After studying Appendix H, you should be able to:

- Define the terms used when talking about databases
- Connect a database to an application
- Bind table and field objects to controls
- Explain the purpose of the DataSet, BindingSource, TableAdapter, and BindingNavigator objects
- Access the records in a dataset
- Write SQL SELECT statements
- Create a query using the Query Configuration Wizard
- Associate a Toolstrip control with a query
DATABASE TERMINOLOGY

In order to maintain accurate records, most businesses store information about their employees, customers, and inventory in files called databases. In general, a database is simply an organized collection of related information stored in a file on a disk.

Many computer products exist for creating databases; some of the most popular are Microsoft Access, Oracle, and Microsoft SQL Server. You can use Visual Basic to access the data stored in databases created by these products. This allows a company to create a standard interface in Visual Basic that employees can use to access database information stored in a variety of formats. Instead of learning each product’s user interface, the employee needs to know only one interface. The actual format of the database is unimportant and will be transparent to the user.

In this appendix, you learn how to access the data stored in a Microsoft Access database. Databases created by Microsoft Access are relational databases. A relational database is one that stores information in tables composed of columns and rows, similar to the format used in a spreadsheet. Each column in a table represents a field, and each row represents a record.

A field is a single item of information about a person, place, or thing—for example, a name, a salary amount, a Social Security number, or a price. A record is a group of related fields that contain all of the necessary data about a specific person, place, or thing. The college you are attending keeps a student record on you. Examples of fields contained in your student record include your Social Security number, name, address, phone number, credits earned, grades earned, and grade point average. The place where you are employed also keeps a record on you. Your employee record contains your Social Security number, name, address, phone number, starting date, salary or hourly wage, and so on. A group of related records is called a table. Each record in a table pertains to the same topic, and each contains the same type of information. In other words, each record in a table contains the same fields.

A relational database can contain one or more tables. A one-table database would be a good choice for storing the information regarding the college courses you have taken. An example of such a table is shown in Figure H.1.

<table>
<thead>
<tr>
<th>ID</th>
<th>Title</th>
<th>Hours</th>
<th>Grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>CIS100</td>
<td>Intro to Computers</td>
<td>3</td>
<td>A</td>
</tr>
<tr>
<td>Eng100</td>
<td>English Composition</td>
<td>3</td>
<td>B</td>
</tr>
<tr>
<td>Phil105</td>
<td>Philosophy Seminar</td>
<td>2</td>
<td>C</td>
</tr>
<tr>
<td>CIS201</td>
<td>Visual Basic 2005</td>
<td>3</td>
<td>A</td>
</tr>
</tbody>
</table>

**FIGURE H.1** Example of a one-table relational database
Notice that each record in the table contains four fields: an ID field that indicates the department name and course number, a course title field, a number of credit hours field, and a grade field. In most tables, one of the fields uniquely identifies each record and is called the **primary key**. In the table shown in Figure H.1, you could use either the ID field or the Title field as the primary key, because the data in those fields will be unique for each record.

To store information about your CD (compact disc) collection, you typically use a two-table database: one table to store the general information about each CD (such as the CD’s name and the artist’s name) and the other table to store the information about the songs on each CD (such as their title and track number). You then use a common field—for example, a CD number—to relate the records contained in both tables. Figure H.2 shows an example of a two-table database that stores CD information.

The first table shown in Figure H.2 is often referred to as the **parent table**, and the second table is referred to as the **child table**. In the parent table, the Number field is the primary key, because it uniquely identifies each record in that table. In the child table, the Number field is used solely to link the song title and track information to the appropriate CD in the parent table. In the child table, the Number field is called the **foreign key**.

Storing data in a relational database offers many advantages. The computer can retrieve data stored in a relational format both quickly and easily, and the data can be displayed in any order. For example, the information in the CD database shown in Figure H.2 can be arranged by artist name, song title, and so on. A relational database also allows you to control the amount of information you want to view at a time. You can view all of the information in the CD database, or you can view only the information pertaining to a certain artist, or only the names of the songs contained on a specific CD.
**ADO.NET 2.0**

When a Visual Basic 2005 application needs to access the information stored in a database, the computer uses a technology called **ADO.NET 2.0** to connect the application to the database. (ADO stands for ActiveX Data Objects.) The connection allows the application to read information from and write information to the database.

After creating the connection between the database and the application, the computer opens the database and then makes a copy of the fields and records the application wants to access. The computer stores the copy, called a **dataset**, in its internal memory. The computer then closes both the database and the connection to the database. Notice that the connection is a temporary one. The computer reconnects the application to the database only when the application requests further information from the database, or when changes made to the dataset (which is in internal memory) need to be saved. After retrieving the additional information or saving any changes, the computer again closes the database and the connection to the database.

In the concepts section of this appendix, you learn how to access the data contained in a Microsoft Access database named Employees. The database is stored in a file named Employees.mdb, which is located in the VbReloaded\Appendix H\Access Databases folder. The Employees database contains one table, which is named tblEmploy. (It is not necessary to begin a table name with “tbl”.) Figure H.3 shows the table data displayed in a window in the IDE. You can open a database table in a window in the IDE by connecting the database to an application, then right-clicking the table’s name in the Server Explorer window, then clicking Show Table Data.

The **tblEmploy** table contains seven fields and 12 records. The Emp_Number, Last_Name, First_Name, Hired, and Rate fields store employee numbers, last names, first names, hire dates, and rates of pay, respectively. The Status field contains the employment status, which is either the letter F (for fulltime) or the letter P (for parttime). The Code field identifies the employee’s department: 1 for Accounting, 2 for Advertising, 3 for Personnel, and 4 for Inventory. In the **tblEmploy** table, the Emp_Number field is the primary key, because it uniquely identifies each record.
CONNECTING A DATABASE TO AN APPLICATION

Before an application can access the data stored in a database, you need to connect the database to the application. You can use the Data Source Configuration Wizard to make the connection. Figure H.4 shows the procedure you follow when using the Data Source Configuration Wizard to connect a database to an application.

HOW TO...

1. Open the application’s solution file. Click Data on the menu bar, and then click Show Data Sources to open the Data Sources window.

2. Click the Add New Data Source link in the Data Sources window. This opens the Data Source Configuration Wizard dialog box and displays the Choose a Data Source Type screen. If necessary, click Database. See Figure H.5. As the figure indicates, the Database option in the dialog box allows you to connect to a database and choose the database objects (such as tables and fields) for your application. The option also creates a dataset that contains the chosen objects.

3. Click the Next button, and then continue using the Data Source Configuration Wizard to specify the data source. The Wizard will add several files to the project. The names of the files will appear in the Solution Explorer window. (Detailed steps for using the wizard can be found in the Programming Tutorial section of this chapter.)

4. When the Wizard is finished, it adds a dataset to the Data Sources window. Figure H.6 shows an example of a dataset added to the Data Sources window in the Morgan Industries application. In this case, the dataset’s name is EmployeesDataSet, and it contains one table object and seven field objects. The table object is the tblEmploy table contained in the Employees database, and the seven field objects correspond to the seven fields in the table.

FIGURE H.4 How to connect a database to an application
Previewing the Data Contained in a Dataset

As you learned in the previous section, the Data Source Configuration Wizard creates a dataset and adds it to the Data Sources window. You can use the procedure shown in Figure H.7 to preview the contents of the dataset.
Connecting a Database to an Application

HOW TO...

**Preview the Contents of a Dataset**

1. Right-click the Data Sources window, and then click Preview Data. This opens the Preview Data dialog box. (Alternatively, you can click the form, then click Data on the menu bar, and then click Preview Data on the menu.)

2. Select the object to preview, and then click the Preview button. The Preview Data dialog box in Figure H.8 shows the contents of the EmployeesDataSet.

3. When you are finished previewing the data, click the Close button in the dialog box.

**FIGURE H.7** How to preview the contents of a dataset

![Preview Data dialog box](Image)

Notice that `EmployeesDataSet.tblEmploy.Fill, GetData()` appears in the Select an object to preview box in Figure H.8. EmployeesDataSet is the name of the dataset in the application, and tblEmploy is the name of the table included in the dataset. Fill and GetData are methods. The Fill method populates an existing table with data, while the GetData method creates a new table and populates it with data.

**FIGURE H.8** Contents of the EmployeesDataSet displayed in the Preview Data dialog box

Notice that `EmployeesDataSet.tblEmploy.Fill, GetData()` appears in the Select an object to preview box in Figure H.8. EmployeesDataSet is the name of the dataset in the application, and tblEmploy is the name of the table included in the dataset. Fill and GetData are methods. The Fill method populates an existing table with data, while the GetData method creates a new table and populates it with data.
BINDING THE OBJECTS IN A DATASET

For the user to view the contents of a dataset while an application is running, you need to connect one or more objects in the dataset to one or more controls in the interface. Connecting an object to a control is called binding, and the connected controls are called bound controls. Figure H.9 lists various ways of binding the objects in a dataset. Notice that you can bind an object either to a control that the computer creates for you, or to an existing control in the interface.

HOW TO...

Bind the Objects in a Dataset

To have the computer create a control and then bind an object to it:
1. In the Data Sources window, click the object you want to bind.
2. If necessary, use the object’s list arrow to change the control type.
3. Drag the object to an empty area on the form, then release the mouse button. The computer creates the appropriate control and binds the object to it.

To bind an object to an existing control:
1. In the Data Sources window, click the object you want to bind.
2. Drag the object to the control on the form, then release the mouse button.
   OR
1. Click the control on the form.
2. Use the Properties window to set the appropriate property or properties.
   (Refer to the Binding to an Existing Control section in this appendix.)

Having the Computer Create a Bound Control

As indicated in Figure H.9, one way to bind an object from a dataset is to drag the object to an empty area on the form. When you do this, the computer creates the necessary control and automatically binds the object to it. The icon that appears before the object’s name in the Data Sources window indicates the type of control the computer will create. For example, the icon shown in Figure H.10 indicates that the computer will create a DataGridView control when you drag the tblEmploy table object to the form. A DataGridView control displays the table data in a row and columnar format, similar to a spreadsheet. Each row in the control represents a record, and each column represents a field. The icon, on the other hand, indicates that the computer will create a text box when you drag a field object to the form.
You can use the list arrow that appears next to an object’s name to change the type of control the computer will create. For example, to display the tblEmploy data in separate text box controls, rather than in a DataGridView control, you first click tblEmploy in the Data Sources window. You then click its list arrow, as shown in Figure H.11, and then click Details in the list. The Details option tells the computer to create a separate control for each field in the table.

Similarly, to display the Last_Name field data in a label control rather than in a text box, you first click Last_Name in the Data Sources window. You then click its list arrow, as shown in Figure H.12, and then click Label in the list.
Using ADO.NET 2.0 with Microsoft Access Databases

Figure H.13 shows the result of dragging the tblEmploy table object to the MainForm, using the default control type for a table, and then sizing the control. In this case, the DataGridView control was sized by first setting its AutoSizeColumnsMode property to Fill. The Fill setting automatically adjusts the column widths so that all of the columns exactly fill the display area of the control. The control’s right border was then dragged to the desired width.

Besides adding a DataGridView control to the form, the computer also adds a BindingNavigator control. When an application is running, the BindingNavigator control allows you to move from one record to the next in the dataset. It also allows you to add and delete a record, as well as save any changes made to the dataset.

In addition, the computer places four objects in the component tray: a DataSet, a BindingSource, a TableAdapter, and a BindingNavigator. As you learned in Chapter 2, the component tray stores objects that do not appear in the user interface when an application is running. An exception to this is the BindingNavigator object, which appears as the BindingNavigator control during both design time and run time. Figure H.14 illustrates the relationships among the database, the objects in the component tray, and the controls on the form.
The TableAdapter object connects the database to the DataSet object, which stores the information you want to access from the database. The TableAdapter is responsible for retrieving the appropriate information from the database and storing it in the DataSet. It also is responsible for saving to the database any changes made to the data contained in the DataSet.

The BindingSource object provides the connection between the DataSet object and the bound controls on the form. For example, the TblEmployBindingSource object (shown earlier in Figure H.13) connects the EmployeesDataSet object to two bound controls: the DataGridView control and the BindingNavigator control. The BindingSource object allows the DataGridView control to display the data contained in the dataset. It also allows the BindingNavigator control to access the records stored in the dataset.

If a table object’s control type is changed from DataGridView to Details, the computer automatically provides the appropriate controls (such as text boxes, labels, and so on) when you drag the table object to the form. It also adds the BindingNavigator control to the form, and adds the DataSet, BindingSource, TableAdapter, and BindingNavigator objects to the component tray. The appropriate controls and objects are also automatically included when you drag a field object to an empty area on the form.

When a table or field object is dragged to the form, the computer not only adds the appropriate controls and objects to the application, but it also enters some code in the Code Editor window. Figure H.15 shows the code that is automatically entered in the Code Editor window when the tblEmploy object is dragged to the form.
As Figure H.15 indicates, two event procedures are automatically entered in the Code Editor window: TblEmployBindingNavigatorSaveItem_Click and MainForm_Load. The TblEmployBindingNavigatorSaveItem_Click procedure is processed when you click the Save button on the BindingNavigator control. The code in the procedure saves, to the database, any changes made to the dataset. It does this using two methods: the EndEdit method of the BindingSource object and the Update method of the TableAdapter object. The EndEdit method applies any pending changes (such as new records, deleted records, or changed records) to the dataset. The Update method commits, to the database, the changes made to the dataset.
The syntax of the EndEdit method is \( [\text{Me.}] \text{bindingSourceName.EndEdit()} \), where \( \text{bindingSourceName} \) is the name of the BindingSource object in the application. The syntax of the Update method is \( [\text{Me.}] \text{tableAdapterName.Update([Me.] dataSetName.tableName}) \). In the syntax, \( \text{tableAdapterName} \) is the name of the TableAdapter object in the application. \( \text{dataSetName} \) is the name of the Dataset object, and \( \text{tableName} \) is the name of the table object contained in the dataset. The \( \text{Me.} \) in the syntax for both methods is optional, as indicated by the square brackets; when used, it refers to the current form.

The MainForm's Load event procedure, on the other hand, is processed when the application is started and the form is loaded into the computer's internal memory. Notice that the Load procedure contains one statement: \( \text{Me.TblEmployTableAdapter.Fill(Me.EmployeesDataSet.tblEmploy)} \). The statement uses the TableAdapter object's \text{Fill method} to retrieve data from the database and store it in the dataset. The Fill method's syntax is \( [\text{Me.}] \text{tableAdapterName.Fill([Me.] dataSetName.tableName}) \). In the syntax, \( \text{tableAdapterName} \) is the name of the TableAdapter object in the application. \( \text{dataSetName} \) is the name of the Dataset object, and \( \text{tableName} \) is the name of the table object contained in the dataset.

In most applications, the statement to fill a dataset with data belongs in the form's Load event procedure, as shown in Figure H.15. However, as the comments in the Load event procedure indicate, you can move the statement to another procedure; or, you can remove the statement from the Code Editor window.

Figure H.16 shows a sample run of the Morgan Industries application. When the application is started, the statement in the MainForm's Load event procedure retrieves the appropriate data from the Employees database and loads the data into the EmployeesDataSet object. As the figure shows, the data is displayed in the DataGridView control, which is bound to the tblEmploy table contained in the EmployeesDataSet.

You can make a change to an existing record by modifying the data in the appropriate cell in the DataGridView control. A cell is an intersection of a row and column in the control. For example, to change Martha Vine's status from...
part-time to full-time, you position the cursor in the cell located in the seventh row, sixth column. You then replace the letter P with the letter F.

Recall that the EmployeesDataSet object is also bound to the TblEmployBindingNavigator control, which is shown in Figure H.17.

**FIGURE H.17** TblEmployBindingNavigator control

You can use the buttons on the TblEmployBindingNavigator control to access the first and last records in the EmployeesDataSet, as well as to access the previous and next records. You also can use the TblEmployBindingNavigator control to access a record by its record number. The first record in a dataset has a record number of one, the second has a record number of two, and so on. To access record number five, you type the number 5 in the box that contains the current record number, and you then press the Enter key on your keyboard. As shown in Figure H.17, the TblEmployBindingNavigator control also contains buttons that allow you to add a record to the dataset, delete a record from the dataset, and save the changes made to the dataset.

**Binding to an Existing Control**

As indicated earlier in Figure H.9, you also can bind an object in a dataset to an existing control on the form. The easiest way to do this is by dragging the object from the Data Sources window to the control. However, you also can click the control and then set one or more properties in the Properties window. The appropriate property (or properties) to set depends on the control you are binding. For example, you use the DataSource and DataMember properties to bind a DataGridView control. However, you use the DataSource and DisplayMember properties to bind a ListBox control. To bind label and text box controls, you use the DataBindings/Text property.

When you drag an object from the Data Sources window to an existing control, the computer does not create a new control; rather, it merely binds the object to the existing control. Because a new control does not need to be created, the computer ignores the control type specified for the object in the Data Sources window. As a result, it is not necessary to change the control type in the Data Sources window to match the existing control’s type. In other words, you can drag an object that is associated with a text box in the Data Sources window to a label control on the form. The computer will bind the object to the label, but it will not change the label to a text box. Figure H.18 shows the result of dragging four field objects to four existing label controls in the Morgan Industries application. In this case, the computer will bind the Emp_Number, Last_Name, Status, and Code field objects to the numberLabel, lastNameLabel, statusLabel, and codeLabel controls, respectively.
In addition to binding the field objects to the appropriate controls, the computer also adds the DataSet, BindingSource, and TableAdapter objects to the component tray. However, notice that it does not include the BindingNavigator object and BindingNavigator control in the application. You can use the BindingNavigator tool in the Toolbox to add a BindingNavigator control to the form; doing this also adds a BindingNavigator object to the component tray. You then must set the BindingNavigator control’s DataSource property to the name of the BindingSource object in the application. In this case, for example, you would set the control’s DataSource property to TblEmployBindingSource.

Figure H.19 shows the code that is automatically entered in the Code Editor window when you drag an object from the Data Sources window to an existing control on the form.

Visual Basic code

' Project name: Morgan Industries Project
' Project purpose: The project displays the employee records stored in a database.
' Created/revised by: <your name> on <current date>

Option Explicit On
Option Strict On

(Figure is continued on next page)
Using ADO.NET 2.0 with Microsoft Access Databases

As you learned earlier, the `Me.TblEmployTableAdapter.Fill(Me.EmployeesDataSet.tblEmploy)` statement in the form's Load event procedure fills the dataset with data. Figure H.20 shows a sample run of this version of the Morgan Industries application.

![Figure H.20 - Sample run of a different version of the Morgan Industries application](Image)

Notice that only the first record in the dataset appears in the interface shown in Figure H.20. Because the form does not contain a BindingNavigator control, which would allow you to move from one record to the next, you will need to code the Next and Previous buttons to access the other records in the dataset.

### Accessing the Records in a Dataset

The BindingSource object uses an invisible record pointer to keep track of the current record. It stores the position of the record pointer in its **Position property**. The first record is in position zero, the second is in position one, and
so on. For example, if the record pointer is pointing to the third record, the Position property contains the integer 2. Figure H.21 shows the Position property's syntax and includes examples of using the property.

**HOW TO...**

**Use the BindingSource Object's Position Property**

**Syntax**

`bindingSourceName.Position`

**Examples**

`recordNum = TblEmployBindingSource.Position`

assigns the position of the current record to the `recordNum` variable

`TblEmployBindingSource.Position = 4`

moves the record pointer to the fifth record in the dataset

`TblEmployBindingSource.Position = _
    TblEmployBindingSource.Position + 1`

moves the record pointer to the next record in the dataset

![FIGURE H.21](How to use the BindingSource object's Position property)

In the first example shown in Figure H.21, the `recordNum = TblEmployBindingSource.Position` statement assigns the record pointer’s position to the `recordNum` variable. The `TblEmployBindingSource.Position = 4` statement in the second example moves the record pointer to the fifth record in the dataset. (Recall that the first record is in position zero.) In the last example, the `TblEmployBindingSource.Position = TblEmployBindingSource.Position + 1` statement moves the record pointer to the next record in the dataset.

In addition to using the Position property to move the record pointer in a dataset, you also can use the Move methods of the BindingSource object. Figure H.22 shows each Move method’s syntax and includes an example of using each method. Notice that you can use the Move methods to move the record pointer to the first, last, next, and previous record in the dataset associated with the BindingSource object.

**HOW TO...**

**Use the BindingSource Object's Move Methods**

**Syntax**

`bindingSourceName.MoveNext()`

`bindingSourceName.MoveLast()`

`bindingSourceName.MoveNext()`

`bindingSourceName.MovePrevious()`

(Figure is continued on next page)
Using ADO.NET 2.0 with Microsoft Access Databases

When the Next button in the Morgan Industries application is clicked, its Click event procedure should move the record pointer to the next record in the dataset. Similarly, when the Previous button is clicked, its Click event procedure should move the record pointer to the previous record in the dataset. You can use the TblEmployBindingSource object's MoveNext and MovePrevious methods to code the procedures, as shown in Figure H.23.

**Examples**

TblEmployBindingSource.MoveNext() moves the record pointer to the next record in the dataset

TblEmployBindingSource.MoveLast() moves the record pointer to the last record in the dataset

TblEmployBindingSource.MoveNext() moves the record pointer to the next record in the dataset

TblEmployBindingSource.MovePrevious() moves the record pointer to the previous record in the dataset

**DATASET DESIGNER**

As you learned earlier, data stored in a relational database can be displayed in any order. For example, you can arrange the data contained in the Employees database by employee number, pay rate, status, and so on. A relational database also allows you to control the amount of information you want to view at a time. You can view all of the information in the Employees database, or you can view only the employee numbers and corresponding pay rates. You also can choose to view only the records for the part-time employees.
You can use the Dataset Designer to indicate the order in which you want data displayed, and also to specify the fields and records you want to view from the database. Figure H.24 shows various ways of opening the Dataset Designer, and Figure H.25 shows the Dataset Designer open in the IDE.

**Open the Dataset Designer**

- Right-click the Data Sources window, then click Edit DataSet with Designer.
- In the Solution Explorer window, right-click the name of the schema file, which has an .xsd extension, then click View Designer.
- Right-click the TableAdapter object in the component tray, then click Edit Queries in DataSet Designer.

Notice that the Solution Explorer window shown in Figure H.25 contains a file named EmployeesDataSet.xsd. The filename also appears on the tab in the Dataset Designer. The .xsd extension on the filename indicates that the file is an XML schema definition file. **XML**, which stands for **Extensible Markup Language**, is a text-based language used to store and share data between applications and across networks and the Internet. An **XML schema definition file** defines the tables and fields that make up the dataset. The .xsd file is added to the Solution Explorer window when you connect a database to an application.

The Dataset Designer in Figure H.25 shows the name of the table object included in the EmployeesDataSet. It also lists the names of the seven field objects included in the dataset. In addition, it shows the name of the application’s TableAdapter object, which connects the dataset to its underlying database.

Every TableAdapter object contains one or more queries, which are listed below the TableAdapter object’s name in the Dataset Designer. A **query** specifies the fields and records to retrieve from the database, as well as the order in which to arrange the fields and records. As Figure H.25 indicates, the TableAdapter object in the

---

**TIP**
The key icon in Figure H.25 indicates that the Emp_Number field is the primary field in the tblEmploy table.
Morgan Industries application contains one query. The query is associated with two methods named Fill and GetData. As you learned earlier, the Fill method populates an existing table with data, and the GetData method creates a new table and populates it with data.

You can view the information contained in a query by right-clicking the query in the DataSet Designer window, and then clicking Configure. Doing this opens the TableAdapter Configuration Wizard, as shown in Figure H.26.

Queries are created using a special language called Structured Query Language or, more simply, SQL. SQL, which is pronounced ess-cue-el, is a set of commands that allows you to access and manipulate the data stored in many database management systems on computers of all sizes, from large mainframes to small microcomputers. You can use SQL commands to perform database tasks such as storing, retrieving, updating, deleting, and sorting data.

The SELECT statement, which appears in Figure H.26, is the most commonly used command in SQL. The SELECT statement allows you to specify the fields and records you want to view, as well as control the order in which the fields and records appear when displayed. Figure H.27 shows the basic syntax of the SELECT statement and includes several examples of using the statement to access the data stored in the Employees database (shown earlier in Figure H.3). As you may remember, the database contains one table (named tblEmploy) and seven fields. The Emp_Number, Rate, and Code fields contain numeric data. The Last_Name, First_Name, and Hired fields contain text data, and the Status field contains one character.
**How to Use the SELECT Statement**

**Syntax**

```
SELECT fields FROM table [WHERE condition] [ORDER BY field]
```

**Example 1**

```
SELECT Emp_Number, Last_Name, First_Name, Hired, Rate, Status, Code FROM tblEmploy
```

selects all of the fields and records in the table

**Example 2**

```
SELECT Emp_Number, First_Name, Last_Name FROM tblEmploy
```

selects the Emp_Number, First_Name, and Last_Name fields from each record in the table

**Example 3**

```
SELECT Emp_Number, Last_Name, First_Name, Hired, Rate, Status, Code FROM tblEmploy WHERE Status = 'F'
```

selects the records for full-time employees

**Example 4**

```
SELECT Emp_Number, Rate FROM tblEmploy WHERE Code = 3
```

selects the Emp_Number and Rate fields for employees in the Personnel department

**Example 5**

```
SELECT Emp_Number FROM tblEmploy WHERE Last_Name LIKE 'Smith'
```

selects the Emp_Number field for records that have a last name of Smith

**Example 6**

```
SELECT Emp_Number FROM tblEmploy WHERE Last_Name LIKE 'S%'
```

selects the Emp_Number field for records whose last name begins with the letter S

**Example 7**

```
SELECT Emp_Number, Last_Name, First_Name, Hired, Rate, Status, Code FROM tblEmploy ORDER BY Code
```

selects all of the fields and records in the table, and sorts the records in ascending order by the Code field

**Example 8**

```
SELECT Emp_Number, Last_Name, First_Name, Hired, Rate, Status, Code FROM tblEmploy WHERE Status = 'p' ORDER BY Code
```

selects the records for part-time employees, and sorts the records in ascending order by the Code field

In the SELECT statement’s syntax, `fields` is one or more field names (separated by commas), and `table` is the name of the table containing the fields. Notice that the SELECT statement’s syntax contains two clauses that are optional: the `WHERE` clause and the `ORDER BY` clause. The **WHERE clause** allows you to limit the records that will be selected, and the **ORDER BY clause** allows you to control the order in which the records appear when displayed. Although
you do not have to capitalize the keywords SELECT, FROM, WHERE, and ORDER BY in a SELECT statement, many programmers do so for clarity.

Study the examples shown in Figure H.27. The SELECT statement in Example 1 selects all of the fields and records from the tblEmploy table. The SELECT statement in Example 2 selects only three of the fields from each record in the table. The SELECT statement in Example 3 uses the WHERE clause to limit the records that will be selected. In this case, the statement indicates that only records for full-time employees should be selected. Notice that, when comparing the contents of the Status field (which contains one character only) with a character, you enclose the character in single quotation marks.

The SELECT statement in Example 4 selects only the Emp_Number and Rate fields for employees working in the Personnel department. The SELECT statement in Example 5 shows how you can compare the contents of a text field with a string. You do so using the Like operator in the WHERE clause, and you enclose the string in single quotation marks. The SELECT statement in Example 5 will select the Emp_Number field for all records that have Smith as the last name.

The SELECT statement in Example 6 shows how you can use the Like operator along with the % (percent sign) wildcard in the WHERE clause. As the example indicates, the SELECT Emp_Number FROM tblEmploy WHERE Last_Name LIKE ‘S%’ statement will select the Emp_Number field for all records with a last name that begins with the letter S.

Example 7’s SELECT statement selects all of the fields and records from the tblEmploy table, and then uses the ORDER BY clause to sort the records in ascending order by the Code field. To sort the records in descending order, you use SELECT Emp_Number, Last_Name, First_Name, Hired, Rate, Status, Code FROM tblEmploy ORDER BY Code DESC. The “DESC” stands for “descending”.

The last SELECT statement shown in Figure H.27 selects the records for part-time employees, and it sorts the records in ascending order by the Code field. Notice that the statement compares the contents of the Status field (which contains uppercase letters) with a lowercase ‘p’. The statement works correctly because SQL commands are not case sensitive.

Creating a New Query
As mentioned earlier, a TableAdapter object can contain more than one query. Figure H.28 shows how to use the Query Configuration Wizard to create a new query.

HOW TO...

Create a Query Using the Query Configuration Wizard
1. Right-click the name of the TableAdapter in the DataSet Designer, then click Add Query to open the TableAdapter Query Configuration Wizard. (Alternatively, you can right-click the DataSet Designer, point to Add, and then click Query. Or, you can click the DataSet Designer, then click Data on the menu bar, then point to Add, and then click Query.)

2. Depending on the way you opened the TableAdapter Query Configuration Wizard, you may need to specify the data connection. If necessary, select the appropriate data connection, then click the Next button.

(Figure is continued on next page)
3. In the Choose a Command Type screen, select the Use SQL statements radio button, if necessary, then click the Next button.

4. In the Choose a Query Type screen, select the SELECT which returns rows radio button, if necessary, then click the Next button.

5. When the Specify a SQL SELECT statement screen appears, you can either type the SELECT statement yourself, or you can use the Query Builder dialog box to construct the statement for you. Figure H.29 shows a SELECT statement entered on the Specify a SQL SELECT statement screen. (You learn how to use the Query Builder dialog box in the next section.)

6. When you have completed the SELECT statement, click the Next button in the Specify a SQL SELECT statement screen.

7. In the Choose Methods to Generate screen, select the Fill a DataTable and Return a DataTable check boxes, if necessary. Enter appropriate names for the query’s FillBy and GetDataBy methods. For example, if the query retrieves the records of full-time employees only, you might assign the names FillByFulltime and GetDataByFulltime to their associated methods, as shown in Figure H.30. Click the Next button.

8. In the Wizard Results screen, click the Finish button. The query is added to the DataSet Designer, as shown in Figure H.31.
Using the Query Builder Dialog Box

Earlier, in Figure H.29, you viewed a SELECT statement entered on the Specify a SQL SELECT statement screen. Rather than typing the SELECT statement on your own, you can use the Query Builder dialog box to construct it for you. Figure H.32 shows an example of a completed Query Builder dialog box.
In the Query Builder dialog box shown in Figure H.32, the word Descending and the number 1 appear in the Sort Type and Sort Order columns, respectively, for the Rate field. As a result, the Query Builder adds the ORDER BY Rate DESC clause to the SELECT statement. The =3 that appears in the Filter column for the Code field in the dialog box tells the Query Builder to include the WHERE (Code = 3) clause in the statement. When it is processed, the SELECT statement in the dialog box will select all of the fields from the tblEmploy table, but only for records having the number 3 in their Code field. In other words, it will select only the records of employees in the Personnel department. The records will be arranged in descending order by their pay rate.

Allowing the User to Run a Query

You can provide a ToolStrip control that allows the user to run a query after an application has been started. Figure H.33 shows the steps you follow to include a ToolStrip control on a form, and associate the control with a query.
HOW TO...

Add a ToolStrip Control That Is Associated with a Query

1. Right-click the name of the TableAdapter in the component tray, then click Add Query.

2. Select the Existing query name radio button in the Search Criteria Builder dialog box, then click the down arrow in the list box that appears next to the radio button. Click the name of the query in the list. See Figure H.34.

3. Click the OK button to close the Search Criteria Builder dialog box. The computer adds a ToolStrip control to the form and a ToolStrip object to the component tray, as shown in Figure H.35. The control and object are associated with the query selected in Step 2.

**FIGURE H.33** How to add a ToolStrip control that is associated with a query

**FIGURE H.34** Search Criteria Builder dialog box
To run the FillByFulltime query in the Morgan Industries application shown in Figure H.35, the user needs to start the application and then click the FillByFulltime button on the ToolStrip control. Figure H.36 shows the result of clicking the button.

The DataGridView control displays only the full-time employee records.

To run the FillByFulltime query in the Morgan Industries application shown in Figure H.35, the user needs to start the application and then click the FillByFulltime button on the ToolStrip control. Figure H.36 shows the result of clicking the button.

The DataGridView control displays only the full-time employee records.
You have completed the concepts section of Appendix H. The next section is the Programming Tutorial section, which gives you step-by-step instructions on how to apply the appendix's concepts to an application. A Programming Example follows the Programming Tutorial. The Programming Example is a completed program that demonstrates the concepts taught in the appendix. Following the Programming Example are the Quick Review, Key Terms, Self-Check Questions and Answers, Review Questions, Review Exercises – Short Answer, Computer Exercises, and Case Projects sections.

**PROGRAMMING TUTORIAL**

**Trivia Game Application**

In this tutorial, you create an application that displays trivia questions and answers. The questions and answers are stored in a table named Game1, which is contained in a Microsoft Access database named Trivia.mdb. The Game1 table contains nine records. Each record contains six fields named Question, AnswerA, AnswerB, AnswerC, AnswerD, and CorrectAnswer. The application also keeps track of the number of incorrect responses made by the user, and it displays that information after all nine questions have been answered. The Trivia Game application’s user interface is shown in Figure H.37, and its TOE chart is shown in Figure H.38.
**TOE Chart:**

<table>
<thead>
<tr>
<th>Task</th>
<th>Object</th>
<th>Event</th>
</tr>
</thead>
<tbody>
<tr>
<td>End the application</td>
<td>exitButton</td>
<td>Click</td>
</tr>
<tr>
<td>Fill the dataset with data</td>
<td>MainForm</td>
<td>Load</td>
</tr>
<tr>
<td>1. Compare the user’s answer to the correct answer</td>
<td>submitButton</td>
<td>Click</td>
</tr>
<tr>
<td>2. Keep track of the number of incorrect answers</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3. Display the next question and answers from the dataset</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4. Display the number of incorrect answers</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Display questions from the dataset</td>
<td>questionTextBox</td>
<td>None</td>
</tr>
<tr>
<td>Display answers from the dataset</td>
<td>aTextBox, bTextBox, cTextBox, dTextBox</td>
<td>None</td>
</tr>
<tr>
<td>Display the user choices</td>
<td>aRadioButton, bRadioButton, cRadioButton, dRadioButton</td>
<td>None</td>
</tr>
</tbody>
</table>

**Coding the Trivia Game Application**

Before you can begin coding the Trivia Game application, you need to open the Trivia Game Solution file.

**To open the Trivia Game solution file:**
1. Start Visual Studio. If necessary, close the Start Page window.
2. Open the **Trivia Game Solution** (Trivia Game Solution.sln) file, which is contained in the VbReloaded\Appendix H\Trivia Game Solution folder. The user interface shown earlier in Figure H.37 appears on the screen.

First, you will connect the Trivia database to the application, and then you will preview the data.

**To connect the Trivia database to the application, then preview the data:**
1. Click **Data** on the menu bar, and then click **Show Data Sources** to open the Data Sources window.
2. Click **Add New Data Source** in the Data Sources window to start the Data Source Configuration Wizard. See Figure H.39.
3. Click the **Next** button to display the Choose Your Data Connection screen.

4. Click the **New Connection** button to display the Add Connection dialog box. If Microsoft Access Database File (OLE DB) does not appear in the Data source box, click the **Change** button to open the Change Data Source dialog box, then click **Microsoft Access Database File**, and then click the **OK** button to return to the Add Connection dialog box.

5. Click the **Browse** button in the Add Connection dialog box. Open the \VbReloaded\Appendix H\Access Databases folder, then click **Trivia.mdb** in the list of filenames. Click the **Open** button. Figure H.40 shows the completed Add Connection dialog box.
6. Click the **Test Connection** button in the Add Connection dialog box. The “Test connection succeeded.” message appears in a dialog box. Click the **OK** button to close the dialog box.

7. Click the **OK** button to close the Add Connection dialog box. Trivia.mdb now appears in the Choose your data connection screen. Click the **Next** button. The computer displays the Local database file dialog box shown in Figure H.41.

![Figure H.41 Local database file dialog box](image1)

8. Click the **Yes** button to add the Trivia.mdb file to the current project. The Save the Connection String to the Application Configuration File screen shown in Figure H.42 appears next. If necessary, select the check box.

![Figure H.42 Save the Connection String to the Application Configuration File screen](image2)

9. Click the **Next** button to display the Choose Your Database Objects screen. Click the **plus box** that appears before Tables, then click the **plus box** that appears before Game1. Notice that the database contains one table (named Game1) and six fields. You can use this screen to select the table and/or field objects to include in the dataset.

10. In this application, you need to include all of the fields in the dataset. Click the **empty box** that appears before Game1. Doing this selects the table and field check boxes, as shown in Figure H.43.
11. Click the Finish button. The computer adds a dataset to the Data Sources window. Click the plus box that appears before Game1 in the Data Sources window. See Figure H.44.

12. Now preview the data contained in the dataset. Right-click the Data Sources window, then click Preview Data, and then click the Preview button in the Preview Data dialog box. See Figure H.45.
13. Click the **Close** button in the Preview Data dialog box, then save the solution.

Next, you will bind the field objects in the dataset to the appropriate text boxes on the form.

**To bind the field objects to the text boxes, then test the application:**

1. Click the **Question** field object in the Data Sources window, then drag the Question field object to the `questionTextBox`, as shown in Figure H.46.

2. Release the mouse button. The computer binds the Question field object to the `questionTextBox`. It also adds the TriviaDataSet, Game1BindingSource, and Game1TableAdapter objects to the component tray, as shown in Figure H.47.
3. Drag the AnswerA field object to the aTextBox, then drag the AnswerB field object to the bTextBox. Drag the AnswerC object to the cTextBox, then drag the AnswerD object to the dTextBox.

4. Save the solution, then start the application. The first record in the dataset appears in the interface, as shown in Figure H.48.
5. Click the Exit button to end the application. If necessary, close the Output window.
6. Close the Data Sources window.

Now you can begin coding the application. According to the TOE chart, shown earlier in Figure H.38, only three event procedures need to be coded: the Exit button’s Click event procedure, the MainForm’s Load event procedure, and the submitButton’s Click event procedure.

To begin coding the application:
1. Open the Code Editor window. Notice that the exitButton’s Click event procedure and the MainForm’s Load event procedure are already coded for you. The only procedure you need to code is the submitButton’s Click event procedure. [Recall that the computer automatically enters the Me.Game1TableAdapter.Fill (Me.TriviaDataSet.Game1) statement when you drag an object from the Data Sources window to the form.]
2. Replace the <your name> and <current date> text in the comments with your name and the current date.

Figure H.49 shows the pseudocode for the submitButton’s Click event procedure.

To finish coding the application:
1. Open the code template for the submitButton’s Click event procedure.
2. Type ‘ determines whether the user’s answer is correct and press Enter, then type ‘ keeps track of the number of incorrect answers and press Enter. Type ‘ displays the next record or the number of and press Enter, then type ‘ incorrect answers and press Enter twice.
3. Type dim recPtrPos as integer and press Enter, then type dim userAnswer as string = “” and press Enter. The recPtrPos variable will keep track of the position of the record pointer, and the userAnswer variable will store the user’s answer to the question displayed in the questionTextBox (either A, B, C, or D).
4. Type static numIncorrect as integer and press Enter twice. The numIncorrect variable will be used as a counter to keep track of the number of incorrect answers made by the user.
5. The first step in the pseudocode is to store the position of the record pointer in a variable. Recall that you can use the BindingSource object’s Position property to determine the current position of the record pointer in a dataset. Type ‘ store position of record pointer and press Enter, then type recptrpos = game1bindingsource.position and press Enter twice.
6. Now you will determine which radio button is selected, and assign its Text property (without the ampersand that designates the access key) to the userAnswer variable. Type the comments and Select Case statement shown in Figure H.50.
The next step is to compare the contents of the userAnswer variable to the contents of the CorrectAnswer field in the current record. If they are not the same, then you need to increase (by one) the contents of the numIncorrect counter variable. You can access the value stored in a field in the current record using the syntax `dataSetName.tableName(recordNumber).fieldname`.

7. Position the insertion point two lines below the `End Select` clause, then type `' if necessary, update number of incorrect answers` and press `Enter`. Then type `if useranswer <> _` and press `Enter`. Press `Tab`, then type `triviadataset.game1(recptrpos).correctanswer` and press `Enter`. Type `numincorrect = numincorrect + 1`.

The last step in the pseudocode is to determine whether to move to the next record or display a message box indicating the number of incorrect answers. If the record pointer is not pointing to the last record in the dataset, then the procedure should move the record pointer to the next record; doing this will display that record's question and answers. However, if the record pointer is pointing to the last record, it means that there are no more questions and answers to display. In that case, the procedure should display the number of incorrect answers made by the user.

8. Position the insertion point two lines below the `End If` clause, then type `' determine position of record pointer` and press `Enter`. Type `if recptrpos <= 7 then` and press `Enter`. Press `Tab`, then type `messagebox.show("Number incorrect: " & numincorrect.tostring, "Trivia Game", messageboxbuttons.ok, messageboxicon.information)`.

9. Save the solution. Figure H.51 shows the code for the Trivia Game application.

### Visual Basic code

```
' determine selected radio button, which
' represents the user’s answer
select case true
    case aradiobutton.checked
        useranswer = aradiobutton.text.replace("&", ")
    case bradiobutton.checked
        useranswer = bradiobutton.text.replace("&", ")
    case cradiobutton.checked
        useranswer = cradiobutton.text.replace("&", ")
    case dradiobutton.checked
        useranswer = dradiobutton.text.replace("&", ")
end select
```

(Figure is continued on next page)
Option Explicit On
Option Strict On

Public Class MainForm

    Private Sub exitButton_Click(ByVal sender As Object, ByVal e As System.EventArgs) Handles exitButton.Click
        Me.Close()
    End Sub

    Private Sub MainForm_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
        'TODO: This line of code loads data into the 'TriviaDataSet.Game1' table. You can move, 'or remove, it as needed.
        Me.Game1TableAdapter.Fill(Me.TriviaDataSet.Game1)
    End Sub

    Private Sub submitButton_Click(ByVal sender As Object, ByVal e As System.EventArgs) Handles submitButton.Click
        ' determines whether the user's answer is correct ' keeps track of the number of incorrect answers ' displays the next record or the number of ' incorrect answers

        Dim recPtrPos As Integer
        Dim userAnswer As String = ""
        Static numIncorrect As Integer

        ' store position of record pointer
        recPtrPos = Game1BindingSource.Position

        ' determine selected radio button, which ' represents the user’s answer
        Select Case True
            Case aRadioButton.Checked
                userAnswer = aRadioButton.Text.Replace("&", "")
            Case bRadioButton.Checked
                userAnswer = bRadioButton.Text.Replace("&", "")
            Case cRadioButton.Checked
                userAnswer = cRadioButton.Text.Replace("&", "")
            Case dRadioButton.Checked
                userAnswer = dRadioButton.Text.Replace("&", "")
        End Select

        ' if necessary, update number of incorrect answers
        If userAnswer <> _
            TriviaDataSet.Game1(recPtrPos).CorrectAnswer Then
            numIncorrect = numIncorrect + 1
        End If

    End Sub

(Figure is continued on next page)
PROGRAMMING EXAMPLE

Cartwright Industries Application

Carl Simons, the sales manager at Cartwright Industries, records the item number, name, and price of each product the company sells in a database named Items. The Items database is stored in the Items.mdb file contained in the VbReloaded\Appendix H\Access Databases folder. Mr. Simons wants an application that allows the sales clerks to view the numbers, names, and prices of the items in the database. The application also should allow the sales clerk to add, delete, and save changes to the database. Name the solution Cartwright Solution. Name the project Cartwright Project. Name the form file Main Form.vb. Save the application in the VbReloaded\Appendix H folder. In the Properties window, set the Items.mdb file’s Copy to Output Directory property to Copy if newer.

To test the application:
1. Start the application. When the first question appears on the screen, answer the question correctly by clicking the B radio button, and then clicking the Submit Answer button.
2. When the second question appears on the screen, answer the question incorrectly by clicking the D radio button, and then clicking the Submit Answer button.
3. Answer the remaining seven questions on your own. When you have submitted the answer for the last question, the submitButton’s Click event procedure displays the number of incorrect responses in a message box.
4. Click the OK button to close the message box.
5. Click the Exit button to close the application. If necessary, close the Output window.
6. Close the Code Editor window, then close the solution.

FIGURE H.51 Trivia Game application’s code

FIGURE H.52 Data contained in the tblItems table
TOE Chart:

<table>
<thead>
<tr>
<th>Task</th>
<th>Object</th>
<th>Event</th>
</tr>
</thead>
<tbody>
<tr>
<td>End the application</td>
<td>exitButton</td>
<td>Click</td>
</tr>
<tr>
<td>Display the dataset</td>
<td>TblItemsDataGridView</td>
<td>None</td>
</tr>
<tr>
<td>1. Move from record to record</td>
<td>TblItemsBindingNavigator</td>
<td>None</td>
</tr>
<tr>
<td>2. Add a record</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3. Delete a record</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4. Save changes to the dataset</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Fill the dataset with data</td>
<td>MainForm</td>
<td>Load</td>
</tr>
</tbody>
</table>

**Figure H.53**

User Interface (drag the table to the form):

**Figure H.54**
Objects, Properties, and Settings:

<table>
<thead>
<tr>
<th>Object</th>
<th>Property</th>
<th>Setting</th>
</tr>
</thead>
<tbody>
<tr>
<td>Form1</td>
<td>Name</td>
<td>MainForm</td>
</tr>
<tr>
<td></td>
<td>Font</td>
<td>Tahoma, 10 point</td>
</tr>
<tr>
<td></td>
<td>MaximizeBox</td>
<td>False</td>
</tr>
<tr>
<td></td>
<td>Size</td>
<td>547, 361</td>
</tr>
<tr>
<td></td>
<td>StartPosition</td>
<td>CenterScreen</td>
</tr>
<tr>
<td></td>
<td>Text</td>
<td>Cartwright Industries</td>
</tr>
<tr>
<td>Button1</td>
<td>Name</td>
<td>exitButton</td>
</tr>
<tr>
<td></td>
<td>Location</td>
<td>435, 289</td>
</tr>
<tr>
<td></td>
<td>Size</td>
<td>75, 26</td>
</tr>
<tr>
<td></td>
<td>Text</td>
<td>E&amp;xit</td>
</tr>
<tr>
<td>TblItemsDataGridView</td>
<td>AutoSizeColumns</td>
<td>Fill</td>
</tr>
<tr>
<td></td>
<td>Mode</td>
<td>12, 37</td>
</tr>
<tr>
<td></td>
<td>Location</td>
<td>515, 231</td>
</tr>
<tr>
<td></td>
<td>Size</td>
<td></td>
</tr>
</tbody>
</table>

**Figure H.55**

Code:

```vbscript
' Project name: Cartwright Project
' Project purpose: The project displays the data contained in a database. It also allows the user to add records, delete records, and save the changes made to records.
' Created/revised by: <your name> on <current date>

Option Explicit On
Option Strict On

Public Class MainForm

    Private Sub exitButton_Click(ByVal sender As Object, ByVal e As System.EventArgs) Handles exitButton.Click
        Me.Close()
    End Sub

    Private Sub TblItemsBindingNavigatorSaveItem_Click
        (ByVal sender As System.Object, ByVal e As System.EventArgs) Handles TblItemsBindingNavigatorSaveItem.Click
        Me.Validate()
        Me.TblItemsBindingSource.EndEdit()
        Me.TblItemsTableAdapter.Update(Me.ItemsDataSet.tblItems)
    End Sub

(Figure is continued on next page)```
Quick Review

- Companies and individuals use databases to organize information.
- You can use Visual Basic to access the data stored in many different types of databases.
- Databases created by Microsoft Access are relational databases. A relational database can contain one or more tables. Each table contains fields and records.
- Most tables contain a primary key that uniquely identifies each record.
- The data in a relational database can be displayed in any order, and you can control the amount of information you want to view.
- Visual Basic 2005 uses a technology called ADO.NET 2.0 to access the data stored in a database.
- The connection between a database and an application that uses ADO.NET 2.0 is only temporary.
- To access the data stored in a database, you first connect the database to an application. Doing this creates a dataset that contains objects, such as table objects and field objects.
- You can display the information contained in a dataset by binding one or more of the dataset objects to one or more controls in the application’s interface.
- A TableAdapter object connects a database to a DataSet object.
- A BindingSource object connects a DataSet object to the bound controls on a form.
- In most applications, the statement to fill a dataset with data is entered in the form’s Load event procedure.
- The BindingSource object uses an invisible record pointer to keep track of the current record.
- The location of the record pointer in a dataset is stored in the BindingSource object’s Position property.
- You can use the BindingSource object’s Move methods to move the record pointer in a dataset.
- You use a SQL SELECT statement to specify the fields and records to include in a dataset.
• You can use the Query Configuration Wizard to create queries.
• The Query Builder dialog box provides a convenient way to create a SELECT statement.
• You can associate a query with a ToolStrip control on a form.

**Key Terms**

A **database** is an organized collection of related information stored in a file on a disk.

A **relational database** is a database that stores information in tables, which are composed of columns (fields) and rows (records).

A **field** is a single item of information about a person, place, or thing.

A **record** is a group of related fields that contain all of the necessary data about a specific person, place, or thing.

A **table** is a group of related records.

A **primary key** is the field that uniquely identifies each record in a table.

A **parent table** is a table to which another table, called a **child table**, links. The tables are linked using a common field. The common field is called the primary key in the parent table. It is called the **foreign key** in the child table.

**ADO.NET 2.0** refers to the technology used in Visual Basic 2005 to connect an application to a database.

A **dataset** is a copy of some or all of the records and fields contained in a database. A dataset is stored in the computer’s internal memory.

**Binding** refers to the process of connecting a dataset object to a control on a form.

Controls connected to a dataset object are called **bound controls**.

You can use a **DataGridView control** to display the data contained in a dataset. The control displays the data in a row and columnar format.

You can use the **BindingNavigator control** to move from one record to another in the dataset, as well as to add, delete, and save records.

A **DataSet object** stores the information contained in a dataset.

A **TableAdapter object** connects a database to a DataSet object.

A **BindingSource object** connects a DataSet object to the bound controls on the form.

The **BindingSource object’s EndEdit method** applies any pending changes (such as new records, deleted records, or changed records) to the dataset.

The **TableAdapter object’s Update method** commits, to the database, the changes made to the dataset.

You can use a TableAdapter object’s **Fill method** to load data into a table contained in a dataset.

The **BindingSource object stores the position of the record pointer in its Position property**.

**XML**, which stands for **Extensible Markup Language**, is a text-based language used to store and share data between applications and across networks and the Internet.
An **XML schema definition file** defines the tables and fields that make up the dataset.

A **query** specifies the fields and records to retrieve from a database, as well as the order in which to arrange the fields and records.

**SQL** stands for **Structured Query Language**, and is a set of commands that allows you to access and manipulate the data stored in databases.

The **SELECT statement** is the most commonly used SQL command. It allows you to specify the fields and records you want to access from a database, as well as control the order in which the fields and records appear when displayed.

You use the **WHERE clause** in a SELECT statement to limit the records that will be selected.

You use the **ORDER BY clause** in a SELECT statement to control the order in which the records are arranged.

---

**Self-Check Questions and Answers**

1. The field that links a child table to a parent table is called the _____.
   a. foreign key in the child table
   b. foreign key in the parent table
   c. link key in the parent table
   d. primary key in the child table

2. When using ADO.NET 2.0 to access the information contained in a database, the computer stores a copy of the information in a _____.
   a. databaseSet in internal memory
   b. dataset in internal memory
   c. dataset on the computer’s hard drive
   d. recordset on the computer’s hard drive

3. The process of connecting a control to a dataset is called _____.
   a. assigning
   b. attaching
   c. joining
   d. None of the above.

4. The ____ object connects a database to a DataSet object.
   a. BindingSource
   b. DataBase
   c. DataGridView
   d. TableAdapter

5. The ____ property stores an integer that represents the location of the record pointer in a dataset.
   a. BindingNavigator object’s Position
   b. BindingSource object’s Position
   c. TableAdapter object’s Position
   d. None of the above.
6. If the record pointer is positioned on record number five in a dataset, which of the following methods can be used to move the record pointer to record number four?
   a. GoPrevious()
   b. Move(4)
   c. MovePrevious()
   d. PositionPrevious

7. SQL stands for ______.
   a. Select Query Language
   b. Semi-Quick Language
   c. Structured Quick Language
   d. Structured Query Language

8. Which of the following will select the FName, MName, and LName fields from a table named tblNames?
   a. SELECT FName, MName, AND LName FROM tblNames
   b. SELECT FName, MName, OR LName FROM tblNames
   c. SELECT FName, MName, LName FROM tblNames
   d. None of the above.

9. Which of the following will arrange the SocialNum field data in descending order? The field is contained in a table named PensionInfo.
   a. SELECT SocialNum FROM PensionInfo DESC
   b. SELECT SocialNum FROM PensionInfo ORDER BY SocialNum DESC
   c. SELECT SocialNum FROM PensionInfo WHERE SocialNum DESC
   d. None of the above.

10. Which of the following will select only records that have the letter A in their Status field? The field is contained in the Worker table.
    a. SELECT Id, EmpName, Status FROM Worker WHERE Status = 'A'
    b. SELECT Id, EmpName, Status FROM Worker ORDER BY Status = 'A'
    c. SELECT Id, EmpName, Status FROM Worker FOR Status = "A"
    d. None of the above.

Answers: 1) a, 2) b, 3) d, 4) d, 5) b, 6) c, 7) d, 8) c, 9) b, 10) a

---

**Review Questions**

1. A(n) ______ is an organized collection of related information stored in a file on a disk.
   a. database
   b. field
   c. object
   d. record

2. A ______ database is one that stores information in tables.
   a. columnar
   b. relational
   c. sorted
   d. tabular
3. A group of related records in a database is called a _____.
   a. column
   b. field
   c. row
   d. table

4. Which of the following statements is true about a relational database?
   a. Data stored in a relational database can be retrieved both quickly and easily by the computer.
   b. Data stored in a relational database can be displayed in any order.
   c. A relational database stores data in a column and row format.
   d. All of the above are true.

5. The _____ object provides the connection between a DataSet object and a control.
   a. Binding
   b. BindingSource
   c. Connecting
   d. None of the above.

6. An application contains DataSet, BindingSource, TableAdapter, and BindingNavigator objects named FriendsDataSet, TblNamesBindingSource, TblNamesTableAdapter, and TblNamesBindingNavigator, respectively. Which of the following statements retrieves data from the Friends database and stores it in the FriendsDataSet?
   a. Me.FriendsDataSet.Fill(Friends.mdb)
   b. Me.TblNamesBindingSource.Fill(Me.FriendsDataSet)
   c. Me.TblNamesBindingNavigator.Fill(Me.FriendsDataSet.tblNames)
   d. Me.TblNamesTableAdapter.Fill(Me.FriendsDataSet.tblNames)

Use the following database table, named tblState, to answer Questions 7 and 8.

<table>
<thead>
<tr>
<th>Field name</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>State</td>
<td>Text</td>
</tr>
<tr>
<td>Capital</td>
<td>Text</td>
</tr>
<tr>
<td>Population</td>
<td>Numeric</td>
</tr>
</tbody>
</table>

7. Which of the following statements allows you to view all of the records in the table?
   a. SELECT ALL records FROM tblState
   b. SELECT State, Capital, Population FROM tblState
   c. VIEW ALL records FROM tblState
   d. VIEW State, Capital, Population FROM tblState

8. Which of the following statements will retrieve all records having a population that exceeds 5,000,000?
   a. SELECT ALL FROM tblState FOR Population > 5000000
   b. SELECT State, Capital, Population FROM tblState WHERE Population > "5000000"
   c. SELECT State, Capital, Population FROM tblState WHERE Population > '5000000'
   d. None of the above.
9. A field that uniquely identifies each record in a table is called a _____.
   a. foreign field
   b. foreign key
   c. primary field
   d. primary key

10. In a SELECT statement, the _____ clause is used to limit the records that will be selected.
    a. LIMIT
    b. ORDER BY
    c. SET
    d. None of the above.

11. Controls connected to a DataSet object are called _____ controls.
    a. bound
    b. connected
    c. data
    d. None of the above.

12. You can use the _____ dialog box to build a SELECT statement.
    a. Query Builder
    b. Select Builder
    c. SQL Builder
    d. SQL Helper

13. If the current record is the second record in the dataset, which of the following statements will position the record pointer on the first record?
    a. TblEmployBindingSource.Position = 0
    b. TblEmployBindingSource.Position = TblEmployBindingSource.Position - 1
    c. TblEmployBindingSource.MoveFirst()
    d. All of the above.

14. Which of the following tells the TblWorkerTableAdapter object to load the WorkersDataSet object with data?
    a. TblWorkerTableAdapter.Fill(WorkersDataSet.tblWorker)
    b. TblWorkerTableAdapter.Load(tblWorker.WorkersDataSet)
    c. TblWorkerTableAdapter.FillInto(tblWorker)
    d. None of the above.

15. The SQL SELECT statement is case sensitive.
    a. True
    b. False

**Review Exercises – Short Answer**

Use the following database table to complete Review Exercises 1 through 6. The table is named tblMagInfo.

<table>
<thead>
<tr>
<th>Field name</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code</td>
<td>Character</td>
</tr>
<tr>
<td>Magazine</td>
<td>Text</td>
</tr>
<tr>
<td>Cost</td>
<td>Numeric</td>
</tr>
</tbody>
</table>

1. Write a SELECT statement that will select the Code and Magazine fields.

2. Write a SELECT statement that will select the Code and Cost fields. Arrange the records in descending order by the Cost field.
3. Write a SELECT statement that will select all of the fields, but only for records having a code of 9.

4. Write a SELECT statement that will select all of the fields, but only for magazines having a cost of $3 or more.

5. Write a SELECT statement that will select the Magazine and Cost fields, but only for the Daily Food Guide magazine.

6. Write a SELECT statement that will select the Magazine and Cost fields, but only for magazines whose names begin with the letter G.

7. Write the statement to assign the location of the record pointer to an Integer variable named recNum.

8. Write the statement to move the record pointer to the last record in the dataset.

9. Explain the purpose of each of the following objects: DataSet, TableAdapter, BindingSource.

10. How do you remove the Delete button from a BindingNavigator control?

---

**Computer Exercises**

1. In this exercise, you create an application that accesses the data stored in a database. The application displays the data in a DataGridView control.
   a. Open the Morgan Industries Solution (Morgan Industries Solution.sln) file, which is contained in the VbReloaded\Appendix H\Morgan Industries Solution-Ex1 folder.
   b. Complete the interface, using Figure H.13 as a guide. In the Properties window, set the Employees.mdb file’s Copy to Output Directory property to Copy if newer. Doing this allows you to save the changes made to the dataset.
   c. Save the solution, then start the application. The interface should appear as shown in Figure H.16. Verify that the BindingNavigator control works appropriately. (Do not try to save an empty record.)
   d. Click the Exit button to end the application, then close the solution.

2. In this exercise, you create an application that accesses the data stored in a database. The application displays the data in label controls.
   a. Open the Morgan Industries Solution (Morgan Industries Solution.sln) file, which is contained in the VbReloaded\Appendix H\Morgan Industries Solution-Ex2 folder.
   b. Complete the interface, using Figure H.18 as a guide. Also be sure to code the Next and Previous buttons. (The code is shown in Figure H.23.)
   c. Save the solution, then start the application. The interface should appear as shown in Figure H.20. Test the Next and Previous buttons to verify that they are working correctly.
   d. Click the Exit button to end the application, then close the solution.

3. In this exercise, you modify the application from Computer Exercise 1. The modified application will allow the user to display the full-time records, part-time records, and all of the records.
   a. Use Windows to make a copy of the Morgan Industries Solution-Ex1 folder, which is contained in the VbReloaded\Appendix H folder. Rename the copy Morgan Industries Solution-Ex3.
b. Open the Morgan Industries Solution (Morgan Industries Solution.sln) file contained in the VbReloaded\Appendix H\Morgan Industries Solution-Ex3 folder.

c. Add the FillByFulltime query to the TblEmployTableAdapter object. (You can use Figures H.28 through H.31 as a guide.)

d. Associate the FillByFulltime query with a ToolStrip control. (You can use Figures H.33 through H.35 as a guide.)

e. Save the solution, then start the application. Test the FillByFulltime button on the ToolStrip control. The interface should appear as shown in Figure H.36.

f. Click the Exit button to end the application.

g. Create another query. The query should select only the records for part-time employees. Arrange the records in order by the Rate field. Use FillByParttime as the name for the query’s Fill method, and use GetDataByParttime as the name for its GetData method.

h. Associate the FillByParttime query with a ToolStrip control. Also associate the Fill query with a ToolStrip control.

i. Save the solution, then start the application. Test the FillByFulltime, FillByParttime, and Fill buttons on the ToolStrip controls.

j. Click the Exit button to end the application.

4. In this exercise, you modify the application from the chapter’s Programming Example.

a. Create the Cartwright Industries application shown in the chapter’s Programming Example. Save the application in the VbReloaded\Appendix H folder.

b. Create a query that allows you to display the data in ascending order by the item number. Also create a query that allows you to display the data in descending order by the price.

c. Associate the two queries with two ToolStrip controls.

d. Save the solution, then start and test the application.

e. Click the Exit button to end the application, then close the solution.

5. In this exercise, you modify the application you created in the chapter’s Programming Tutorial.

a. Use Windows to make a copy of the Trivia Game Solution folder, which is contained in the VbReloaded\Appendix H folder. Rename the copy Modified Trivia Game Solution.

b. Open the Trivia Game Solution (Trivia Game Solution.sln) file contained in the VbReloaded\Appendix H\Modified Trivia Game Solution folder.

c. Display the question number (from 1 through 9) along with the word “Question” in the interface.

d. Add another button to the interface. The button should allow the user to start a new game. Allow the user to click the New Game button only after he or she has answered all nine questions.

e. Save the solution, and then start and test the application.

f. End the application, then close the solution.

6. In this exercise, you modify the application you created in the chapter’s Programming Tutorial.

a. Use Windows to make a copy of the Trivia Game Solution folder, which is contained in the VbReloaded\Appendix H folder. Rename the copy Disc Trivia Game Solution.

b. Open the Trivia Game Solution (Trivia Game Solution.sln) file contained in the VbReloaded\Appendix H\Disc Trivia Game Solution folder.
c. Allow the user to answer the questions in any order, and also to change his or her answers. (You will probably need to modify the interface by including additional buttons.) Only display the number of incorrect answers when the user requests that information.
d. Save the solution, and then start and test the application.
e. End the application, and then close the solution.

7. In this exercise, you find and correct an error in an application. The process of finding and correcting errors is called debugging.
a. Open the Debug Solution (Debug Solution.sln) file, which is contained in the VbReloaded\Appendix H\Debug Solution folder.
b. Open the Code Editor window. Review the existing code.
c. Notice that a jagged line appears below one of the lines of code in the Code Editor window. Correct the code to remove the jagged line.
d. Save the solution, and then start and test the application. Notice that the application is not working correctly. Click the Exit button to end the application.
e. Correct the errors in the application’s code, and then save the solution and start the application. Test the application.
f. Click the Exit button to end the application, and then close the solution.

Case Projects

Addison Playhouse

In this Case Project, you use a Microsoft Access database named Play. The Play.mdb database file is located in the VbReloaded\Appendix H\Access Databases folder. The Play database contains one table named tblReservations. The table contains 20 records, each having three fields: a numeric field named Seat and two text fields named Patron and Phone. Create an application that allows the user to display the contents of the dataset, and also to add, delete, and save records. In the Properties window, set the Play.mdb file’s Copy to Output Directory property to Copy if newer.

College Courses

In this Case Project, you use a Microsoft Access database named Courses. The Courses.mdb database file is located in the VbReloaded\Appendix H\Access Databases folder. The Courses database contains one table named tblCourses. The table contains 10 records, each having four fields: course ID, course title, credit hours, and grade. The credit hours field is numeric; the other fields contain text. Create an application that uses six queries. Five of the queries should allow you to display the data for a specific grade (A, B, C, D, F). The sixth query should display all of the records in their original order.

Sports Action

In this Case Project, you use a Microsoft Access database named Sports. The Sports.mdb database file is located in the VbReloaded\Appendix H\Access Databases folder. The Sports database contains one table named tblScores. The table contains five records, each having four fields that store the following information: name of the opposing team, the date of the game, your favorite team’s score, and the opposing team’s score. Create an application that allows you to view each record, and also to add, delete, and save records. In the Properties window, set the Sports.mdb file’s Copy to Output Directory property to Copy if newer.
The Fiction Bookstore

Jerry Schmidt, the manager of the Fiction Bookstore, uses a Microsoft Access database named Books to keep track of the books in his store. The Books.mdb database file is contained in the VbReloaded\Appendix H\Access Databases folder. The database has one table named tblBooks. The table has three fields: a numeric field named BookNumber and two text fields named Title and Author. Mr. Schmidt wants an application that he can use to enter an author’s name, and then display only the titles of books written by the author. Display the information in a DataGridView control. (Hint: In this application, you need to allow the user to specify the records he or she wants to select while the application is running. You can use a parameterized query to accomplish this task. A parameterized query is simply a SELECT statement that contains fieldname = ? in the WHERE clause. The question mark is a placeholder for the missing data. For example, you can create a parameterized query using Author = ? in the WHERE clause. Associate the query with a ToolStrip control in the interface.)