
    "brand" : "RoughRider",
    "type" : "chair",
    "qoh" : 10
}

{
    "_id" : ObjectId("598e01b23ae3ad8abf1b8301"),
    "name" : "cushioned desk chair",
    "brand" : "RoughRider",
    "type" : "chair",
    "keywords" : [
        "chair",
        "office",
        "cushioned"
    ],
    "price" : 300
}
```

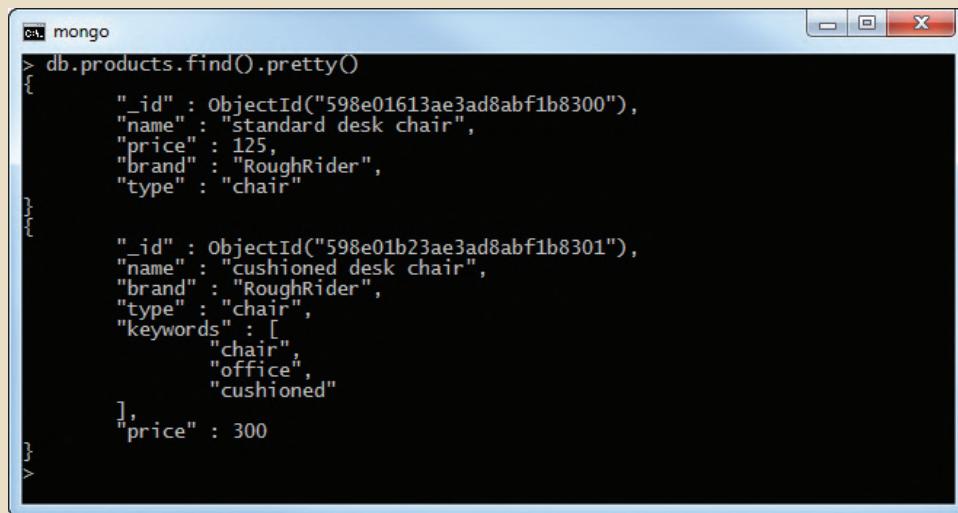
Just as the \$set operator can be used to add a new key:value pair to a document, the **\$unset** operator can be used to remove a selected key:value pair from a document. The following command removes the qoh key:value pair from the document. As shown in Figure P.7, the qoh key is removed.

```
db.products.update({name: "standard desk chair"},  
{$unset:  
  {qoh: 1}  
})
```

\$unset

A MongoDB operator used with the update() method to remove a key:value pair from a document.

FIGURE P.7 REMOVING A KEY WITH \$UNSET



```
mongo
> db.products.find().pretty()
{
    "_id" : ObjectId("598e01613ae3ad8abf1b8300"),
    "name" : "standard desk chair",
    "price" : 125,
    "brand" : "RoughRider",
    "type" : "chair"
}

{
    "_id" : ObjectId("598e01b23ae3ad8abf1b8301"),
    "name" : "cushioned desk chair",
    "brand" : "RoughRider",
    "type" : "chair",
    "keywords" : [
        "chair",
        "office",
        "cushioned"
    ],
    "price" : 300
}
```

Remember, all operators accept an object parameter. The object parameter must include a minimum of a key:value pair. Since \$unset is removing the pair, the actual value provided in the command is immaterial as long as some value is provided. In the previous example, the value “1” is provided for the qoh key. Using the value “1” with \$unset is common, but any value would work.

The following command changes the brand key to contain a subdocument (see Figure P.8).

```
db.products.update({name: "standard desk chair"},  
{$set:  
{brand:  
{name: "CheapCo",  
phone: "555-1212"}  
}})
```

FIGURE P.8 A DOCUMENT WITH A SUBDOCUMENT

```
mongo  
db.products.find().pretty()  
[  
{"_id": ObjectId("598e01613ae3ad8abf1b8300"),  
"name": "standard desk chair",  
"price": 125,  
"brand": {  
    "name": "CheapCo",  
    "phone": "555-1212"  
},  
"type": "chair"  
  
{"_id": ObjectId("598e01b23ae3ad8abf1b8301"),  
"name": "cushioned desk chair",  
"brand": "RoughRider",  
"type": "chair",  
"keywords": [  
    "chair",  
    "office",  
    "cushioned"  
],  
"price": 300  
]
```

The **\$inc** operator is used to increment or decrement a numeric value. The object parameter for the \$inc operator specifies the key whose value is to be changed and the change amount. For example, if we wish to increase the price of chairs by \$50, we could use the following command:

```
db.products.update({type: "chair"},  
{$inc:  
{price: 50},  
},  
false, true)
```

\$inc

A MongoDB operator used with the update() method to increment or decrement values in a key:value pair without having to retrieve the existing value first.

FIGURE P.9 MULTI-UPDATE USING \$INC

```
mongo
db.products.find().pretty()
{
    "_id" : ObjectId("598e01613ae3ad8abf1b8300"),
    "name" : "standard desk chair",
    "price" : 175,
    "brand" : {
        "name" : "CheapCo",
        "phone" : "555-1212"
    },
    "type" : "chair"
}

{
    "_id" : ObjectId("598e01b23ae3ad8abf1b8301"),
    "name" : "cushioned desk chair",
    "brand" : "RoughRider",
    "type" : "chair",
    "keywords" : [
        "chair",
        "office",
        "cushioned"
    ],
    "price" : 350
}
```

Figure P.9 shows that the previous command changes the price of both chairs by \$50. Although the price of either chair could have been changed using the \$set operator, that would require knowing the existing price of the chair. The advantage of the \$inc operator is that the current value does not have to be retrieved to calculate the new value. To decrease the value of an attribute, simply enter a negative value for the change amount.



Note

With the MongoDB update() method, a value for the upsert or multi flag can be set using an object containing a key:value pair instead of relying on the ordinal position within the command. For example, the previous update could have specified the value true for the multi flag as follows:

```
db.products.update({type:"chair"}, {$inc: {price: 50}}, {multi: true})
```

Updating the values in a subdocument can also be done using the \$set, \$unset, and \$inc operators. The only modification is that the name of the subdocument must be added to the name of the key in the subdocument that is to be modified. Dot (.) notation is used when adding the subdocument name to the key name with the format “<subdocument>.key”. To ensure that MongoDB correctly interprets the combined name as a single value, the combined name must be placed inside quotes. The following command changes the phone number for the embedded subdocument in the standard desk chair document (see Figure P.10).

```
db.products.update({name: "standard desk chair"},  
{$set:  
 {"brand.phone": "555-2121"}  
})
```

FIGURE P.10 UPDATING A SUBDOCUMENT

```
ca mongo  
db.products.find().pretty()  
[{"_id": ObjectId("598e01613ae3ad8abf1b8300"), "name": "standard desk chair", "price": 175, "brand": { "name": "CheapCo", "phone": "555-2121" }, "type": "chair"}, {"_id": ObjectId("598e01b23ae3ad8abf1b8301"), "name": "cushioned desk chair", "brand": "RoughRider", "type": "chair", "keywords": [ "chair", "office", "cushioned" ], "price": 350}]
```

P-6c Updating Arrays with \$push, \$pull, and \$addToSet

Changing values in an array can be slightly different. To change the values in an array, it is possible to use the \$set operator, but the entire array would have to be set at the same time. The cushioned chair document that was previously inserted included a “keywords” key that contains the array [“chair”, “office”, “cushioned”]. The additional keyword “fancy” could be added to the array using the \$set operator if the entire array is reentered, as follows:

```
db.products.update({name: "cushioned desk chair"},  
{$set:  
 [keywords: ["chair", "office", "cushioned", "fancy"]]}  
})
```

Although this produces the required result, it is not normal practice. In most cases, the programmer is attempting to add or remove values from the array, not replace every value in the array. Using \$set in this situation is inefficient because it would require first retrieving the entire array to know which values already exist in the array so that they can be included in the update operation. Typically, values are added to or removed from the array without having to retrieve the array first. This is done using the \$push and \$pull operators.

The **\$push** operator is used to add a value to an array. The **\$pull** operator is used to remove a value from an array. Other values in the array are unaffected by \$push and \$pull. Just like \$set and \$unset, \$push and \$pull accept an object parameter that describes the change to make. For example, to add the keyword “elegant” to the cushioned chair document, the following command could be used (results shown in Figure P.11):

```
db.products.update({name: "cushioned desk chair"},  
    {$push:  
        {keywords: "elegant"}  
    })
```

FIGURE P.11 ARRAY VALUE ADDED WITH \$PUSH

```
ca mongo  
> db.products.find().pretty()  
{  
    "_id" : ObjectId("598e01613ae3ad8abf1b8300"),  
    "name" : "standard desk chair",  
    "price" : 175,  
    "brand" : {  
        "name" : "CheapCo",  
        "phone" : "555-2121"  
    },  
    "type" : "chair"  
}  
  
{  
    "_id" : ObjectId("598e01b23ae3ad8abf1b8301"),  
    "name" : "cushioned desk chair",  
    "brand" : "RoughRider",  
    "type" : "chair",  
    "keywords" : [  
        "chair",  
        "office",  
        "cushioned",  
        "fancy",  
        "elegant"  
    ],  
    "price" : 350  
}
```

The query object tells the update() method to find a document with the value “cushioned desk chair” in the name key. The change object tells the method to push, or add, the value “elegant” to the array of values for the keywords key. Since the upsert and multi flags were omitted, they take the default value of false.

The \$pull operator has the exact same syntax as the \$push operator. The only difference is that \$pull removes the value from the array instead of adding it to the array. The keyword “office” can be removed from the array using the \$pull operator as follows (results shown in Figure P.12):

```
db.products.update({name: "cushioned desk chair"},  
    {$pull:  
        {keywords: "office"}  
    })
```

\$push

A MongoDB operator used with the update() method to add values, possibly duplicates, to an array in a document.

\$pull

A MongoDB operator used with the update() method to remove all occurrences of a value from an array in a document.

FIGURE P.12 ARRAY VALUE REMOVED WITH \$PULL

```
mongo
> db.products.find().pretty()
{
    "_id" : ObjectId("598e01613ae3ad8abf1b8300"),
    "name" : "standard desk chair",
    "price" : 175,
    "brand" : {
        "name" : "CheapCo",
        "phone" : "555-2121"
    },
    "type" : "chair"
}

{
    "_id" : ObjectId("598e01b23ae3ad8abf1b8301"),
    "name" : "cushioned desk chair",
    "brand" : "RoughRider",
    "type" : "chair",
    "keywords" : [
        "chair",
        "cushioned",
        "fancy",
        "elegant"
    ],
    "price" : 350
}
>
```

As you examine the results of the update in Figure P.12, notice that the other values in the array were not impacted.

It is possible, and in some cases desirable, for an array to contain duplicate values. If an array contains a list of websites visited by a user, we may wish to know every time the user visited a given website even if that means the website appears in the array multiple times. Other times, we do not want duplicates to appear in an array. In the keywords for our cushioned chair, there is no advantage to storing the keyword “elegant” multiple times. The \$push operator allows duplicate array values. If you were to re-execute the earlier command to push “elegant” to the keywords array, the value would appear more than once. To allow programmers to add a value to an array only if the value does not already exist in the array, the **\$addToSet** operator is used. The \$addToSet operator uses the same syntax and operates in the same manner as the \$push operator except that it does not add a value to the array if the value already exists in the array. The following command can be used to add the value “expensive” to the keywords array:

```
db.products.update({name: "cushioned desk chair"},  
{$addToSet:  
{keywords: "expensive"}  
})
```

This command adds “expensive” to the keyword array because the value “expensive” was not already in the array. If you use the \$addToSet operator with a value that already exists in the array, the value will not be added again, as shown below.

```
db.products.update({name: "cushioned desk chair"},  
{$addToSet:  
{keywords: "cushioned"}  
})
```

If a value appears in an array multiple times, the \$pull operator will remove every occurrence of that value from the array.

\$addToSet

A MongoDB operator used with the update() method to add values, suppressing duplicates, to an array in a document.

P-6d Updating Arrays with Arrays using \$each and \$pullAll

The previous operators are very good at working with individual values in arrays, whether it is a single, simple value or an object value. However, since arrays are designed to hold multiple values, we often want to modify the contents of an array with multiple values simultaneously. It is not possible to update an array with an array of values using \$push, \$pull, and \$addToSet. Luckily, only small modifications are needed to accomplish this task. To remove an array of values from an existing array, use the **\$pullAll** operator instead of the \$pull operator. The \$pullAll operator syntax is the same as the \$pull operator except that it accepts an array of values instead of a single value. The following command removes the values “fancy,” “expensive,” and “cushioned” from the keywords array, as shown in Figure P.13.

```
db.products.update({name: "cushioned desk chair"},  
{$pullAll:  
  {keywords: ["fancy", "expensive", "cushioned"]}  
})
```

FIGURE P.13 REMOVING AN ARRAY FROM AN ARRAY

```
mongo  
db.products.find().pretty()  
[  
  {  
    "_id": ObjectId("598e01613ae3ad8abf1b8300"),  
    "name": "standard desk chair",  
    "price": 175,  
    "brand": {  
      "name": "CheapCo"  
      "phone": "555-2121"  
    },  
    "type": "chair"  
  },  
  
  {  
    "_id": ObjectId("598e01b23ae3ad8abf1b8301"),  
    "name": "cushioned desk chair",  
    "brand": "RoughRider",  
    "type": "chair",  
    "keywords": [  
      "chair",  
      "elegant"  
    ],  
    "price": 350  
  }  
]
```

The \$push and \$addToSet operators can be used in conjunction with the **\$each** operator. The \$each operator modifies \$push and \$addToSet to iterate, or loop, over a set of values and act on each one separately. For example, if we want to add the values “heavy duty,” “leather,” and “adjustable” to the keywords array, the following command combines \$push and \$each to complete that task, as shown in Figure P.14.

```
db.products.update({name: "cushioned desk chair"},  
{$push:  
  {keywords:  
    {$each: ["heavy duty", "leather", "adjustable"]}  
  }  
})
```

\$pullAll

A MongoDB operator used with the update() method to remove an array of values from an array in a document.

\$each

A MongoDB operator used with the update() method that modifies the function of the \$push and \$addToSet operators to allow an array of values to be added to an array in a document.

FIGURE P.14 ADDING AN ARRAY TO AN ARRAY

```
mongo
> db.products.find().pretty()
{
    "_id" : ObjectId("598e01613ae3ad8abf1b8300"),
    "name" : "standard desk chair",
    "price" : 175,
    "brand" : {
        "name" : "CheapCo",
        "phone" : "555-2121"
    },
    "type" : "chair"

    "_id" : ObjectId("598e01b23ae3ad8abf1b8301"),
    "name" : "cushioned desk chair",
    "brand" : "RoughRider",
    "type" : "chair",
    "keywords" : [
        "chair",
        "elegant",
        "heavy duty",
        "leather",
        "adjustable"
    ],
    "price" : 350
}
```

Just as before, the \$push operator takes an object parameter. When adding a single value to the array, the object parameter is a simple key:value pair. When adding an array, the *value* in the key:value pair is an object that contains its own key:value pair with the \$each operator as the key and an array as the value. This syntax is also used when the \$each operator is used with the \$addToSet operator. The following command attempts to add the values “arms,” “heavy duty,” and “executive” to the keywords array, with the results shown in Figure 4.26.

```
db.products.update({name: "cushioned desk chair"},  
{$addToSet:  
  {keywords:  
    {$each: ["arms", "heavy duty", "executive"]}  
  }  
})
```

Looking at Figure P.15, notice that the value “heavy duty” appears only once because \$addToSet does not create duplicate values in the array.

FIGURE P.15 ADDING ONLY NEW VALUES TO AN ARRAY WITH AN ARRAY

```
mongo
> db.products.find().pretty()
{
    "_id" : ObjectId("598e01613ae3ad8abf1b8300"),
    "name" : "standard desk chair",
    "price" : 175,
    "brand" : {
        "name" : "CheapCo",
        "phone" : "555-2121"
    },
    "type" : "chair"
}

{
    "_id" : ObjectId("598e01b23ae3ad8abf1b8301"),
    "name" : "cushioned desk chair",
    "brand" : "RoughRider",
    "type" : "chair",
    "keywords" : [
        "chair",
        "elegant",
        "heavy duty",
        "leather",
        "adjustable",
        "arms",
        "executive"
    ],
    "price" : 350
}
```

P-6e Updating to Rename Keys with \$rename

All of the previous update operations have worked on changing the values in key:value pairs. It is also possible to change the key in the key:value pair by using the **\$rename** operator. The \$rename operator also uses an object parameter. In this case, the object parameter contains a simple, text key:value pair in which the key contains the name of the key to be renamed, and the value is the text that the key name should be changed to, as in {<old_name>:<“new_name”>}. The value component that contains the new name is read by MongoDB as a value when it is checking the syntax of the command. Therefore, the new value must be inside quotes because it is a text value. For example, the following command renames the price key to saleprice for all documents in the collection, as shown in Figure P.16:

```
db.products.update({},
{$rename:
  {price: "saleprice"}
},
{multi:true})
```

\$rename

A MongoDB operator used with the update() method to change the name of the key portion of a key:value pair in a document.

FIGURE P.16 RENAMING THE KEY IN A PAIR

```

mongosh
> db.products.find().pretty()
[{"_id": ObjectId("598e01613ae3ad8abf1b8300"), "name": "standard desk chair", "brand": {"name": "CheapCo", "phone": "555-2121"}, "type": "chair", "saleprice": 175}, {"_id": ObjectId("598e01b23ae3ad8abf1b8301"), "name": "cushioned desk chair", "brand": "RoughRider", "type": "chair", "keywords": ["chair", "elegant", "heavy duty", "leather", "adjustable", "arms", "executive"], "saleprice": 350}]

```

Notice that the query object in this command is empty. Including a query object is required, but the object is allowed to be empty. An empty query object matches every document, so combined with the *multi* flag being set to true, the update operation is performed on every document in the collection.

P-7 Deleting Documents in MongoDB

Deleting documents from a MongoDB collection is straightforward with few options. Documents are deleted using the [remove\(\)](#) method of the collection. The remove() method accepts a single query object as a parameter. Documents that match the query object are deleted from the collection. As with the update() method, the query object can be an empty object, which would delete every document in the collection, but that is rarely used in practice. The following code deletes the standard desk chair document, as shown in Figure P.17:

```
db.products.remove({name: "standard desk chair"})
```

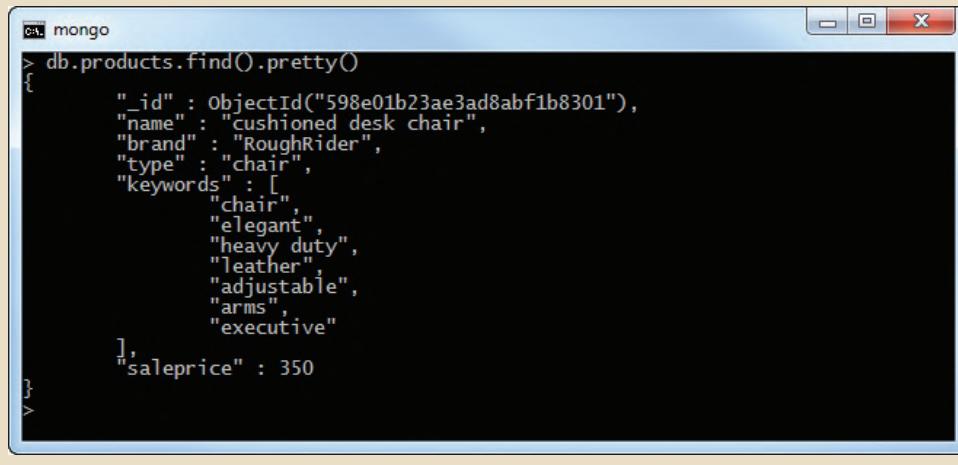
remove()

In MongoDB, a method to delete specified documents from a collection.

Note

Documents cannot be manually deleted in a capped collection.

FIGURE P.17 REMOVING A DOCUMENT FROM MONGODB



```
mongo
> db.products.find().pretty()
{
    "_id" : ObjectId("598e01b23ae3ad8abf1b8301"),
    "name" : "cushioned desk chair",
    "brand" : "RoughRider",
    "type" : "chair",
    "keywords" : [
        "chair",
        "elegant",
        "heavy duty",
        "leather",
        "adjustable",
        "arms",
        "executive"
    ],
    "saleprice" : 350
}
>
```

P-8 Querying Documents in MongoDB

The collection of documents used to illustrate MongoDB queries is based on the data used in earlier chapters for a small library. The portion of the model that is being used here consists of documents with *patron* as the central entity. The documents have the following structure:

```
{_id: <system-generated ObjectId>,
display: <the patron's full name as it will be displayed to users>,
fname: <patron's first name in all lowercase letters>,
lname: <patron's last name in all lowercase letters>,
type: <either "faculty" or "student">,
age: <patron's age in years only if the patron is a student>,
checkouts: <an array of objects for the patron's checkout history>
    [id: <an assigned number for this checkout object>,
     year: <the year in which this checkout occurred>,
     month:<the month in which this checkout occurred>,
     day: <the day of the month in which this checkout occurred>,
     book:<the book number of the book for this checkout>,
     title:<the title of the book>,
     pubyear: <the year the book was published>,
     subject:<the subject of the book>]
}
```



Online Content

The documents for the *fact* database are available as a collection of JSON documents that can be directly imported into MongoDB. The file is named Ch14_Fact.json and is available at www.cengagebrain.com.

Notice that the patron's name is stored twice, once with first and last name together with capitalization, and again with first name and last name in all lowercase letters in separate key:value pairs. All searches in MongoDB are case sensitive by default. Making a query case insensitive typically requires the use of a programming technique called regular expressions, which can be very detrimental to performance for MongoDB. Therefore, it is common in practice to store text values in which capitalization matters twice: once capitalized the way it should be displayed, and once in all uppercase or lowercase letters to simplify queries.



Note

The following section uses the *fact* database and the *patron* collection that was adapted from the Ch07_FACT database used in Chapter 7, Introduction to SQL.

Free Access to Computer Technology (FACT) is a small library run by the Computer Information Systems department at Tiny College. The database can be created using the Ch14_Fact.json file by using the following command at an operating system command prompt (note that the command is for use at a command prompt in the OS, not inside the MongoDB shell).

```
mongoimport --db fact --collection patron --type json --file Ch14_Fact.json
```

Mongoimport is an executable program that is installed with MongoDB that is used to import data into a MongoDB database. The above command specifies that the imported documents should be placed in the *fact* database (if one does not exist, it will be created) and in the *patron* collection (if one does not exist, it will be created). Mongoimport can work with different file types such as CSV files and JSON files. The type parameter specifies that the imported documents are already in JSON format. The file parameter specifies the file name of the file to be imported. If your copy of the Ch14_Fact.json file is not in the current directory for your command prompt, you will need to provide an appropriate path for the file location.

P-8a Selection and Restriction with the find() Method

As shown previously, the basic method used to retrieve documents in MongoDB is the *find()* method. The *find()* method of a collection retrieves documents from the collection that match the restrictions provided. The method accepts two object parameters: the *query* object that specifies criteria that the documents must match to be included and a *projection* object that specifies which keys will be included in the returned documents. Both parameters are optional. The syntax for the *find()* method is:

```
find({<query>}, {<projection>})
```



Note

To work along with the following commands, you must have imported the Ch14_Fact.json file as described earlier. Then enter the Mongo shell using the command:

```
mongo
```

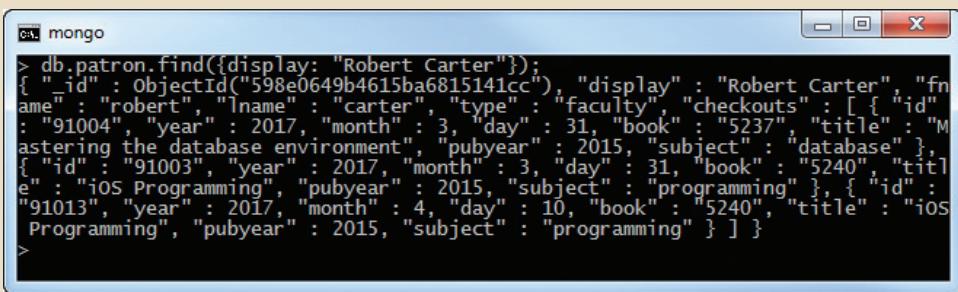
Then change to the *fact* database using the command:

```
use fact
```

The query object parameter functions similar to the WHERE clause of a SQL SELECT query. Providing a simple key:value pair in the query parameter is interpreted as an equality condition. For example, the following command retrieves the patron with the display name “Robert Carter,” as shown in Figure P.18:

```
db.patron.find({display: "Robert Carter"})
```

FIGURE P.18 RETRIEVING A DENSE DOCUMENT



A screenshot of a Windows-style application window titled "mongo". Inside, a command-line interface shows the output of a MongoDB query. The command is "db.patron.find({display: "Robert Carter"});". The result is a single document with many fields, including "_id", "display", "fn", "name", "lname", "type", "checkouts", "id", "year", "month", "day", "book", "title", and "subject". The "checkouts" field contains an array of objects, each with its own set of fields like "id", "year", "month", "day", "book", and "title".



Note

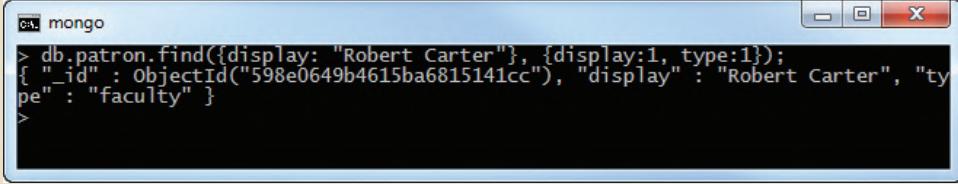
Notice that because the projection object parameter is optional, it can be omitted when the entire document should be returned.

In the previous command, the query object parameter is `{display: "Robert Carter"}`. This is the equivalent of `WHERE display = "Robert Carter"` in a SQL SELECT query.

The projection object parameter is used to specify which key:value pairs to return. The projection object contains one or more key:value pairs. The keys in the projection objects are the keys from the document to be projected. The value with each key in the projection object is either the value 0 or 1. Specifying the value 1 with a key indicates that that key:value pair from the document should be included in the results. Specifying the value 0 with a key indicates that that key:value pair from the document should be omitted. For example, the following command retrieves the document for patron “Robert Carter” but only returns the `_id`, display name, and type keys, as shown in Figure P.19:

```
db.patron.find({display: "Robert Carter"}, {display:1, type:1})
```

FIGURE P.19 PROJECTING KEY:VALUE PAIRS



A screenshot of a mongo shell window titled "mongo". The command is "db.patron.find({display: "Robert Carter"}, {display:1, type:1});". The result is a single document with only two fields: "_id" and "display". The "display" field has a value of "Robert Carter". The "type" field has a value of "faculty".

Notice that the `_id` key is returned by default even when it is not specifically asked for. To return all keys except the ones specified, use the `0` value in the projection object. For example, the following command returns all keys in the Robert Carter document except the checkouts, as shown in Figure P.20:

```
db.patron.find({display: "Robert Carter"}, {checkouts: 0})
```

FIGURE P.20 EXCLUDING A PAIR FROM THE OUTPUT

```
mongo
> db.patron.find({display: "Robert Carter"}, {checkouts: 0});
{ "_id" : ObjectId("598e0649b4615ba6815141cc"), "display" : "Robert Carter", "fname" : "robert", "lname" : "carter", "type" : "faculty" }
```

Generally, it is not possible to specifying including and excluding keys in the same query. The exception is the `_id` key that is returned by default. When specifying keys to include, if the programmer wishes to suppress the `_id` key, it can be explicitly excluded. The following command returns just the `display` name key of all faculty patrons, as shown in Figure P.21:

```
db.patron.find({type: "faculty"}, {display:1, _id:0})
```

FIGURE P.21 MIXING INCLUSION AND EXCLUSION IN PROJECTIONS

```
mongo
> db.patron.find({type: "faculty"}, {display:1, _id:0});
{ "display" : "Robert Carter" }
{ "display" : "Helena Hughes" }
{ "display" : "Kelsey Koch" }
{ "display" : "Carlton Morton" }
{ "display" : "Cedric Baldwin" }
{ "display" : "Cory Barry" }
```

pretty()

In MongoDB, a method that can be chained to the `find()` method to improve the readability of retrieved documents through the use of line breaks and indentation.

The JSON document is returned in dense format, where each key:value pair is separated by a space. This is the default view of documents and is appropriate when the documents are very simple or when the document is being returned to an application, not for human readability. To improve the readability of the returned document, the `pretty()` method can be used. The `pretty()` method is a MongoDB method to improve the readability of documents by placing key:value pairs on separate lines. The `pretty()` method can be added to the `find()` method, and it does not take any parameters. The following command combines the `find()` and `pretty()` methods, as shown in Figure P.22.

```
db.patron.find({display: "Robert Carter"}).pretty()
```

FIGURE P.22 USING THE PRETTY() METHOD

The screenshot shows a Windows command-line window titled "mongo". The command entered was `db.patron.find().pretty()`. The output is a JSON document representing multiple documents from a collection named "patron". Each document includes fields like _id, display, fname, lname, type, and checkouts, which itself is an array of book objects.

```
{
  "_id" : ObjectId("598e0649b4615ba6815141cc"),
  "display" : "Robert Carter",
  "fname" : "robert",
  "lname" : "carter",
  "type" : "faculty",
  "checkouts" : [
    {
      "id" : "91004",
      "year" : 2017,
      "month" : 3,
      "day" : 31,
      "book" : "5237",
      "title" : "Mastering the database environment",
      "pubyear" : 2015,
      "subject" : "database"
    },
    {
      "id" : "91003",
      "year" : 2017,
      "month" : 3,
      "day" : 31,
      "book" : "5240",
      "title" : "iOS Programming",
      "pubyear" : 2015,
      "subject" : "programming"
    },
    {
      "id" : "91013",
      "year" : 2017,
      "month" : 4,
      "day" : 10,
      "book" : "5240",
      "title" : "iOS Programming",
      "pubyear" : 2015,
      "subject" : "programming"
    }
  ]
}
```

Recall that both the query object parameter and the projection object parameter are optional. If both are omitted, then the `find()` method is written with no parameters. If the query object parameter is needed, but the projection object parameter is not, then just the projection object can be omitted. However, if the query object parameter is not needed, but the projection object is, then the programmer cannot simply omit the query object. If only one object parameter is provided, then MongoDB will assume that the one parameter is intended to be the query object parameter. In these cases, an empty object can be used for the query object similar to the way an empty object was used for updating all documents previously. For example, the following command retrieves only the name key of every document, as shown in Figure P.23:

```
db.patron.find({}, {display:1, _id:0})
```

Notice in Figure P.23 that when working in the MongoDB shell, if a query returns more than 20 documents, only the first 20 are displayed on screen followed by a prompt to type “it” to display the next 20. In these cases, all documents are found and returned, but the shell is performing a simple manipulation to reduce scrolling the screen with data.

FIGURE P.23 ITERATING OVER MANY DOCUMENTS

```
> db.patron.find({}, {display:1, _id:0});
"display" : "Robert Carter"
"display" : "Allen Horne"
"display" : "Jimmie Love"
"display" : "Helena Hughes"
"display" : "Sandra Yang"
"display" : "Holly Anthony"
"display" : "Thomas Duran"
"display" : "Tyler Pope"
"display" : "Iva Ramos"
"display" : "Rena Mathis"
"display" : "Keith Cooley"
"display" : "Kelsey Koch"
"display" : "Desiree Rivas"
"display" : "Marina King"
"display" : "Maureen Downs"
"display" : "Iva McClain"
"display" : "Angel Terrell"
"display" : "Desiree Harrington"
"display" : "Carlton Morton"
"display" : "Cedric Baldwin"
Type "it" for more
it
"display" : "Cory Barry"
"display" : "Gloria Pitts"
"display" : "Zach Kelly"
"display" : "Jose Hayes"
"display" : "Jewel England"
"display" : "Wilfred Fuller"
"display" : "Jeff Owens"
"display" : "Homer Goodman"
"display" : "Peggy Marsh"
"display" : "Jerald Gaines"
"display" : "Betsy Malone"
"display" : "Tony Miles"
```

P-8b Document Pagination with limit() and skip()

method chaining
In MongoDB, serially listing multiple methods of a collection in the same command to perform multiple operations on the collection simultaneously.

limit()
In MongoDB, a method that can be chained to the find() method to restrict output to a specified number of documents to be retrieved. Often used for pagination of results.

The MongoDB shell's behavior of displaying only a few documents at a time can be helpful, but is rather limited and might not meet the programmer's needs. Other methods can be used with the results of the find() method to help improve the flow of documents both when viewing them interactively in the shell or when outputting them to an application.

When multiple methods are applied to the same collection, this is referred to as **method chaining**. Using the pretty() method with find() in an earlier example was a simple example of method chaining. Other methods can also be chained with the find() method. Among the most common are limit(), skip(), and sort(). The **limit()** method is used to restrict the number of documents returned. Limit() takes a single numeric parameter to specify the number of documents to limit to. Unlike the query object parameter of the find() method, limit() is not a query based on the content of the documents. After the find() method has determined which documents to return, the limit() method simply restricts the result set to the number of documents specified. For example, the following command finds all documents for student patrons but limits the results to just the first two documents.

```
db.patron.find({type: "student"}, {display:1}).limit(2)
```

`Limit()` is often used with applications that are retrieving data for display to users so that the data can be paginated into chunks by the application or website. The `skip()` function is also used to support pagination by allowing retrieval of subsequent chunks of documents. For example, if we are retrieving student patrons so that they can be listed for display, the previous `limit()` command indicates that we are using chunks of two documents at a time. To retrieve the next page of two documents, we would want to skip the first two documents because they have already been displayed and then display the next two documents. The following command would do that, as shown in Figure P.24:

```
db.patron.find({type: "student"}, {display:1}).skip(2).limit(2)
```

FIGURE P.24 USING LIMIT() AND SKIP() FOR PAGINATION

```
mongo
> db.patron.find({type: "student"}, {display:1}).limit(2);
   "_id" : ObjectId("598e0649b4615ba6815141cd"), "display" : "Allen Horne"
   "_id" : ObjectId("598e0649b4615ba6815141ce"), "display" : "Jimmie Love"

> db.patron.find({type: "student"}, {display:1}).skip(2).limit(2);
   "_id" : ObjectId("598e0649b4615ba6815141d0"), "display" : "Sandra Yang"
   "_id" : ObjectId("598e0649b4615ba6815141d1"), "display" : "Holly Anthony"
```



Note

If you know that the document you want will be the first document returned, instead of chaining `find()` and `limit(1)`, there is a shorthand method named `findOne()` that returns only the first document.

```
db.patron.findOne({type: "student"})
```

is the same as

```
db.patron.find({type: "student"}).limit(1)
```

To retrieve the third paginated chunk of documents, the `skip` parameter would be increased to 4.

P-8c Sorting Documents with `sort()`

Thus far, we have not concerned ourselves with the order of the documents that are returned. Due to the internal storage mechanisms used by MongoDB, documents are often returned in the order in which they were entered into the collection. However, just as with SELECT queries, we often want to control the sorted order of the data that is being returned. The `sort()` method allows us to do this. The `sort()` method of a collection orders the retrieved documents based on the values of one or more keys. Similar to the projection object parameter of the `find()` method, the `sort()` method takes an object of one or more keys paired with numeric values. In the case of projection, the values were 0 or 1. With sorting, the values are 1 or -1. If a key is associated with the value 1 in the `sort()` method, then the documents will be sorted on ascending values in that key. If the

skip()

In MongoDB, a method that can be chained to the `find()` method to specify a number of documents to skip over in the output of documents that have been retrieved. Often used with `limit()` for pagination of results.

sort()

In MongoDB, a method that can be chained to the `find()` method to order documents being returned based on the value of one or more key:value pairs.

key is associated with the value `-1` in the `sort()` method, then the documents will be sorted in descending order. Just as with the `ORDER BY` clause of a SQL `SELECT` query, when multiple sort attributes are specified, the results are sorted based on the attributes from left to right. For example, the following command retrieves the student patron documents, projects the display name and age, then sorts the results in descending order by age, and then within matching values for age the documents are sorted in ascending order by last name, as shown in Figure P.25:

```
db.patron.find({type: "student"}, {display:1, age:1, _id:0}).sort({age: -1, lname:1})
```

FIGURE P.25 SORTING DOCUMENTS

The screenshot shows a Windows-style application window titled 'mongo'. Inside, the MongoDB shell is running a command to find student patrons and sort them by age in descending order and then by last name in ascending order. The output lists 30 documents, each containing a 'display' name and an 'age' value. The names are sorted by age from oldest to youngest, and within each age group, they are sorted by last name.

```
> db.patron.find({type: "student"}, {display:1, age:1, _id:0}).sort({age: -1, lname:1})
{
  "display" : "Jerald Gaines", "age" : 41
  "display" : "Sandra Yang", "age" : 34
  "display" : "Jimmie Love", "age" : 29
  "display" : "Desiree Harrington", "age" : 28
  "display" : "Keith Cooley", "age" : 27
  "display" : "Holly Anthony", "age" : 25
  "display" : "Betsy Malone", "age" : 24
  "display" : "Iva Ramos", "age" : 24
  "display" : "Wilfred Fuller", "age" : 23
  "display" : "Homer Goodman", "age" : 23
  "display" : "Zach Kelly", "age" : 23
  "display" : "Rena Mathis", "age" : 23
  "display" : "Tony Miles", "age" : 23
  "display" : "Jeff Owens", "age" : 23
  "display" : "Marina King", "age" : 22
  "display" : "Iva McClain", "age" : 22
  "display" : "Gloria Pitts", "age" : 22
  "display" : "Maureen Downs", "age" : 21
  "display" : "Jewel England", "age" : 21
  "display" : "Peggy Marsh", "age" : 21
}
Type "it" for more
it
{
  "display" : "Desiree Rivas", "age" : 21
  "display" : "Jose Hayes", "age" : 20
  "display" : "Allen Horne", "age" : 20
  "display" : "Angel Terrell", "age" : 20
  "display" : "Tyler Pope", "age" : 19
  "display" : "Thomas Duran", "age" : 18
}
```

Interestingly, MongoDB does not care which order the `limit()`, `skip()`, and `sort()` methods are chained after the `find()` method. So the following three commands return the exact same result:

```
db.patron.find({type: "student"}, {display:1, age:1, _id:0}).sort({age: -1}).limit(5).skip(5)
db.patron.find({type: "student"}, {display:1, age:1, _id:0}).limit(5).skip(5).sort({age: -1})
db.patron.find({type: "student"}, {display:1, age:1, _id:0}).limit(5).sort({age: -1}).skip(5)
```

P-8d Queries Using Inequalities in MongoDB

Thus far, all of the queries that we've looked at use at most one criteria in the query object with a simple equality condition. Inequalities such as greater than, less than, and not equal to are also possible, but require the use of functions. Table P.2 lists the functions used in MongoDB for inequality comparisons.

TABLE P.2

MONGODB INEQUALITY FUNCTIONS

FUNCTION	DESCRIPTION
\$gt	Greater than >
\$gte	Greater than or equal to ≥
\$lt	Less than <
\$lte	Less than or equal to ≤
\$ne	Not equal ≠

All of the inequality functions operate the same way and have the same syntax. The functions are used as key for a key:value pair. The value is the value that is being applied to the function. For example, the \$gt function is used to match values that are greater than the value provided. Therefore, {\$gt:20} would mean greater than 20. The function's key:value pair is used as an object value for the key that the function is to work within. If we want to limit to documents where *age* > 20, then we would place the function object {\$gt:20} as the value in a pair with the key *age*, like *age*:{\$gt:20} in the query object parameter. Similarly, if we want to find the documents where the age ≤ 30, the criteria would be written *age*:{\$lte:30} because \$lte is the *less than or equal to* function, 30 is the target value for the function, and the function is being used with the *age* key. The following command retrieves the display name and age from documents for patrons that are age 30 or less, as shown in Figure P.26:

```
db.patron.find({age: {$lte:30} }, {display:1, age:1}).sort({age: -1})
```

FIGURE P.26 USING INEQUALITIES

```

> db.patron.find({age: {$lte:30} }, {display:1, age:1}).sort({age: -1});
{
  "_id": ObjectId("598e0649b4615ba6815141ce"), "display": "Jimmie Love", "age": 29
}
{
  "_id": ObjectId("598e0649b4615ba6815141dd"), "display": "Desiree Harrington", "age": 28
}
{
  "_id": ObjectId("598e0649b4615ba6815141d6"), "display": "Keith Cooley", "age": 27
}
{
  "_id": ObjectId("598e0649b4615ba6815141d1"), "display": "Holly Anthony", "age": 25
}
{
  "_id": ObjectId("598e0649b4615ba6815141d4"), "display": "Iva Ramos", "age": 24
}
{
  "_id": ObjectId("598e0649b4615ba6815141ea"), "display": "Betsy Malone", "age": 24
}
{
  "_id": ObjectId("598e0649b4615ba6815141d5"), "display": "Rena Mathis", "age": 23
}
{
  "_id": ObjectId("598e0649b4615ba6815141e2"), "display": "Zach Kelly", "age": 23
}
{
  "_id": ObjectId("598e0649b4615ba6815141e5"), "display": "Wilfred Fuller", "age": 23
}
{
  "_id": ObjectId("598e0649b4615ba6815141e6"), "display": "Jeff Owens", "age": 23
}
{
  "_id": ObjectId("598e0649b4615ba6815141e7"), "display": "Homer Goodman", "age": 23
}
{
  "_id": ObjectId("598e0649b4615ba6815141eb"), "display": "Tony Miles", "age": 23
}
{
  "_id": ObjectId("598e0649b4615ba6815141d9"), "display": "Marina King", "age": 22
}
{
  "_id": ObjectId("598e0649b4615ba6815141db"), "display": "Iya McClain", "age": 22
}
{
  "_id": ObjectId("598e0649b4615ba6815141e1"), "display": "Gloria Pitts", "age": 22
}
{
  "_id": ObjectId("598e0649b4615ba6815141d8"), "display": "Desiree Rivas", "age": 21
}
{
  "_id": ObjectId("598e0649b4615ba6815141da"), "display": "Maureen Downs", "age": 21
}
{
  "_id": ObjectId("598e0649b4615ba6815141e4"), "display": "Jewel England", "age": 21
}
{
  "_id": ObjectId("598e0649b4615ba6815141e8"), "display": "Peggy Marsh", "age": 21
}
{
  "_id": ObjectId("598e0649b4615ba6815141cd"), "display": "Allen Horne", "age": 20
}
Type "it" for more
> it
{
  "_id": ObjectId("598e0649b4615ba6815141dc"), "display": "Angel Terrell", "age": 20
}
{
  "_id": ObjectId("598e0649b4615ba6815141e3"), "display": "Jose Hayes", "age": 20
}
{
  "_id": ObjectId("598e0649b4615ba6815141d3"), "display": "Tyler Pope", "age": 19
}
{
  "_id": ObjectId("598e0649b4615ba6815141d2"), "display": "Thomas Duran", "age": 18
}

```

P-8e Combining Criteria with \$and and \$or

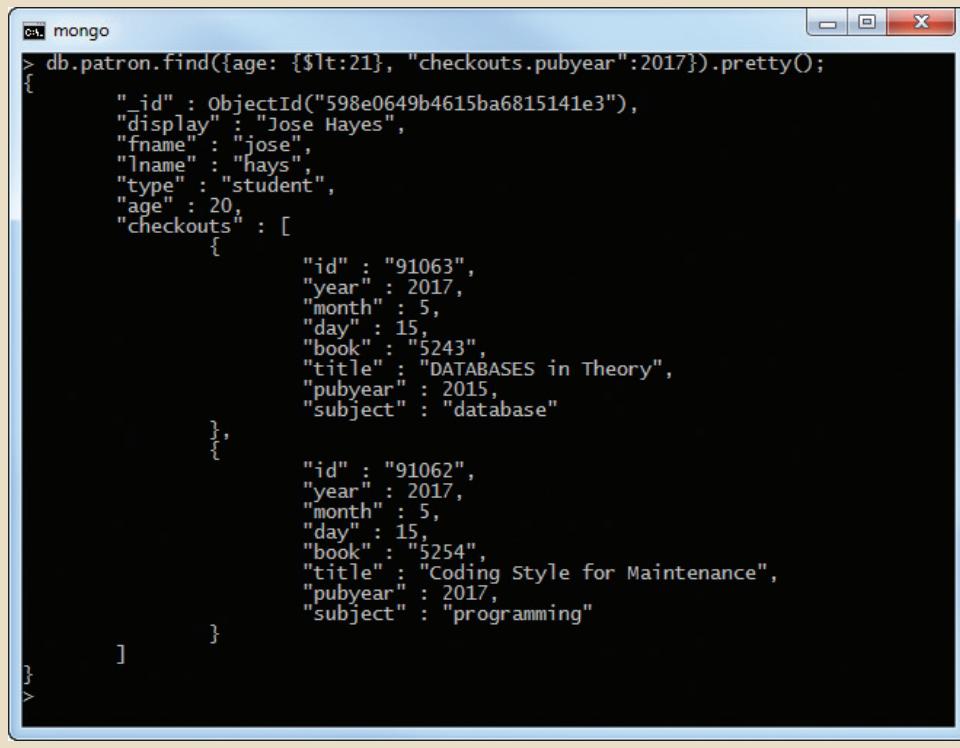
implicit and

In MongoDB, the combining of multiple criteria with a logical AND connector by separating the criteria with commas.

As you would expect, MongoDB provides the ability to combine multiple criteria in the query object parameter of the `find()` method using logical *and* and *or* operators. Combining criteria with a logical operator can be done both implicitly and explicitly. Separating criteria with commas in the query object employs an **implicit and**. For example, the following query finds the patrons who are under 21 years of age and have checked out a book published in 2017. To improve readability, the result is formatted with the `pretty()` method, as shown in Figure P.27.

```
db.patron.find({age: {$lt:21}, "checkouts.pubyear":2017}).pretty()
```

FIGURE P.27 IMPLICIT LOGICAL AND



The screenshot shows a Windows-style application window titled "mongo". Inside, a command is entered and its results are displayed. The command is:

```
> db.patron.find({age: {$lt:21}, "checkouts.pubyear":2017}).pretty();
```

The resulting document is:

```
{
  "_id" : ObjectId("598e0649b4615ba6815141e3"),
  "display" : "Jose Hayes",
  "fname" : "jose",
  "lname" : "hays",
  "type" : "student",
  "age" : 20,
  "checkouts" : [
    {
      "id" : "91063",
      "year" : 2017,
      "month" : 5,
      "day" : 15,
      "book" : "5243",
      "title" : "DATABASES in Theory",
      "pubyear" : 2015,
      "subject" : "database"
    },
    {
      "id" : "91062",
      "year" : 2017,
      "month" : 5,
      "day" : 15,
      "book" : "5254",
      "title" : "Coding Style for Maintenance",
      "pubyear" : 2017,
      "subject" : "programming"
    }
  ]
}
```

\$and

A MongoDB operator to explicitly combine multiple criteria with a logical AND connector by placing the criteria as objects within an array of criteria.

explicit and

In MongoDB, the combining of multiple criteria with a logical AND connector through the use of the `$and` operator.

Remember, the year of publication is in a subdocument so the “`checkouts.pubyear`” notation is required. The previous command contained two criteria: `age:{$lt:21}` and `“checkouts.pubyear”:2017`. Because these criteria are separated by a comma, MongoDB combines them with an implicit logical *and*.

Explicit logical *and* requires the use of the `$and` function. In practice, **explicit and** using the `$and` function tends to be preferred. With implicit *and*, all criteria in the list are always evaluated to true or false. With an explicit *and*, as soon as any criteria evaluates to false for a document, the remainder of the criteria are skipped for that document. This makes the explicit *and* faster and more efficient than an implicit *and* for most operations. The `$and` function appears as a key in a key:value pair that accepts an array of objects as the target value. For example, if `x` and `y` are criteria, then the syntax of `$and` would be

`$and: [{x}, {y}]`. Notice that each criterion is enclosed in {} to indicate that it is an object. The following command retrieves all documents that include a student over 26 years old checking out a book within the database subject, as shown in Figure P.28.

```
db.patron.find({$and: [{type: "student"}, {age: {$gte: 27}}, {"checkouts.subject": "database"}]}, {display: 1, checkouts:1, _id: 0}).pretty()
```

FIGURE P.28 EXPLICIT LOGICAL AND USING \$AND

```
mongo
> db.patron.find({$and: [{type: "student"}, {age: {$gte: 27}}, {"checkouts.subject": "database"}]}, {display: 1, checkouts:1, _id: 0}).pretty()
{
  "display" : "Desiree Harrington",
  "checkouts" : [
    {
      "id" : "91044",
      "year" : 2017,
      "month" : 4,
      "day" : 30,
      "book" : "5248",
      "title" : "What You Always Wanted to Know About Database, But Were Afraid to Ask",
      "pubyear" : 2016,
      "subject" : "database"
    }
  ]
}
>
```

Again, notice that each criterion is contained inside its own set of brackets so that it is an object all to itself. The criteria objects are separated with a comma.

All logical *or* operations are explicit. The **\$or** function has the same syntax as the \$and function. The following command retrieves documents that are for faculty or for students over 30 years of age. Only the display name, type, and age will be displayed with the results sorted by age, as shown in Figure P.29:

```
db.patron.find({$or: [{type: "faculty"}, {age: {$gt:30}}]}, {display:1, type:1, age: 1, _id:0}).sort({age:1})
```

FIGURE P.29 EXPLICIT LOGICAL OR USING \$OR

```
mongo
> db.patron.find({$or: [{type: "faculty"}, {age: {$gt:30}}]}, {display:1, type:1, age: 1, _id:0}).sort({age:1})
{
  "display" : "Robert Carter",
  "type" : "faculty"
},
{
  "display" : "Helena Hughes",
  "type" : "faculty"
},
{
  "display" : "Kelsey Koch",
  "type" : "faculty"
},
{
  "display" : "Carlton Morton",
  "type" : "faculty"
},
{
  "display" : "Cedric Baldwin",
  "type" : "faculty"
},
{
  "display" : "Cory Barry",
  "type" : "faculty"
},
{
  "display" : "Sandra Yang",
  "type" : "student",
  "age" : 34
},
{
  "display" : "Jerald Gaines",
  "type" : "student",
  "age" : 41
}
```

As MQL queries become more complex, they can seem confusing because of all of the punctuation. Breaking down the previous query, the symbols become a little clearer.

- The query is based on the db.patron.find() method.
 - Within the find() method there are two objects, the query object and the projection object.
 - The query object is `[$or: [{type: "faculty"}, {age: {$gt:30}}]]`

\$or

A MongoDB operator to explicitly combine multiple criteria with a logical OR connector by placing the criteria as objects within an array of criteria.

- The query object contains one key:value pair. \$or is the key and [{type: "faculty"}, {age: {\$gt:30}}] is the value.
- The value for the \$or key is an array composed of two objects. The first object is {type: "faculty"}, and the second object is {age: {\$gt:30}}.
- The first object in the array is a simple key:value pair, that is, type: "faculty".
- The second object in the array is a key:value pair with age as the key and an object for the value. The object value is {\$gt:30}, which calls the *greater than* function with 30 as the target value of the function.
- The projection object contains a list of key:value pairs.
- The projection object is {display:1, type:1, age:1, _id:0}
 - The keys in the object associated with value 1 are included in the result. The key in the object associated with the value 0 is excluded from the result.
- The sort() method is chained to the find() method to specify a sorted order for the output.
 - The sort() method contains an object made of simple key:value pairs.
 - The keys in the sort object associated with the value 1 are sorted in ascending order. If any key has been associated with the value -1, it would have been sorted in descending order.

The \$and and \$or functions can be mixed within the same query object. The following query, with results shown in Figure P.30, retrieves the _id, display name, and age for patrons that either have the last name “barry” and are faculty, or have the last name “hays” and are under 30 years old:

```
db.patron.find({$or: [
  {$and: [{lname: "barry"}, {type: "faculty"}]},
  {$and: [{lname: "hays"}, {age: {$lt: 30}}]}
]}, 
{display: 1, age: 1, type: 1}).pretty()
```

FIGURE P.30 MIXING LOGICAL AND WITH LOGICAL OR IN THE SAME QUERY

```
mongo
> db.patron.find({$or: [
...   {$and: [{lname: "barry"}, {type: "faculty"}]},
...   {$and: [{lname: "hays"}, {age: {$lt: 30}}]}
... ]}, 
...   {display: 1, age: 1, type: 1}).pretty()
{
  "_id" : ObjectId("598e0649b4615ba6815141e0"),
  "display" : "Cory Barry",
  "type" : "faculty"
}

{
  "_id" : ObjectId("598e0649b4615ba6815141e3"),
  "display" : "Jose Hayes",
  "type" : "student",
  "age" : 20
}
```

P-8f Matching Elements in Arrays with \$elemMatch

As with performing updates, working with arrays can pose unique challenges when writing queries. Consider the following requirement: find the patrons who have checked out a book in the programming subject that was published in 2015. Based on our coverage of \$and and using dot notation with subdocuments, we can craft the following query:

```
db.patron.find({$and: [{"checkouts.subject": "programming"}, {"checkouts.pubyear": 2015}]})pretty()
```

Carefully examining the results of that query (partial results shown in Figure P.31) will illustrate the problem of querying arrays.

FIGURE P.31 PARTIAL RESULTS OF CHECKOUTS FOR PROGRAMMING BOOKS

The screenshot shows a mongo shell window with the command and its output. The command is:

```
db.patron.find({$and: [{"checkouts.subject": "programming"}, {"checkouts.pubyear": 2015}]})pretty()
```

The output displays several patron documents, each with their details and a list of checkouts. One checkout entry per patron is shown, illustrating the challenge of querying arrays directly.

```

{
  "_id": ObjectId("598e0649b4615ba6815141ea"),
  "display": "Betsy Malone",
  "fname": "betsy",
  "lname": "malone",
  "type": "student",
  "age": 24,
  "checkouts": [
    {
      "id": "91007",
      "year": 2017,
      "month": 4,
      "day": 5,
      "book": "5244",
      "title": "Cloud-based Mobile Applications",
      "pubyear": 2015,
      "subject": "cloud"
    },
    {
      "id": "91033",
      "year": 2017,
      "month": 4,
      "day": 23,
      "book": "5235",
      "title": "Beginner's Guide to JAVA",
      "pubyear": 2014,
      "subject": "programming"
    }
  ]
},
{
  "_id": ObjectId("598e0649b4615ba6815141eb"),
  "display": "Tony Miles",
  "fname": "tony",
  "lname": "miles",
  "type": "student",
  "age": 23,
  "checkouts": [
    {
      "id": "91025",
      "year": 2017,
      "month": 4,
      "day": 21,
      "book": "5246",
      "title": "Capture the Cloud",
      "pubyear": 2016,
      "subject": "cloud"
    },
    {
      "id": "91032",
      "year": 2017,
      "month": 4,
      "day": 23,
      "book": "5238",
      "title": "Conceptual Programming",
      "pubyear": 2015,
      "subject": "programming"
    },
    {
      "id": "91061",
      "year": 2017,
      "month": 5,
      "day": 15,
      "book": "5246",
      "title": "Capture the Cloud",
      "pubyear": 2016,
      "subject": "cloud"
    }
  ]
}

```

The results include documents that have checkouts of books in the programming subject, and books published in 2015, so both criteria are met. However, looking through the output we find a few patrons who checked out a book that was published in 2015 and who checked out a book on programming, but they were not the same book! Our intention was that the same book satisfies both criteria, not two separate books each satisfying one criteria apiece. This problem is solved with the **\$elemMatch** function. In mathematical terminology, items in a set are called elements. It is, therefore, appropriate to refer to the values in an array as elements. The \$elemMatch function accepts an object containing a list of criteria as a parameter. The \$elemMatch function requires that all criteria included in the function be satisfied by the same element in an array. In the current database, the elements in the checkouts array are objects that represent each checkout by a patron. Using \$elemMatch with the criteria in the previous command requires both criteria to be satisfied by the same checkout, as shown in the following command (see Figure P.32 for partial results).

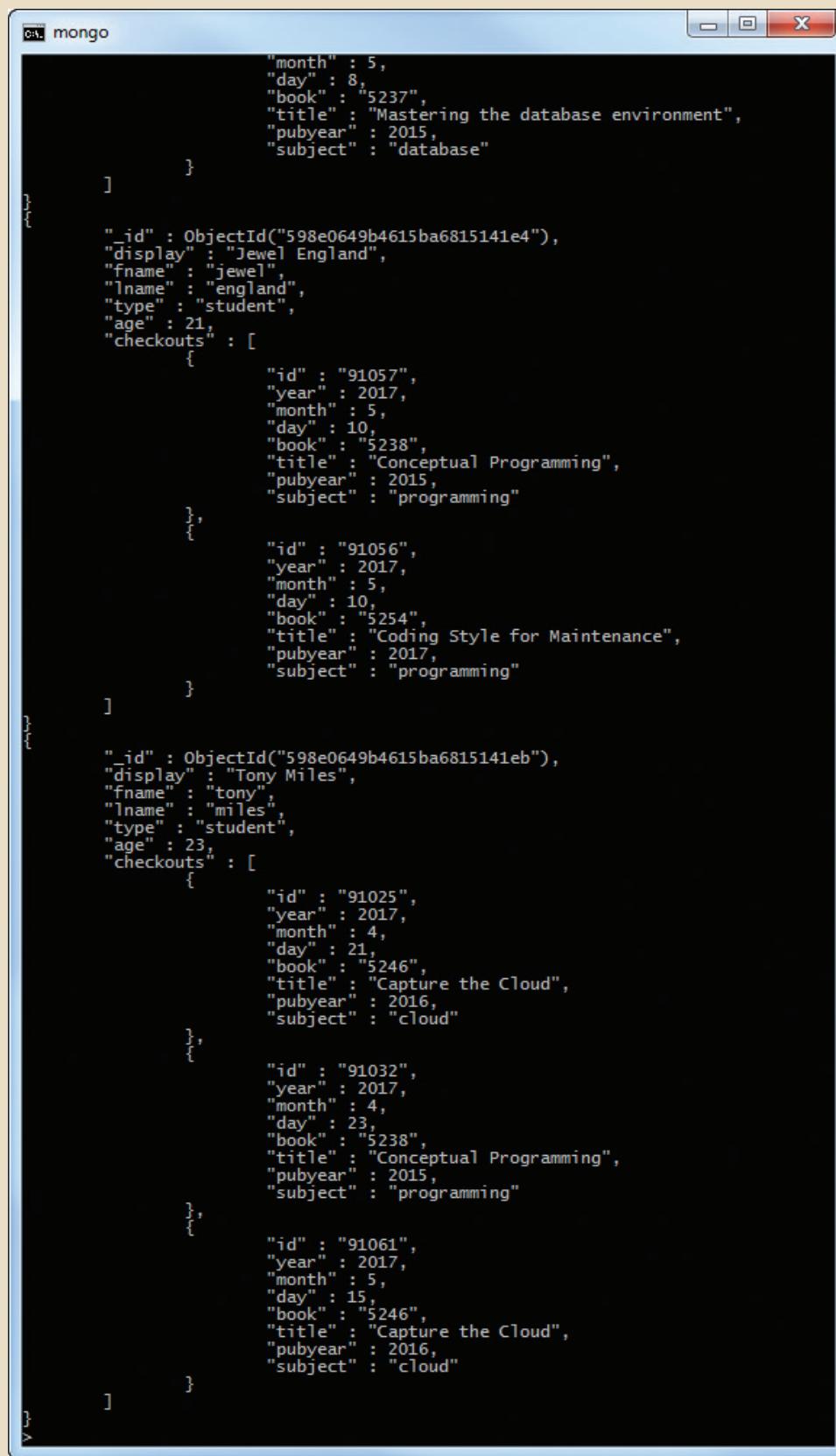
```
db.patron.find({checkouts: {$elemMatch: {subject: "programming", pubyear: 2015}}}).pretty()
```

Examining the output of this command shows that within the checkouts of each patron returned is at least one checkout event that satisfies both criteria.

\$elemMatch

A MongoDB operator used to specify that all criteria in an array of criteria must evaluate to true for the same element in a document array.

FIGURE P.32 USING \$ELEMMatch TO FIND CRITERIA IN THE SAME SUBDOCUMENT



The screenshot shows a mongo shell window with the title 'mongo'. The content of the window displays a list of documents from a collection. Each document represents a student record with their details and a 'checkouts' array. The 'checkouts' array contains multiple objects, each representing a book checkout with its details. The fields shown in the documents include: _id, display, fname, lname, type, age, and checkouts. The checkouts array contains objects with fields: id, year, month, day, book, title, pubyear, and subject.

```
        "month" : 5,
        "day" : 8,
        "book" : "5237",
        "title" : "Mastering the database environment",
        "pubyear" : 2015,
        "subject" : "database"
    }
]

{
    "_id" : ObjectId("598e0649b4615ba6815141e4"),
    "display" : "Jewel England",
    "fname" : "jewel",
    "lname" : "england",
    "type" : "student",
    "age" : 21,
    "checkouts" : [
        {
            "id" : "91057",
            "year" : 2017,
            "month" : 5,
            "day" : 10,
            "book" : "5238",
            "title" : "Conceptual Programming",
            "pubyear" : 2015,
            "subject" : "programming"
        },
        {
            "id" : "91056",
            "year" : 2017,
            "month" : 5,
            "day" : 10,
            "book" : "5254",
            "title" : "Coding Style for Maintenance",
            "pubyear" : 2017,
            "subject" : "programming"
        }
    ]
}

{
    "_id" : ObjectId("598e0649b4615ba6815141eb"),
    "display" : "Tony Miles",
    "fname" : "tony",
    "lname" : "miles",
    "type" : "student",
    "age" : 23,
    "checkouts" : [
        {
            "id" : "91025",
            "year" : 2017,
            "month" : 4,
            "day" : 21,
            "book" : "5246",
            "title" : "Capture the Cloud",
            "pubyear" : 2016,
            "subject" : "cloud"
        },
        {
            "id" : "91032",
            "year" : 2017,
            "month" : 4,
            "day" : 23,
            "book" : "5238",
            "title" : "Conceptual Programming",
            "pubyear" : 2015,
            "subject" : "programming"
        },
        {
            "id" : "91061",
            "year" : 2017,
            "month" : 5,
            "day" : 15,
            "book" : "5246",
            "title" : "Capture the Cloud",
            "pubyear" : 2016,
            "subject" : "cloud"
        }
    ]
}
```

MongoDB is a powerful document database that is being adopted by many organizations. It was originally designed to support web-based operations, and as such, it draws heavily on JavaScript for the structure of its documents and for MQL. The previous several sections have introduced you to the basic create, read, update, and delete operations that are central to database processing. This should give you a basic familiarity with MongoDB and MQL, but there is much more to learn if you are interested in pursuing a career in document databases.

Summary

- MongoDB is a document database that stores documents in JSON format. The documents can be created, updated, deleted, and queried using a JavaScript-like language, named MongoDB Query Language.
- Documents of a similar nature are stored together in a collection. Even though the documents are similar, they are not required to have exactly the same structures. A document database may contain many collections.
- Documents can be updated by either a replacement update or an operator update. A replacement update will replace an entire document with a new document that has the same unique id. An operator update can be used to modify specific key:value pairs in a document without impacting the remainder of the document.
- Data retrieval is done primarily through the `find()` method. Within the `find()` method, functions can be used to write complex logical criteria.
- Other methods can be chained to the `find()` method to support pagination, restricting results, and formatting output.

Key Terms

\$addToArray, P-18	\$inc, P-14	\$rename, P-21
\$and, P-32	insert(), P-8	renameCollection, P-8
capped collection, P-7	limit(), P-28	remove(), P-22
createCollection, P-7	method, P-6	replacement update, P-10
drop(), P-8	method chaining, P-28	\$set, P-12
\$each, P-19	\$or, P-33	skip(), P-29
\$elemMatch, P-36	operator update, P-10	sort(), P-29
explicit <i>and</i> , P-32	pretty(), P-26	subdocument, P-3
find(), P-9	\$pull, P-17	\$unset, P-13
getName(), P-6	\$pullAll, P-19	update(), P-10
implicit <i>and</i> , P-32	\$push, P-17	upsert, P-10

Review Questions

1. What is the difference between a replacement update and an operator update in MongoDB?
2. Explain what an upsert does.
3. Describe the difference between using \$push and \$addToSet in MongoDB.
4. Explain the functions used to enable pagination of results in MongoDB.
5. Explain the difference in processing when using an explicit *and* and an implicit *and* with MongoDB.

Problems

For the following set of problems, use the *fact* database and *patron* collection created in the text for use with MongoDB.

1. Create a new document in the patron collection. The document should satisfy the following requirements:

First name is “Rachel”

Last name is “Cunningham”

Display name is “Rachel Cunningham”

Patron type is student

Rachel is 24 years old

Rachel has never checked out a book

Be certain to use the same keys as already exist in the collection. Be certain capitalization is consistent with the documents already in the collection. Do not store any keys that do not have a value (in other words, no NULLs).

2. Modify the document entered in the previous question with the following data. Do not replace the current document.

Rachel has checked out two books on January 25, 2018.

The id of the first checkout is “95000”

The first book checked out was book number 5237

Book 5237 is titled “Mastering the database environment”

Book 5237 was published in 2015 and is in the “database” subject

The id of the second checkout is “95001”

The second book checked out was book number 5240

Book 5240 is titled “iOS Programming”

Book 5240 was published in 2015 and is in the “programming” subject

Use the same keys as already exist within the collection. Conform to the existing documents in terms of capitalization.

Online Content



The documents for the *fact* database is available as a collection of JSON documents that can be directly imported into MongoDB. The file is named Ch14_Fact.json and is available at www.cengagebrain.com.

3. Write a query to retrieve the _id, display name and age of students that have checked out a book in the cloud subject.
4. Write a query to retrieve only the first name, last name, and type of faculty patrons that have checked out at least one book with the subject “programming”.
5. Write a query to retrieve the documents of patrons that are faculty and checked out book 5235, or that are students under the age of 30 that have checked out book 5240. Display the documents in a readable format.
6. Write a query to display the only the first name, last name, and age of students that are between the ages of 22 and 26.

Appendix Q

Working with Neo4j

Preview

Even though Neo4j is not yet as widely adopted as MongoDB, it has been one of the fastest growing NoSQL databases, with thousands of adopters such as LinkedIn and Walmart. Neo4j is a graph database. Other types of NoSQL databases focus primarily on problem domains where volume and velocity are the core issues. In those situations, aggregate aware databases that can minimize interdependencies among the data can reap the benefits of scaling out to massive clusters of servers. Graph databases address a completely different type of problem. Like relational databases, graph databases still work with concepts similar to entities and relationships. However, in relational databases, the focus is primarily on the entities. In graph databases, the focus is on the relationships. This appendix shows you some step-by-step illustrations of using the Neo4j graph database.

Data Files and Available Formats

File name	Format/Description
Ch14_FCC.txt	Text file for Neo4j examples (also used in Chapter 14)

Data Files Available on cengagebrain.com



Note

Neo4j is a product of Neo4j, Inc. There are multiple versions of Neo4j available. In this appendix, we use the Community Server v.3.2.2 edition, which is open source and available free of charge from Neo4j, Inc. New versions are released regularly. This version of Neo4j is available from the Neo4j website for Windows (64-bit and 32-bit), MacOS, and Linux.

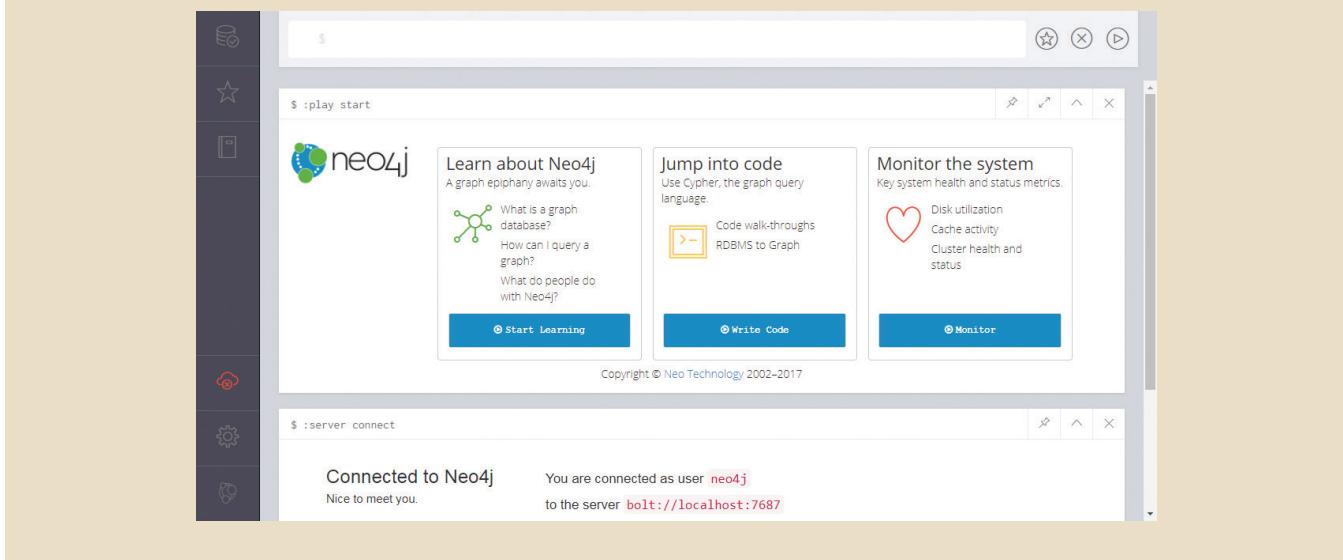
Q-1 Working with Graph Databases Using Neo4j

Graph databases are used in environments with complex relationships among entities. Graph databases, therefore, are heavily reliant on interdependence among their data, which is why they are the least able to scale out among the NoSQL database types. Consider an example of a social network such as LinkedIn that connects people together. A person can be friends with many other people, each of whom can be friends with many people. In terms of a relational model, we could represent this as a person entity with a many-to-many unary relationship. In implementation, we would create a bridge for the relationship and end up with a two-entity solution. Imagine the person table has 10,000 people (rows) in it, and those people average 30 friends each so that the bridge table has 300,000 rows. A query to retrieve a person and the names of his or her friends would require two joins: one to link the person to their friends in the bridge and another to retrieve those friends' names from the person entity. A relational database can perform this query quickly. The problem comes when we look beyond that direct friend relationship. What if we want to know about friends of friends? Then another join connecting the bridge table to itself will have to be included. Joining a 300,000-row table to itself is not trivial (there are 90 billion rows in the Cartesian product that the DBMS engine is contending with to construct the join). The relational database can handle that volume, but it is starting to slow. Now query for friends of friends of friends. This requires joining yet another copy of the bridge table so the query, producing a Cartesian product with 2.7×10^{16} rows! As you can see, by the time we are working the “six degrees of separation” types of problems, relational database technology is unable to keep up. These types of highly interdependent queries about relationships that could take hours to run in a relational database are the forte of graph databases. Graph databases can complete these queries in seconds. In fact, you often encounter the phrase “minutes to milliseconds” when adopters describe their use of graph databases.

Q-2 The Neo4j Interface

Neo4j provides multiple interface options. It was originally designed with Java programming in mind, and optimized for interaction through a Java API. Later releases have included the options for a Neo4j command shell, similar to the MongoDB shell, a REST API for website interaction, and a graphical, browser-based interface for intuitive interactive sessions. Figure Q.1 shows the browser-based interface.

FIGURE Q.1 GRAPHICAL, BROWSER-BASED NEO4J INTERFACE



We focus on the browser-based interface throughout this appendix. The browser interface has evolved over different versions and is likely to evolve further. In the current version of the web interface, the left side of the screen is a menu with options for viewing database information, documentation, and changing settings. The main section of the screen is a stream of database activity for the current session. When you first log in, this section shows links to take you to various activities. Notice that the stream is composed of several individual blocks, or frames. Each block can be closed by clicking the “X” in the upper-right corner of that block, expanded to fill the screen, and other options. The results of query execution appear in the stream. As a session goes on, the stream can become quite long and filled with the results of all previous queries. Eventually, this can begin to slow your browser performance, so clearing unneeded results by entering the `:clear` command at the editor prompt is recommended.

At the top of the main section is an editor bar with a “\$” prompt. This is the editor prompt for entering new commands. At the right end of the editor bar are options to “play” (right triangle) or execute the query that has been entered at the prompt, “clear” (x) to clear the editor bar, and “update favorite” (star) to save the query to the list of favorite queries that is accessible in the left-side menu. The editor is in single-line mode by default. When in single-line mode, pressing Enter executes the command. This can be convenient because many Neo4j commands can be rather short. If a command is too long or complex for single-line mode, the editor can be placed in multiline mode by pressing Shift+Enter. For very large commands, full-screen mode for the editor can be toggled on and off by pressing Esc. When in multiline or full-screen mode, commands can be executed by pressing Ctrl+Enter (Cmd+Enter for Mac). In all cases, the command in the editor bar can be executed by clicking the “play” button at the right end of the bar.

Q-3 Creating Nodes in Neo4j



Note

An instance of Neo4j can have only one active database at a time. However, the data path for the database can be changed in the configuration before starting the Neo4j server. If the data path is changed to point at an empty directory, Neo4j automatically creates all needed files in that directory on start-up. By keeping each database in a separate folder and changing the data path before starting the server, multiple databases can be maintained for practice.

Graph databases are composed of nodes and edges. Roughly speaking, nodes in a graph database correspond to entity instances in a relational database. In Neo4j, a label is the closest thing to the concept of a table from the relational model. A label is a tag that is used to associate a collection of nodes as being of the same type or belonging to the same group. Just as entity instances have values for attributes to describe the characteristics of that instance, a node has properties that describe the characteristics of that node. Unlike the relational model, graph databases are schema-less so nodes with the same label are not required to have the same set of properties. In fact, nodes can have more than one label if they logically belong to more than one group.

Consider an example of a club for food critics where members share reviews of area restaurants. Each club member would be represented as a node. Each restaurant would be represented as a node. Although both members and restaurants are nodes, the members are one kind or type of node while the restaurants are another kind or type of node. To help distinguish the types of nodes both in code and in the minds of users and programmers, you can use labels. The nodes for members might get a Member label, and nodes for restaurants get a Restaurant label. This makes it more convenient in code to distinguish between the types of nodes.

The interactive, declarative query language in Neo4j is called **Cypher**. We will be using Cypher to manipulate and traverse (query) the graph. Cypher is declarative, like SQL, even though the syntax is very different. However, being a declarative language instead of an imperative language, like MQL, Cypher is very easy to learn and a few simple commands can be used to perform basic database processing.

Nodes and relationships are created using a CREATE command. The following code creates a member node, as shown in Figure Q.2:

```
CREATE (:Member {mid: 1, fname: "Phillip", lname: "Stallings"})
```

Cypher

A declarative query language used in Neo4j for querying a graph database.

FIGURE Q.2 CREATING A NODE IN NEO4J

The screenshot shows the Neo4j browser interface. On the left, there is a sidebar with two tabs: 'Table' (selected) and 'Code'. The main area displays the command '\$ CREATE (:Member {mid: 1, fname: "Phillip", lname: "Stallings"})' and its output: 'Added 1 label, created 1 node, set 3 properties, completed after 16 ms.' Below this, another identical command and output are shown.



Note

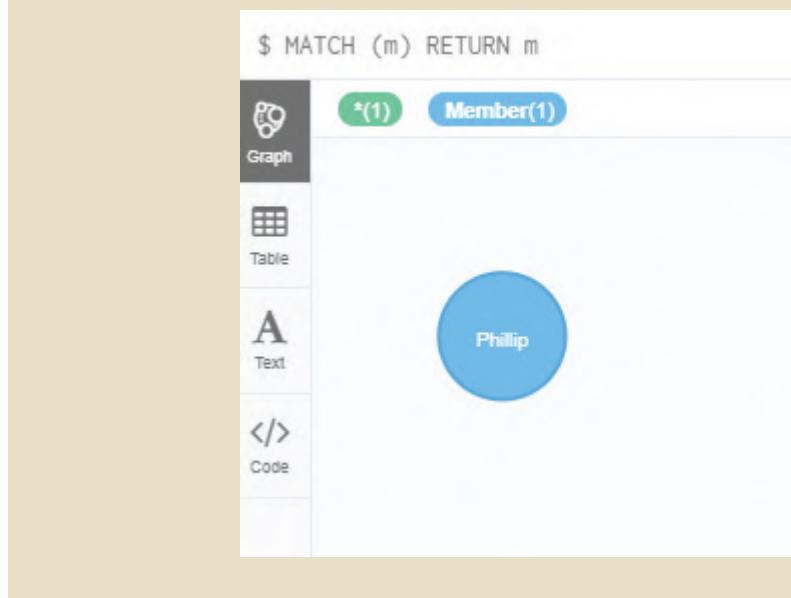
Neo4j creates an internal ID field named <id> for every node and relationship; however, this field is for internal use within the database for storage algorithms. It is not intended to be, and should not be used as, a unique key.

The previous command creates a node with the Member label. That node was given the properties *mid* with the value 1, *fname* with the value “Phillip”, and the property *lname* with the value “Stallings”. The *mid* property is being used as a member ID field to identify the members. If there is not already a label named Member, it is created at the same time the node is, as shown in the output.

Retrieving nodes and relationships will be covered in detail a little later, but for now, the following command can be used to retrieve all nodes from the database, as shown in Figure Q.3:

```
MATCH (m)  
RETURN (m)
```

FIGURE Q.3 VIEWING THE NEW MEMBER NODE



MATCH is the Cypher command to find data in the database. In this case, “m” is being used as a variable. Just as with most variables, we can name the variable anything we want, but we try to be descriptive. The current database contains only a single Member node so “m” was chosen to represent Member. Without any qualifiers to restrict the results, the MATCH command matches every node by default. In the previous command, then, all nodes are found and assigned to the variable *m*. **RETURN** is the Cypher command to return the values in a variable. In this case, we are returning every value in the variable *m*, which contains every node in the database due to the unrestricted MATCH command.

Executing this command retrieves the node we created and displays it in a frame in the stream of the main window. Options for viewing the data being returned are located on the left side of the frame. If the data contains nodes, as with the previous command, there is an option to view the result as a graph. Alternatively, you can choose to view it in a tabular format that displays the result in a JSON format, in a text format that mirrors the output you would see in the Neo4j shell, or the coded format that would be returned if the command had been issued through one of the programming APIs. In the graph view of the data, the layout can be manipulated by dragging nodes, and the display options such as node size, color, and caption can be customized.

The following command can be used to create a restaurant node with the label Restaurant, and properties for restaurant id (rid), name, and city:

```
CREATE (:Restaurant {rid: 1, name: "Joe's Eatery", city: "Browningville"})
```

Q-4 Updating Nodes in Neo4j

Like most processing in Cypher, updating a node in Neo4j involves the use of the MATCH command. Before a change can be made to a node, the node must be found in the database, which is the purpose of MATCH. Once MATCH locates the node, the SET command can be used to update or add properties to the node. For example, if we want

match

A keyword in the Cypher language used by Neo4j for pattern matching.

return

A keyword in a Cypher language used by Neo4j to specify the nodes and properties returned by a traversal

to change the city of our restaurant node from “Browningville” to “Brownville” and add property named state with the value “OH”, the following command could be used:

```
MATCH(r {rid: 1})  
SET r.city = "Brownville", r.state = "OH"
```

Neo4j responds that the properties have been set as shown in Figure Q.4. The MATCH command finds all nodes that have rid = 1 and assigns them all to the variable *r* (there was only one such node in our database). The SET command changes the values in properties. In the previous command, SET changes the city property of the node contained in variable *r* to have the value “Brownville”, and it changes the state property of the node contained in variable *r* to have the value “OH”. If the property does not exist, SET will add that property to the node. You can retrieve all of the nodes in the database again as shown previously. By hovering the cursor over the Joe’s Eatery node, you can see the changes to the properties, or you can click on the table or text views on the left of the frame to see the properties, as shown in Figure Q.5.

FIGURE Q.4 UPDATING A RESTAURANT NODE

The screenshot shows the Neo4j browser interface. At the top, there is a command line input field containing the query: \$ MATCH(r {rid: 1}) SET r.city = "Brownville", r.state = "OH". Below the command line, there is a status message: "Set 2 properties, completed in less than 1 ms." To the left of the main content area, there is a sidebar with two tabs: "Table" and "Code". The "Table" tab is currently selected, showing a single row of data with columns for "Property" and "Value". The "Code" tab is also present in the sidebar.

FIGURE Q.5 VIEWING THE RESTAURANT NODE PROPERTIES

The screenshot shows the Neo4j browser interface. On the left, there's a sidebar with icons for Graph, Table (selected), Text, and Code. The main area displays a query result for a node 'm'. The node has two properties: 'r' and 'm'. The 'r' property contains a record with 'city', 'name', 'state', and 'rid' fields. The 'm' property contains a record with 'fname', 'lname', and 'mid' fields. At the bottom, a message says 'Started streaming 2 records after 1 ms and completed after 1 ms.'

```
$ MATCH (m) RETURN m
```

m
{ "r": { "city": "Brownville", "name": "Joe's Eatery", "state": "OH", "rid": 1 }, "m": { "fname": "Phillip", "lname": "Stallings", "mid": 1 } }

Started streaming 2 records after 1 ms and completed after 1 ms.

Removing a property is done in a similar fashion, except with the REMOVE command instead of the SET command. If we wish to remove the city property from Joe's Eatery, the following command could be used (see Figure Q.6):

```
MATCH (r {rid: 1})  
REMOVE r.city
```

FIGURE Q.6 RESTAURANT WITH CITY PROPERTY REMOVED

```
$ MATCH (m) RETURN m
```

m
<pre>{ "fname": "Phillip", "lname": "Stallings", "mid": 1 }</pre>
<pre>{ "name": "Joe's Eatery", "state": "OH", "rid": 1 }</pre>

Started streaming 2 records in less than 1 ms and completed in less than 1 ms.

Q-5 Retrieving Nodes Using MATCH

As we have already seen, MATCH is used to find nodes. More accurately, MATCH is used for pattern matching. Retrievals in Neo4j are all about pattern matching. Recall the graph view of the nodes that were created above. The nodes are represented as circles. MATCH uses the symbols on the keyboard to “draw” patterns. Nodes are inside parentheses () because they look like a circle. Properties are listed inside the parentheses because they are the properties of the node. MATCH places the nodes that match the pattern specified in the variable specified. Multiple patterns can be specified with MATCH. For example, we could use the following command to find both our Member node and Restaurant node, with the result shown in Figure Q.7:

```
MATCH (m {mid: 1}), (r {rid: 1})
RETURN m, r
```

FIGURE Q.7 RETRIEVING NODES WITH TWO PATTERNS



The first pattern finds all nodes with the property mid with the value 1 and assigns them to the variable *m*. The second pattern finds all nodes with the property rid with the value 1 and assigns them to the variable *r*. RETURN then displays all the nodes in both variables. Combining patterns with a comma causes a logical *and* between the patterns. In this case, the query returned both nodes because the set of nodes in the two patterns did not have to overlap. Consider the following command, which will not return any nodes:

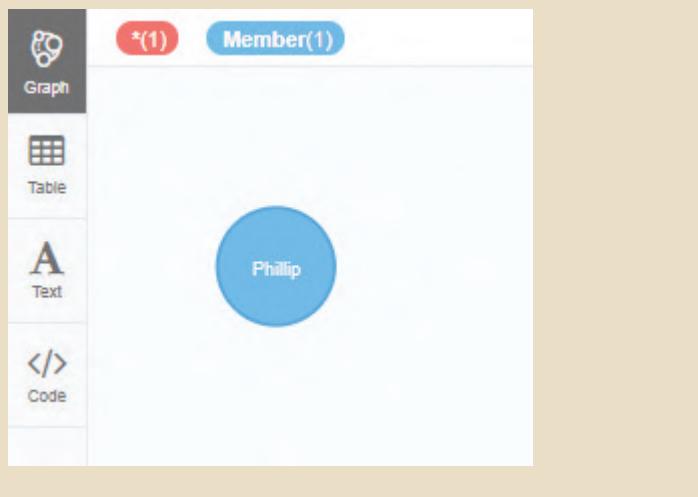
```
MATCH (x {mid: 1}), (x {rid: 1})
RETURN x
```

The previous command returns no nodes because the same variable *x* was used in both patterns. This means that to be returned, a node would have to match both patterns. On the other hand, the following command will return a node:

```
MATCH (x {fname: "Phillip"}), (x {lname: "Stallings"})
RETURN x
```

Figure Q.8 shows that this command will return a node because our Member node matches both patterns. Notice again that the name of the variable does not matter. In this case, the variable *x* is not very descriptive. The important thing to remember is that whatever the variable name is, it must be used consistently throughout the command whenever you want to refer to the same set.

FIGURE Q.8 MATCHING TWO PATTERNS WITH ONE NODE



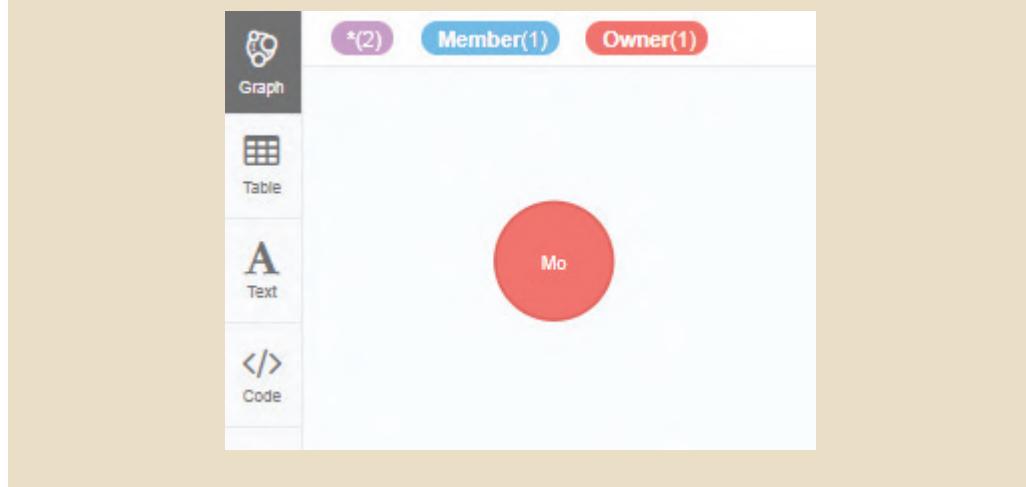
In addition to matching patterns based on properties, it is also possible to use labels in the pattern. The following command will return our only Member node:

```
MATCH (phil :Member)
RETURN phil
```

Notice that the variable name used is “phil”. You will see in practice that when programmers know that the pattern is designed to match only one node, the variable name often reflects the name of the node. This becomes more useful in improving the readability of the query when the retrievals become more complex, as we will see later. In the previous command, the label Member is used to limit the pattern to only nodes with that label. As mentioned earlier, it is possible for a node to have more than one label. For example, a club member may also be the owner of a restaurant. It may be valuable to use an Owner label to simplify working with the group of nodes that represent restaurant owners. The following command creates a new node that is labeled as both a member and an owner, and returns that node all in one command, as shown in Figure Q.9:

```
CREATE (new :Member:Owner {mid: 2, fname: "Mo", lname: "Saleem"})
RETURN new
```

FIGURE Q.9 CREATING A NODE WITH TWO LABELS



The newly created node has both the Member label and the Owner label. The following command will return all nodes with the Member label:

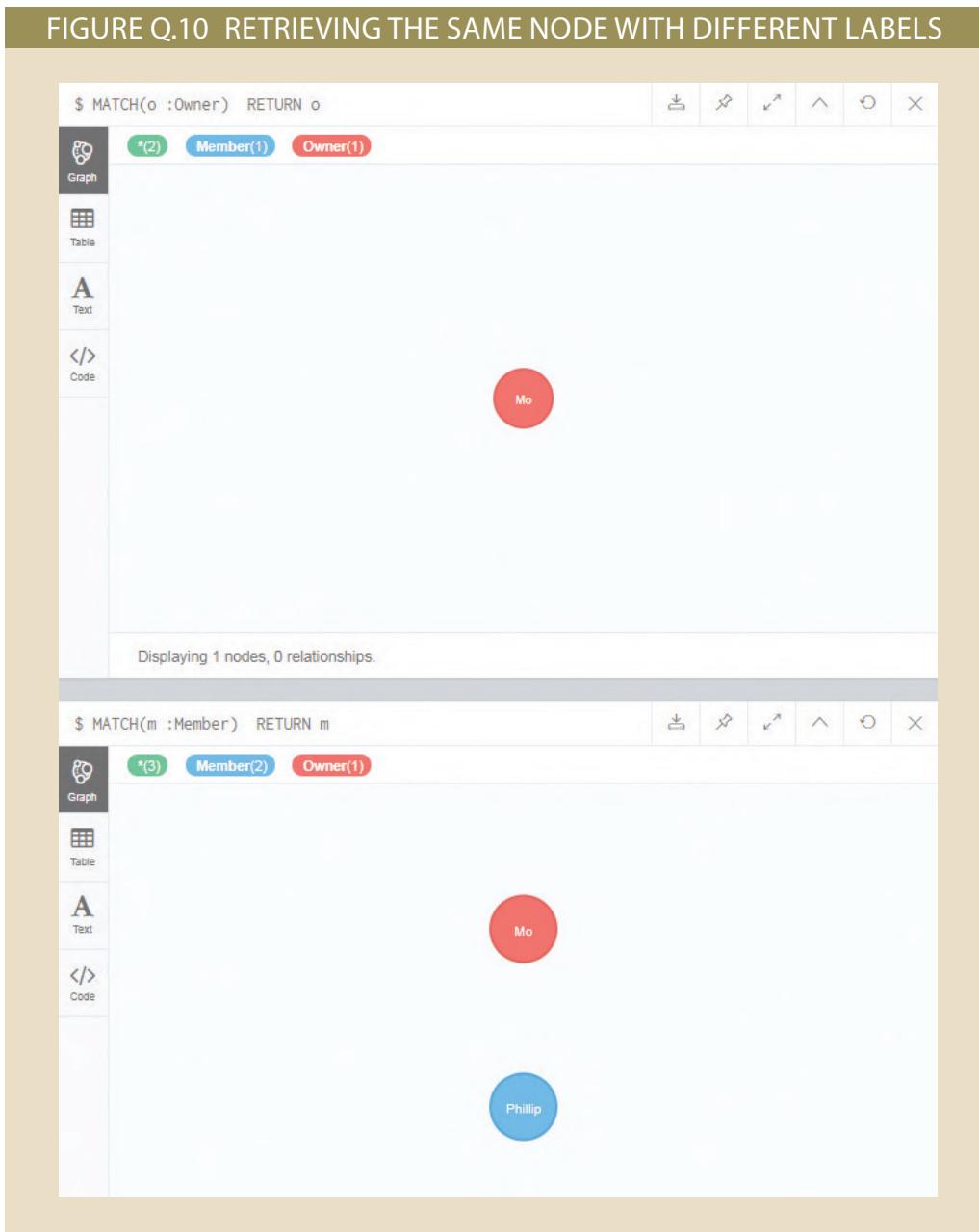
```
MATCH(m :Member)  
RETURN m
```

The next command will return all nodes with the Owner label:

```
MATCH(o :Owner)  
RETURN o
```

The results of both commands are shown in Figure Q.10. Notice that the Mo Saleem node was returned both times.

FIGURE Q.10 RETRIEVING THE SAME NODE WITH DIFFERENT LABELS



Q-5a RETURN Nodes and Properties

The RETURN keyword in Cypher is used to specify the data that is to be returned by the query by listing the variables that contain the data to be returned. As you have already seen, listing a variable that contains a node will return the entire node. We can view the node graphically with the graph view of the Neo4j interface, or we can view tabular or text versions of the data. However, it is not required to retrieve the entire node. Just as when using the SET command to update a property of a node, dot notation can be used to specify a property of a node to be returned. For example, the following command returns only the first and last name properties of the Phillip Stallings member node, as shown in Figure Q.11:

```
MATCH (phil :Member {mid: 1})
RETURN phil.fname, phil.lname
```

FIGURE Q.11 RETRIEVING SPECIFIED PROPERTIES

"phil.fname"	"phil.lname"
"Phillip"	"Stallings"

Notice that the graph view is not available in the interface because the entire node was not returned.

When returning individual properties, it is possible to provide aliases as is done in SQL with column aliases. In Cypher, the AS keyword is required before the alias. If an alias contains a space or a reserved keyword, then it must be enclosed in backticks, as shown in Figure Q.12 for the following code:

```
MATCH (mem :Member {mid: 2})
RETURN mem.fname AS 'First Name', mem.lname AS 'Last Name'
```

FIGURE Q.12 PROPERTIES WITH ALIASES

"First Name"	"Last Name"
"Mo"	"Saleem"

Note

The following sections on Neo4j assume that you have loaded the data from the Ch14_FCC.txt file that is available online and used with Chapter 14, Big Data and NoSQL. This file contains a single, massive command that will create 78 additional members, 43 owners, 67 restaurants, and 8 cuisines. Providing the code as a single command is necessary if you are using the browser interface. Because it is designed for interactive use, it does not support script files with multiple commands. The command includes many statements that may seem unfamiliar to you, but by the time you finish this chapter, all of the commands should make sense.

Q-5b Retrieving Nodes with WHERE

The data model used for the following sections is from the Food Critics Club (FCC). The database tracks data on members, restaurants, and owners like the ones created in commands in the previous sections. It also records data on the different cuisines, or styles of food. Members may have left reviews of one or more restaurants. With each review, the member can rate the restaurant on a scale of 1 to 5 for each of the following categories: taste, service, atmosphere, and value. Each member is assigned a unique member id (mid). This is recorded along with the member's first and last name, email address, and a FCC user name. Optionally, the member's year of birth and state of residency may also be recorded. Restaurants have a restaurant id number (rid) and name. Typically, the restaurant's address and phone number are also recorded. Most restaurants have a price rating on a scale of 1 to 5, where 1 is the cheapest and 5 is the most expensive.

Thus far, we have specified criteria for the nodes to retrieve either by label or by property inside the node pattern. The major limitation with using properties inside the node for retrievals is that only equalities can be included. Further, multiple properties and values are always treated with a logical *and*. This is similar to the way MongoDB interpreted a key:value pair in a criterion as an equality. Neo4j and Cypher do the same thing.

Alternatively, we can add a WHERE clause following the MATCH pattern. The WHERE clause allows both *and* and *or* logical connectors, and it allows criteria to include inequalities. Unlike MongoDB, Neo4j uses mathematical operators for inequalities instead of functions. Therefore, the WHERE clause in Cypher is similar to the WHERE clause in SQL. The following command retrieves all members that were born after 1987.

```
MATCH (m :Member)
WHERE m.birth > 1987
RETURN m
```

Using the WHERE clause allows both logical *and* and *or* operations. As with SQL, when combining *and* and *or* in the same WHERE clause, it is recommended to use parentheses to ensure the correct order of operation for the logical connectors. The following command finds all members that live in the states of TN or KY and were born before 1984:

```
MATCH (m :Member)
WHERE (m.state = "TN" or m.state = "KY") and m.birth < 1984
RETURN m
```

Q-5c Common String Functions in Neo4j

The Cypher language in Neo4j provides many of the same character string functions available in SQL that were discussed in Chapter 7, Introduction to Structured Query Language (SQL). Table Q.1 lists some of the most common string functions in Cypher. By default, queries in Neo4j are case sensitive. For example, it is common practice to start labels with a capital letter, thus we have :Member, :Restaurant, :Owner, and :Cuisine labels. If we issue a query that includes :member or :MEMBER as a label, it will not match any nodes because the label is case sensitive. Similarly, if we had queried for the STATE or State property of a member, no nodes would be found because the property has been entered in all lowercase letters in our database.



The Ch14_FCC.txt file used in the following sections is available at www.cengagebrain.com. The contents of the file should be copied and pasted into the Neo4j editor bar and executed using the play button in the interface.

TABLE Q.1

COMMON CYpher STRING FUNCTIONS

FUNCTION	DESCRIPTION
lower(<text>)	Converts text to lowercase letters
left(<text>, <number>)	Retrieves the specified number of letters from the left side of the text
right(<text>, <number>)	Retrieves the specified number of letters from the right side of the text
substring(<text>, <start>, <number>)	Retrieves the specified number of letters from the string starting with the specified character position
trim(<text>)	Removes all whitespace from both sides of the text
upper(<text>)	Converts text to uppercase letter

As the database programmer, it is your responsibility to ensure consistency in how labels and properties are capitalized to avoid problems later. Your organization probably has defined standards for these issues. However, we cannot always control the way that users capitalize data values that are being entered into the system. Therefore, string functions like upper() and lower() are available. The following command retrieves all restaurants located in a city named Brentwood, regardless of how the value is capitalized in the database. The results are shown in Figure Q.13.

```

MATCH (res :Restaurant)
WHERE lower(res.city) = "brentwood"
RETURN res

```

FIGURE Q.13 RESTAURANTS IN BRENTWOOD

The screenshot shows the Neo4j browser interface with a sidebar on the left containing icons for Graph, Table (selected), Text, and Code. The main area displays two nodes under the label 'res'. Each node has a JSON-like representation of its properties.

```

res[{"id": 1, "name": "El Toros", "city": "Brentwood", "state": "GA", "street": "1412 Frolick Wind Road", "zip": "3833", "price": 3, "phone": "(404) 555-1616", "rid": 4639}, {"id": 2, "name": "The Fridge", "city": "Brentwood", "state": "TN", "street": "1412 Frolick Wind Road", "zip": "3833", "price": 1, "phone": "(615) 555-1606", "rid": 4651}]

```

Instead of the LIKE operator with wildcards that are used in SQL, Cypher provides three separate operators with similar capabilities for substring searches. Recall that a substring search looks for a smaller piece of text inside a larger piece of text. The *starts with*, *contains*, and *ends with* keywords can be used to create criteria that start with, contain, or end with a specified character value. The following command uses the *starts with* operator to find the restaurant names that start with the word “Cheese”:

```

MATCH (res :Restaurant)
WHERE upper(res.name) STARTS WITH "CHEESE"
RETURN res.name

```

The next command finds restaurant names that contain the word “Cheese” anywhere within the name. The results of this command and the previous command can both be seen in Figure Q.14.

```
MATCH (res :Restaurant)
WHERE upper(res.name) CONTAINS "CHEESE"
RETURN res.name
```

FIGURE Q.14 SUBSTRING SEARCHES IN NEO4J

The screenshot shows the Neo4j browser interface with two query results. The top result is for a CONTAINS search and the bottom is for a STARTS WITH search.

Top Result (CONTAINS):

```
$ MATCH (res :Restaurant) WHERE upper(res.name) CONTAINS "CHEESE" RE...
res.name
"Cheese Please"
"Chubby Charlies Cheeseburgers"
```

Started streaming 2 records in less than 1 ms and completed after 4 ms.

Bottom Result (STARTS WITH):

```
$ MATCH (res :Restaurant) WHERE upper(res.name) STARTS WITH "CHEESE" ...
res.name
"Cheese Please"
```

Started streaming 1 records in less than 1 ms and completed in less than 1 ms.

Interestingly, many of the string functions used in Cypher can also be used in the RETURN command to change the way that properties are displayed. For example, the substring function can be used in both the WHERE and RETURN clauses, as shown in the following command:

```
MATCH (res :Restaurant)
WHERE substring(res.phone, 1, 3) = "615"
RETURN res.name, substring(res.phone, 5) as phone
```

The substring function returns a part of a text value. Notice that the substring function accepts three parameter values. The first parameter specifies the text field to take the partial value from, the second parameter is which character position to start with, and the third parameter is how many letters of text to return. If the third parameter is omitted,

substring will return the rest of the string from the starting position. For the second parameter, the starting position, counting the characters begins with 0. So in the phone number (615) 555-1212, the left parenthesis is in position 0 and the area code begins in position 1. It can seem a little confusing at first because the first character is in position 0, the second character is in position 1, the third character is in position 2, and so, but with a little practice, you will get accustomed to it.

In the WHERE clause of the previous command, the substring looks in the restaurant phone number property, then, starting with the character in position 1, retrieves three characters (in this case, the second, third, and fourth characters from the phone property). In the RETURN clause, substring uses the restaurant's phone number again, and then returns all of the characters starting with the sixth character, as shown in Figure Q.15.

FIGURE Q.15 USING SUBSTRINGS IN WHERE AND RETURN

The screenshot shows a Neo4j browser window. On the left, there is a sidebar with three tabs: 'Table' (selected), 'Text', and 'Code'. The main area displays a table with two columns. The first column contains the restaurant names, and the second column contains their corresponding phone numbers. The data is as follows:

"res.name"	"phone"
"Beijing Table"	" 555-1571"
"Shanghai Home"	" 555-6521"
"Authentica Rustica"	" 555-6113"
"Free Range Salads"	" 555-2276"
"Elvenskirts"	" 555-2512"
"Thick Cuts Steakhouse"	" 555-2933"
"The Sicilian"	" 555-2751"
"NOLOs"	" 555-1708"
"La Grande"	" 555-8889"
"Racy Burgers"	" 555-1137"
"Ravmonds"	" 555-2481"

Q-5d Retrieving Data on Relationships in Neo4j

At the heart of the need for graph databases is the ability to work with highly interconnected data. Therefore, data retrievals involving relationships are the most common type of queries. Fortunately, retrieving data on relationships is very similar to retrieving data on nodes, so the skills you have been developing in the previous sections will work very well with more complex queries too.

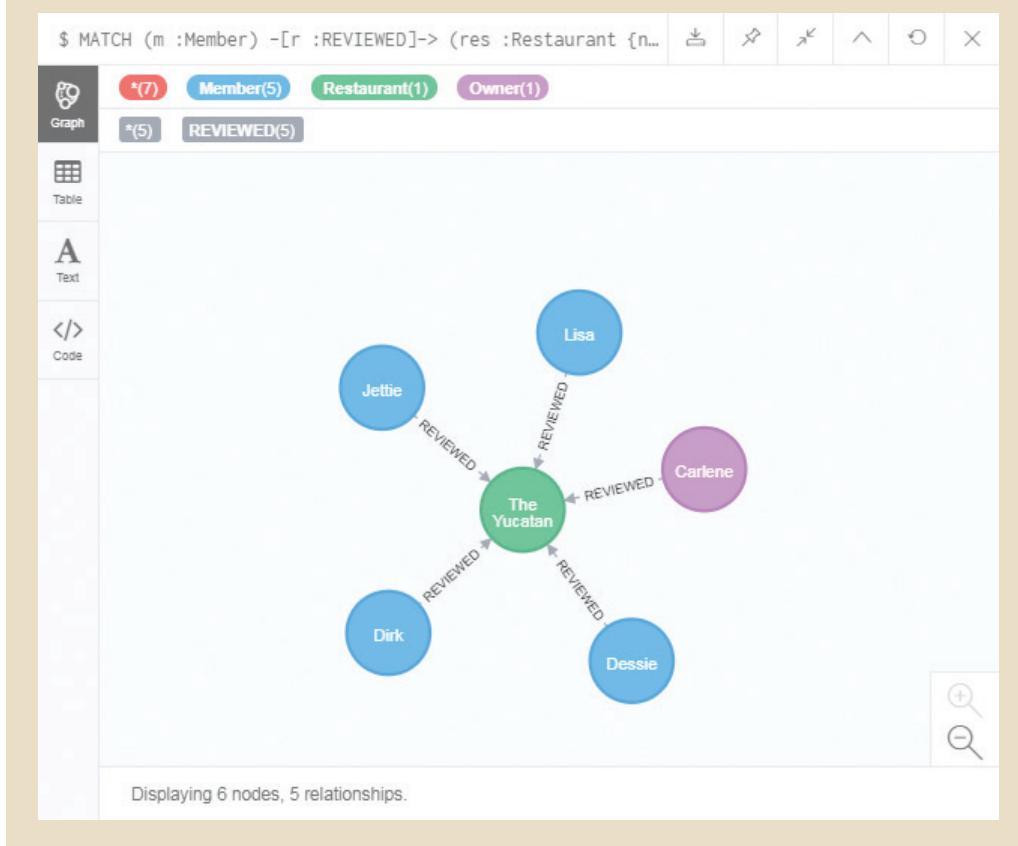
Retrieving Data on Immediate Relationships Recall that the MATCH command is used for pattern matching. In the patterns, nodes are represented with parentheses and relationships are represented with arrows. A relationship can be in only one direction, or it can be bidirectional. A relationship in one direction is specified in a pattern as --> or as <-- depending on the direction.

Relationships can have labels, just like nodes can have labels. Relationships can have properties, just like nodes. In a Cypher query, a relationship can be assigned to a variable so that data about it can be returned, just like a node. The techniques for specifying labels and properties and assigning to variables are the same for relationships as nodes. In the FCC database, there is a directional “review” relationship between members and restaurants. The relationship is directional because members review restaurants, restaurants do not review members. The pattern would be (Member) --> (Restaurant). Just as data about a node can be placed inside the node parentheses, data about the relationship can be placed inside the arrow. Data inside a relationship pattern is placed inside square brackets. The following command retrieves data that matches the pattern of a member reviewing a restaurant named The Yucatan.

```
MATCH (m :Member) -[r :REVIEWED]-> (res :Restaurant {name: "The Yucatan"})  
RETURN m, r, res
```

If you view the result in graph view, as shown in Figure Q.16, you will see all of the members who have reviewed the specified restaurant. Changing to the table view, you can see the properties of the nodes as well as the properties of the relationships.

FIGURE Q.16 RETRIEVING A RELATIONSHIP PATTERN



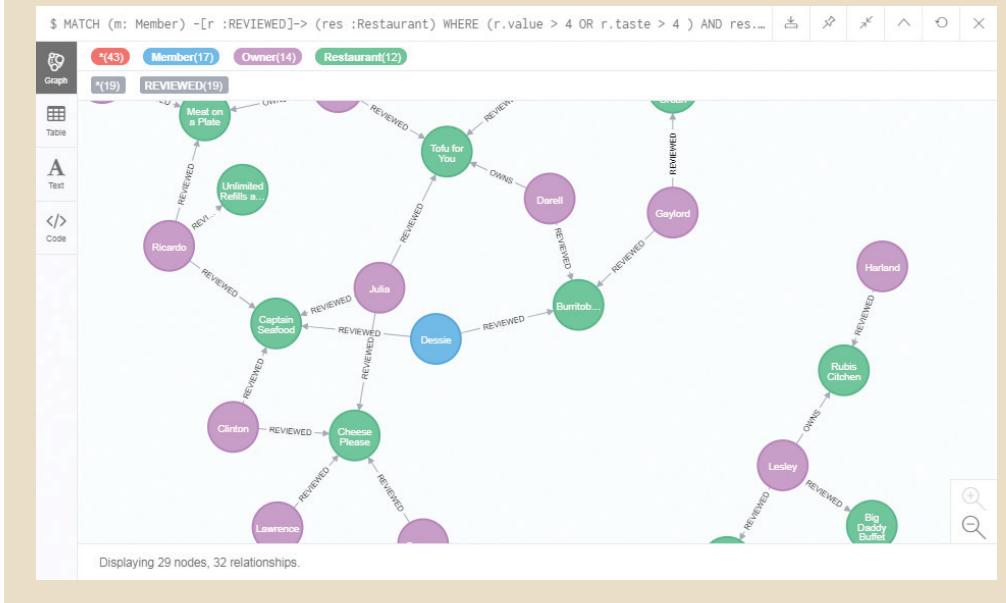
Relationship properties can be included in the pattern of the query or in a WHERE clause. The following command retrieves every member who has reviewed the restaurant “Tofu for You” and rated the restaurant a “4” on taste.

```
MATCH (m :Member) -[r :REVIEWED {taste: 4}]-> (res :Restaurant {name: "Tofu for You"})
RETURN m, r, res
```

Again, to use inequalities or logical *or* connectors, the WHERE command can be used, as shown in the following command, with the results shown in Figure Q.17:

```
MATCH (m :Member) -[r :REVIEWED]-> (res :Restaurant)
WHERE (r.value > 4 OR r.taste > 4 ) AND res.state = "KY"
RETURN m, r, res
```

FIGURE Q.17 RETRIEVING A RELATIONSHIP PATTERN USING WHERE



The order in which the pattern is written does not affect the result. For example, the following command produces exactly the same result as the previous command:

```
MATCH (res :Restaurant) <-[r :REVIEWED]- (m :Member)
WHERE (r.value > 4 OR r.taste > 4) AND res.state = "KY"
RETURN m, r, res
```

When the direction of the relationship is not important, the relationship pattern omits the arrowhead, which matches on a relationship in either direction. Using a label for the relationship is optional as well. For example, if we want to see everyone associated with The Sicilian restaurant, the following command can be used:

```
MATCH (m :Member) --(res :Restaurant {name: "The Sicilian"})
RETURN m, res
```

Notice that because a variable was not assigned to the relationship, the relationship could not be included in the RETURN. Therefore, even though the relationships show in the graph view, looking at the table view shows that none of the relationship data is returned.

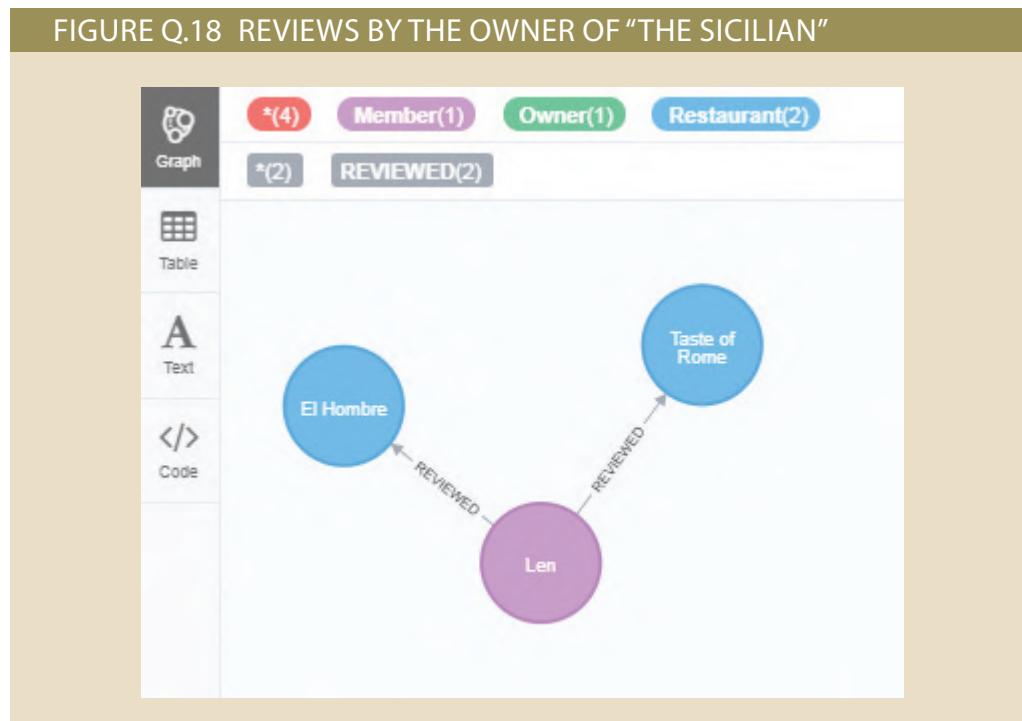
Thus far, we have limited our queries to a single relationship; however, a query in Neo4j can involve multiple relationships using more than two nodes. A simple example would be to find the name of every owner of an Italian restaurant, as shown here:

```
MATCH (c :Cuisine {name: "Italian"}) <-- (res :Restaurant) <-[r :OWNS]- (m :Member)
RETURN m.fname, m.lname
```

Recall from the previous discussion of using multiple patterns with the MATCH command, that if the same variable is used in multiple nodes, Cypher interprets this as requiring the same node in both patterns. The same is true when there are multiple references to a node in a relationship pattern. To include patterns that include multiple instances of a node label, but allow the nodes to be different, be certain to assign the nodes to

different variables. For example, the following command can retrieve all reviews completed by the owner of “The Sicilian” restaurant by using different variables for the two restaurant nodes. The results are shown in Figure Q.18.

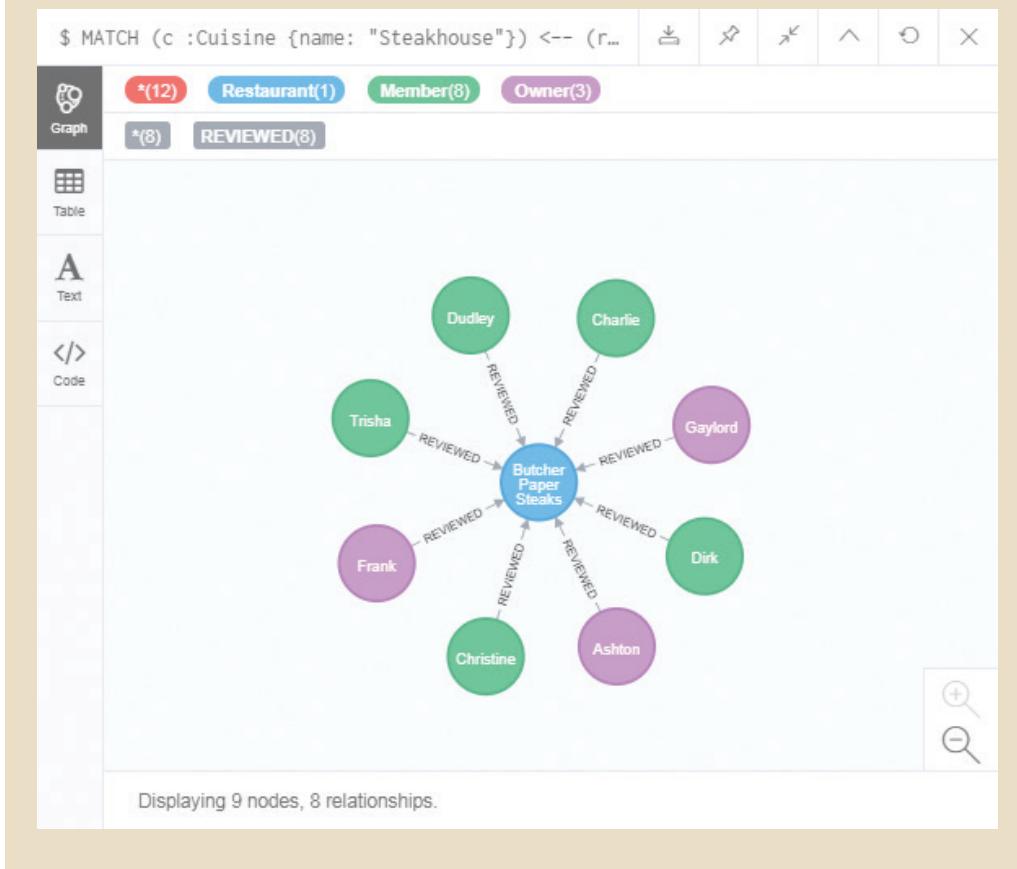
```
MATCH (sicilian :Restaurant {name: "The Sicilian"}) <-[{:OWNS}]- (m :Member) -[r :REVIEWED]-> (res :Restaurant)
RETURN m, r, res
```



Using multiple patterns is also possible with relationships. In fact, if a query pattern requires more than two relationships connecting to the same node, then multiple patterns will be required. This requirement is logical if you consider that when written as a command, one relationship can be written on the left side of a node and another relationship can be written on the right side of a node, but there is no way to write a relationship above or below a node in command format. Therefore, multiple patterns are used. For example, imagine that member “Dirk Gray” is thought to be influential in the Food Critics Club, so we want to see everyone that has reviewed a steakhouse restaurant that was also reviewed by Dirk Gray. This will require a restaurant to have three relationships. It must have been reviewed by Dirk. The cuisine must be steakhouse. It must be reviewed by another member. The following command uses multiple patterns to provide the requested data, with the results shown in Figure Q.19.

```
MATCH (c :Cuisine {name: "Steakhouse"}) <-- (res :Restaurant) <- [r :REVIEWED]-
(m :Member),
(res) <- [dr :REVIEWED] - (dirk :Member {fname: "Dirk", lname: "Gray"})
RETURN res, r, m, dirk, dr
```

FIGURE Q.19 MATCHING MULTIPLE RELATIONSHIP PATTERNS

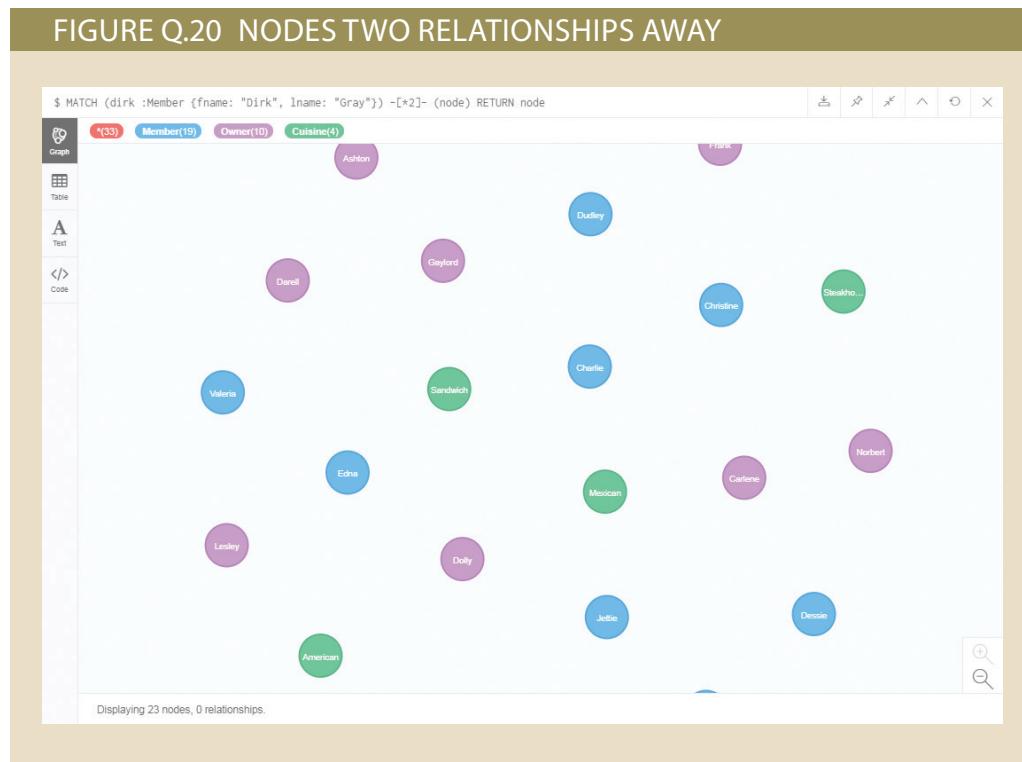


The previous command uses different variables for the two member nodes because we want the member in the first pattern to be any member, but we want the member in the second pattern to be one specific member who is different from the member in the first pattern. Similarly, we used different variables for the two reviewed relationships because we know that the reviews made by the other members are not the same as the reviews made by Dirk. However, we use the same variable for both restaurant nodes because we want the restaurants found in the first pattern to match the restaurants found in the second pattern.

Retrieving Data on Distant Relationships The previous section looked at immediate relationships, or relationships of one node being directly related to another node. The power of graph databases comes into play, however, when we begin to examine more distant relationships. As stated earlier, problems like six degrees of separation are what set graph databases apart in performance. Graph databases are designed for these types of problems, so Neo4j and the Cypher language make creating these types of queries very easy. If you understand retrieving data on immediate relationships, it is a simple modification to examine distant relationships. The only change is to specify a distance in the relationship of the pattern. The distance can be a specific number or it can be a range. Distances are indicated with an asterisk (*) in the relationship. For example, the following command returns all nodes that are two relationships away from Dirk Gray,

with the results shown in Figure Q.20. The relationship pattern ignores direction and contains no label so that the query returns every node that is connected through any type of relationship.

```
MATCH (dirk :Member {fname: "Dirk", lname: "Gray"}) -[*2]- (node)
RETURN node
```



Notice that the 23 nodes returned by the previous command include members and cuisines, but not restaurants. Each restaurant that Dirk reviewed is one relationship away. Other members who own or reviewed those same restaurants would be two relationships away. The cuisines associated with those restaurants are two relationships away. The command, however, specified to only show the nodes that are exactly two relationships away from Dirk, so the intervening restaurants that are one relationship away are not included in the output. Moving from a distance of two to a distance of three, four, or more is just a matter of changing the distance specified inside the relationship. This is how Neo4j easily deals with the types of “friends of friends of friends of friends” problems faced by social media sites that relational databases cannot handle well, as discussed earlier in this chapter.



Note

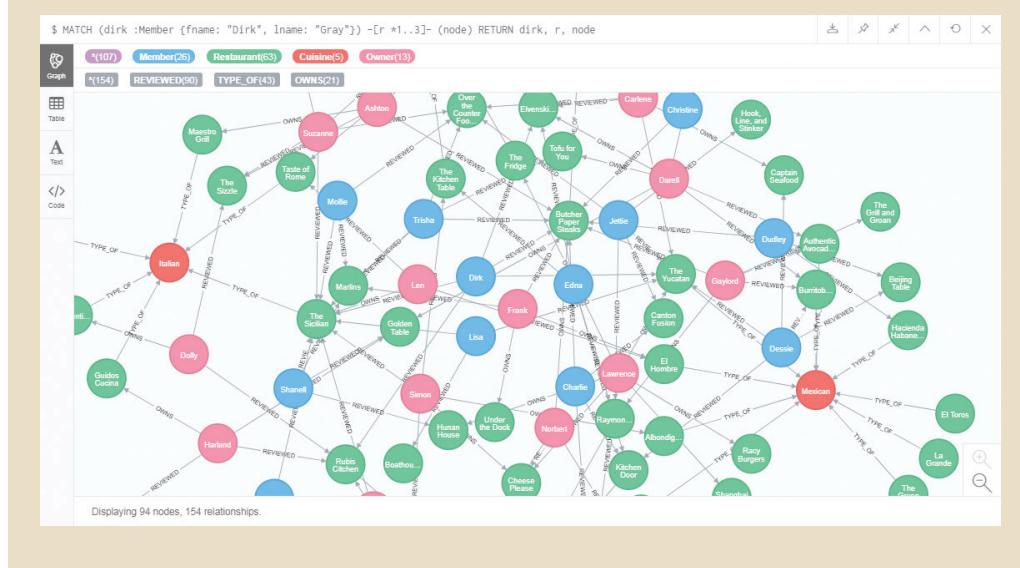
At first glance, Neo4j appears to provide conflicting information in Figure Q.20. At the top of the figure, the output shows *(33) indicating 33 nodes. At the bottom, it states that it is displaying 23 nodes. The discrepancy is because all owners are also members. Therefore, the 10 owner nodes are being counted twice because they are counted as members and then as owners.

Instead of providing a specific distance, a distance range can be supplied that allows the intervening nodes and relationships to be included in the output. The following command modifies the previous command to display the distance as a range from one to three relationships away from Dirk.

```
MATCH (dirk :Member {fname: "Dirk", lname: "Gray"}) -[r *1..3]- (node)
RETURN dirk, r, node
```

Notice that when specifying a range for the distance, the beginning and ending values for the range are separated with two dots (...). As shown in Figure Q.21, as the distance grows the number of nodes and relationships returned increases significantly.

FIGURE Q.21 NODES WITHIN A DISTANCE RANGE



As noted before, the last two commands are retrieving nodes based on any type of relationship. If desired, the type of relationship can be restricted. For example, if we want to return only the members who are up to four relationships away from Dirk, but who are connected only through restaurant reviews, the following command can be used:

```
MATCH (dirk :Member {fname: "Dirk", lname: "Gray"}) -[r :REVIEWS *1..4]- (m :Member)
RETURN dirk, r, m
```

Retrieving Paths in Neo4j One way to conceptualize relationships is to think of them as paths. For example, there is a path from Dirk to the restaurant “Rubis Citchen” to the member Lesley Haywood. Like finding distant relationships, graph databases excel at finding paths because this is a primary task for graph databases. Finding paths is also simple to program in Neo4j because the syntax is very similar to finding distant or immediate relationships. We are still focused on pattern matching when querying Neo4j, even when we are looking for a path. That is because all relationship patterns are paths! When we look for immediate relationships, we are looking for a path with a length of one relationship. When we look for a distant relationship with a distance in the range of

*1..3, we are looking for a path with a path length of three. When we look for a distant relationship with a specific distance of *4, we are looking for the endpoint node for a path with a path length of four.

The difference between retrieving a path and other retrievals that we have done so far is primarily in the way the data is returned. Let's consider two similar commands. First, the following command retrieves data about every restaurant reviewed by member Dirk Gray, with the text view of the data shown in Figure Q.22.

```
MATCH (dirk :Member {fname: "Dirk", lname: "Gray"}) -[r]-> (res :Restaurant)
RETURN dirk, r, res
```

FIGURE Q.22 TEXT VIEW OF RESTAURANTS REVIEWED BY DIRK GRAY

The screenshot shows the Neo4j browser interface with the following query in the 'Text' tab:

```
$ MATCH (dirk :Member {fname: "Dirk", lname: "Gray"}) -[r]-> (res :Restaurant) RETURN dirk, r, res
```

The results pane displays a list of nodes and relationships:

- Nodes:**
 - "dirk" (Member)
 - "r" (Relationship)
 - "res" (Restaurant)
- Relationships:**
 - Relationship from "dirk" to "res" with properties: {"atmosphere":4, "service":5, "taste":4, "value":3}, {"zip":47441, "city": "Vicksburg", "street": "84 Peppertree Loop", "price":4, "name": "The Sicilian", "state": "TN", "rid": 4636}
 - Relationship from "dirk" to "res" with properties: {"atmosphere":2, "service":2, "taste":2, "value":3}, {"zip":39441, "city": "Fort Mitchell", "street": "1075 Alpine Avenue", "price":4, "name": "Rubis Citchen", "state": "KY", "rid": 4668}
 - Relationship from "dirk" to "res" with properties: {"atmosphere":3, "service":1, "taste":1, "value":3}, {"zip":355-2512, "city": "Pine", "street": "966 Vilarian Street", "price":2, "name": "Elvenskirts", "state": "TN", "rid": 4632}
 - Relationship from "dirk" to "res" with properties: {"atmosphere":5, "service":5, "taste":4, "value":4}, {"zip":15262, "city": "Pittsburgh", "street": "367 Lakeview Drive", "name": "Butcher Paper Steaks", "state": "PA", "rid": 4663}
 - Relationship from "dirk" to "res" with properties: {"atmosphere":3, "service":1, "taste":1, "value":3}, {"phone": "(494) 555-6774", "price":2, "name": "The Yucatan", "rid": 4641}

Second, the following command retrieves data about the path for every restaurant reviewed by member Dirk Gray and assigns that path to the variable *p*. The text view of the data returned is shown in Figure Q.23.

```
MATCH p = (dirk :Member {fname: "Dirk", lname: "Gray"}) -[r]-> (res :Restaurant)
RETURN p
```

FIGURE Q.23 TEXT VIEW OF THE PATH OF REVIEWS BY DIRK GRAY

```
$ MATCH p = (dirk :Member {fname: "Dirk", lname: "Gray"})-[:REVIEWED_BY]-> (r)
p
[{"path": [{"id": 1032, "label": "Member", "properties": {"fname": "Dirk", "lname": "Gray", "birth": 1988, "mid": 1032, "state": "TN", "email": "dirgray@madeupemailaddresses.com", "username": "dirgray", "atmosphere": 4, "service": 5, "taste": 4, "value": 3, "zip": "47441", "phone": "(615) 555-2751", "city": "Vicksburg", "street": "784 Peppertree Loop", "price": 4, "name": "The Sicilian", "state": "TN", "rid": 4636}}, {"id": 4668, "label": "Member", "properties": {"fname": "Dirk", "lname": "Gray", "birth": 1988, "mid": 1032, "state": "TN", "email": "dirgray@madeupemailaddresses.com", "username": "dirgray", "atmosphere": 2, "service": 2, "taste": 2, "value": 3, "zip": "23941", "phone": "(502) 555-8037", "city": "Fort Mitchell", "street": "1075 Alpine Avenue", "price": 4, "name": "Rubis Citchen", "state": "KY", "rid": 4668}}, {"id": 4632, "label": "Member", "properties": {"fname": "Dirk", "lname": "Gray", "birth": 1988, "mid": 1032, "state": "TN", "email": "dirgray@madeupemailaddresses.com", "username": "dirgray", "atmosphere": 3, "service": 1, "taste": 1, "value": 3, "zip": "12203", "phone": "(615) 555-2512", "city": "Pine", "street": "966 Valarian Street", "price": 2, "name": "Elvenskirts", "state": "TN", "rid": 4632}}, {"id": 4663, "label": "Member", "properties": {"fname": "Dirk", "lname": "Gray", "birth": 1988, "mid": 1032, "state": "TN", "email": "dirgray@madeupemailaddresses.com", "username": "dirgray", "atmosphere": 5, "service": 5, "taste": 4, "value": 4, "zip": "15262", "phone": "(615) 555-7115", "city": "Pittsburgh", "street": "367 Lakeview Drive", "price": 5, "name": "Butcher Paper Steaks", "state": "TN", "rid": 4663}}, {"id": 4641, "label": "Member", "properties": {"fname": "Dirk", "lname": "Gray", "birth": 1988, "mid": 1032, "state": "TN", "email": "dirgray@madeupemailaddresses.com", "username": "dirgray", "atmosphere": 3, "service": 1, "taste": 1, "value": 3, "zip": null, "phone": "(404) 555-6774", "city": null, "street": null, "price": 2, "name": "The Yucatan", "state": "GA", "rid": 4641}}], "pathString": "[{"id": 1032, "label": "Member", "properties": {"fname": "Dirk", "lname": "Gray", "birth": 1988, "mid": 1032, "state": "TN", "email": "dirgray@madeupemailaddresses.com", "username": "dirgray", "atmosphere": 4, "service": 5, "taste": 4, "value": 3, "zip": "47441", "phone": "(615) 555-2751", "city": "Vicksburg", "street": "784 Peppertree Loop", "price": 4, "name": "The Sicilian", "state": "TN", "rid": 4636}}, {"id": 4668, "label": "Member", "properties": {"fname": "Dirk", "lname": "Gray", "birth": 1988, "mid": 1032, "state": "TN", "email": "dirgray@madeupemailaddresses.com", "username": "dirgray", "atmosphere": 2, "service": 2, "taste": 2, "value": 3, "zip": "23941", "phone": "(502) 555-8037", "city": "Fort Mitchell", "street": "1075 Alpine Avenue", "price": 4, "name": "Rubis Citchen", "state": "KY", "rid": 4668}}, {"id": 4632, "label": "Member", "properties": {"fname": "Dirk", "lname": "Gray", "birth": 1988, "mid": 1032, "state": "TN", "email": "dirgray@madeupemailaddresses.com", "username": "dirgray", "atmosphere": 3, "service": 1, "taste": 1, "value": 3, "zip": "12203", "phone": "(615) 555-2512", "city": "Pine", "street": "966 Valarian Street", "price": 2, "name": "Elvenskirts", "state": "TN", "rid": 4632}}, {"id": 4663, "label": "Member", "properties": {"fname": "Dirk", "lname": "Gray", "birth": 1988, "mid": 1032, "state": "TN", "email": "dirgray@madeupemailaddresses.com", "username": "dirgray", "atmosphere": 5, "service": 5, "taste": 4, "value": 4, "zip": "15262", "phone": "(615) 555-7115", "city": "Pittsburgh", "street": "367 Lakeview Drive", "price": 5, "name": "Butcher Paper Steaks", "state": "TN", "rid": 4663}}, {"id": 4641, "label": "Member", "properties": {"fname": "Dirk", "lname": "Gray", "birth": 1988, "mid": 1032, "state": "TN", "email": "dirgray@madeupemailaddresses.com", "username": "dirgray", "atmosphere": 3, "service": 1, "taste": 1, "value": 3, "zip": null, "phone": "(404) 555-6774", "city": null, "street": null, "price": 2, "name": "The Yucatan", "state": "GA", "rid": 4641}}]"}]
```

If you only view the graph view of the two commands, the results look identical. However, when looking at the table or text view, the difference in the data returned is unmistakable. The data in Figure Q.22 shows each variable, the node for Dirk, the relationship, and the node for the restaurant, being returned as separate values in separate objects. The data in Figure Q.23 shows the path being returned as a single value containing an array of objects. When querying interactively, this may seem like a small detail, but when embedded in a program, the difference is quite significant.

In addition to the changes in the way paths can be handled programmatically in applications, placing the path inside an array simplifies the ability to compare the lengths of paths by comparing the sizes of the arrays. This means that it is easy for Neo4j to complete tasks like finding the shortest path between two nodes because the DBMS merely has to compare the sizes of the arrays. This is done using a **shortestPath()** function in Cypher along with an unspecified relationship distance. The shortestPath() function takes a relationship pattern as a parameter. To create a relationship of unspecified length, use the asterisk to indicate a distant relationship, but do not provide a distance length or range. For example, the following command finds the shortest path, based on restaurant reviews, between members Dirk Gray and Isiah Horn, with results shown in Figure Q.24. The results indicate that the shortest path based only on restaurant reviews has a path length of six.

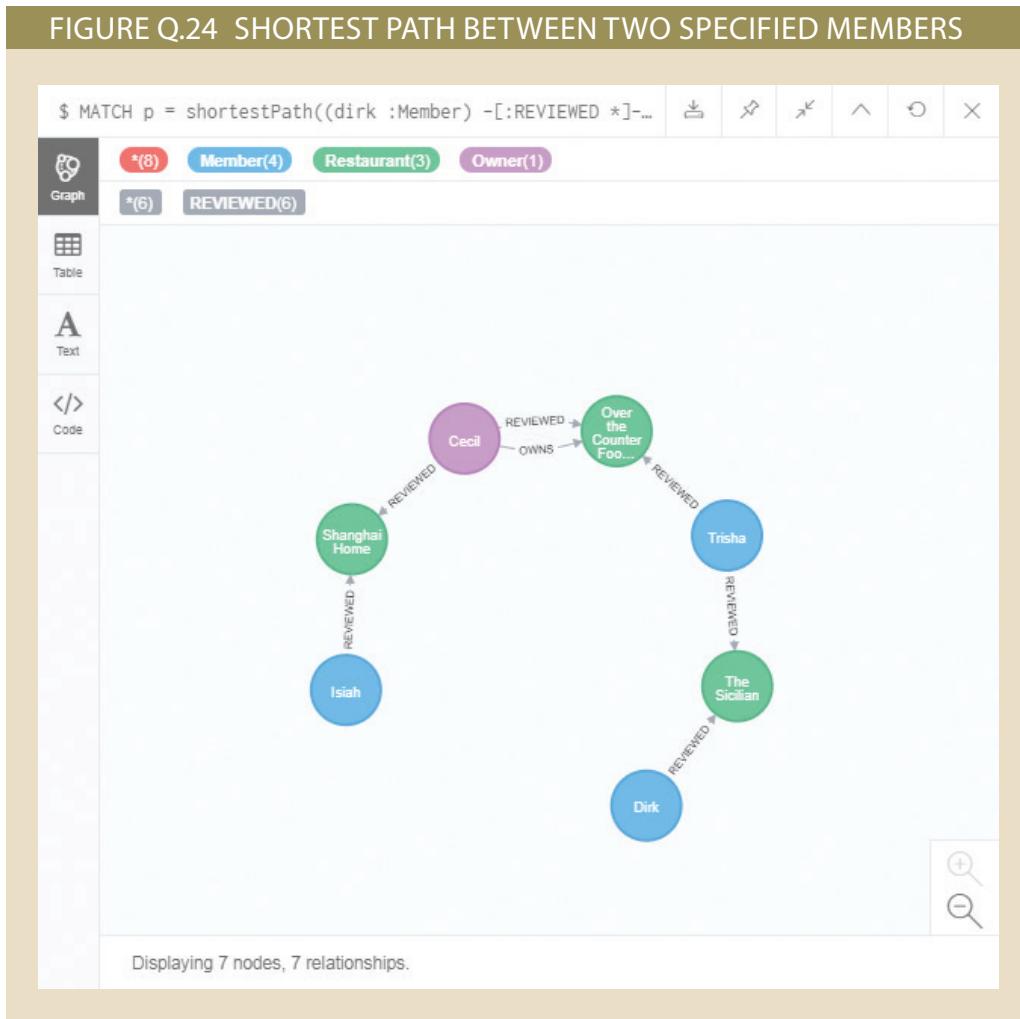
shortestPath()

A function in the Cypher language used by Neo4j to find the path with the shortest distance between two given nodes.

```

MATCH p = shortestPath((dirk :Member) -[:REVIEWED *]- (isiah :Member))
WHERE dirk.fname = "Dirk" AND dirk.lname = "Gray" AND isiah.fname = "Isiah"
AND isiah.lname = "Horn"
RETURN p

```



Creating Relationships in Neo4j Once you are comfortable with matching patterns, creating new relationships is straightforward. The simplest way to create a relationship between two nodes is to match the two nodes, and then specify the new relationship. For example, to create a restaurant review between Dirk Gray and The Sicilian, match the member node for Dirk, match the restaurant node for The Sicilian, and then issue a CREATE command that includes the pattern for the relationship desired:

```

MATCH (dirk :Member {fname: "Dirk", lname: "Gray"}), (sicilian :Restaurant
{name: "The Sicilian"})
CREATE (dirk) -[:REVIEWED {taste: 4, service: 5, atmosphere: 4, value:3}]-> (sicilian)

```

This appendix has introduced you to working with Neo4j as an example of a graph database. Graph databases are an important and growing segment of NoSQL database options that organizations are using to improve their ability to work with highly interdependent

data. You have been introduced to creating a graph database through the creation of nodes and relationships. You have seen that nodes are often indicated as being similar by sharing the same labels, but that even nodes with the same label can vary in their properties. Allowing the nodes to vary in properties makes Neo4j a schema-less database. In addition to properties of the nodes, relationships can also have properties. The graph can be traversed, or queried, using pattern matching. Patterns can specify nodes, relationships, and properties to identify paths through the graph. The ability to quickly match complex patterns allows graph databases to excel at queries that focus on the relationships among the data.

Summary

- Neo4j is a graph database that stores data as nodes and relationships, both of which can contain properties to describe them.
- Neo4j databases are queried using Cypher, a declarative language that shares many commonalities with SQL, but is still significantly different in many ways.
- Data retrieval is done primarily through the MATCH command to perform pattern matching.

Key Terms

Cypher, Q-4

match, Q-6

return, Q-6

shortestPath(), Q-28

Review Questions

1. Explain the difference between using the same variable name and different variable names when matching multiple patterns in Neo4j.
2. What is the difference between using WHERE and embedding properties in a node when creating a pattern in Neo4j?

Problems

For the following problems, use the Food Critics Club (FCC) graph database that was created and used earlier in the text for use with Neo4j.

1. Create a node that meets the following requirements. Use existing labels and property names as appropriate.

The node will be a member, and should be labeled as such, with member id 5000.

The member's name is "Abraham Greenberg".

Abraham was born in 1978, and lives in the state of "OH".

Abraham's email address is agreen@nomail.com, and his username is agberg.

2. Create a restaurant node with restaurant id is 10000, the name "Hungry Much", and located in Cobb Place, KY.
3. Update the "Hungry Much" restaurant created above to add the phone number "(931) 555-8888", and a price rating of 2.
4. Create a REVIEWED relationship between the member created above and the restaurant created above. The review should rate the restaurant as a 5 on taste, service, atmosphere, and value.
5. Create a REVIEWED relationship between member Frank Norwood and the restaurant created above. The review should rate the restaurant as a 4 on taste, service, and value, and rate the restaurant as a 2 on atmosphere.
6. Write a query to display member Frank Norwood and every restaurant that he has rated as a 4 or above on value.
7. Write a query to display cuisine, restaurant, and owner for every "American" or "Steakhouse" cuisine restaurant.
8. Write a query to return the shortest path based only on reviews between members Abraham Greenberg and Herb Christopher.

Online Content



The Ch14_FCC.txt file to create the graph database used in these questions was also used in Chapter 14. The file is available at www.cengagebrain.com. The contents of the file should be copied and pasted into the Neo4j editor bar and executed using the play button in the interface.