In this chapter you will:

- Study databases
- Connect to databases with ASP.NET
- Execute SQL commands through ASP.NET

A common use of Web pages is to gather information that is stored in a database on a Web server. Most server-side scripting languages, including ASP.NET have the ability to create Web pages that can read and write data to and from databases. In this chapter, you will learn how to add database connectivity to your Web pages with ASP.NET.
Your goal in this chapter is to learn how to use ASP.NET to read, write, and modify database information. To accomplish this goal, it helps to first understand how databases work. Formally defined, a **database** is an ordered collection of information from which a computer program can quickly access information. You can probably think of many databases which you work with in your everyday life. For example, your address book is a database. So is the card file containing recipes in a kitchen. Other examples of databases include a company’s employee directory and a file cabinet containing client information. Essentially, any information that can be organized into ordered sets of data, then quickly retrieved, can be considered a database. A collection of hundreds of baseball cards thrown into a shoebox is not a database, because an individual card cannot be quickly or easily retrieved (except by luck). However, if the baseball card collection was organized in binders by team, and then further organized according to each player’s field position or batting average, then it could be considered a database because you could quickly locate a specific card.

The information stored in computer databases is actually stored in tables similar to spreadsheets. Each row in a database table is called a record. A **record** in a database is a single complete set of related information. Each recipe in a recipe database, for instance, is a single database record. Each column in a database table is called a field. **Fields** are the individual categories of information stored in a record. Examples of fields that might exist in a recipe database include ingredients, cooking time, cooking temperature, and so on.

To summarize, you can think of databases as consisting of tables, which consist of records, which consist of fields. Figure 12–1 shows an example of an employee directory for programmers at an application development company. The database consists of five records, one for each employee. Each record consists of five fields: last_name, first_name, address, city, state, and zip.

<table>
<thead>
<tr>
<th>last_name</th>
<th>first_name</th>
<th>address</th>
<th>city</th>
<th>state</th>
<th>zip</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blair</td>
<td>Dennis</td>
<td>204 Spruce Lane</td>
<td>Brookfield</td>
<td>MA</td>
<td>01506</td>
</tr>
<tr>
<td>Hernandez</td>
<td>Louis</td>
<td>68 Boston Post Road</td>
<td>Spencer</td>
<td>MA</td>
<td>01562</td>
</tr>
<tr>
<td>Miller</td>
<td>Erica</td>
<td>271 Baker Hill Road</td>
<td>Brookfield</td>
<td>MA</td>
<td>01515</td>
</tr>
<tr>
<td>Morinaga</td>
<td>Scott</td>
<td>17 Ashley Road</td>
<td>Brookfield</td>
<td>MA</td>
<td>01515</td>
</tr>
<tr>
<td>Picard</td>
<td>Raymond</td>
<td>1113 Oakham Road</td>
<td>Barre</td>
<td>MA</td>
<td>01531</td>
</tr>
</tbody>
</table>

**Figure 12-1** Employee directory database

The database in Figure 12-1 is an example of a flat-file database, one of the simplest types of databases. A **flat-file database** stores information in a single table. For simple
collections of information, flat-file databases are usually adequate. With large and complex collections of information, flat-file databases can become unwieldy. A better solution for large and complex databases is a relational database. A relational database stores information across multiple related tables. Although you will not actually work with a relational database in this chapter, understanding how they work is helpful because relational databases are among the most common in use today.

Two other types of database systems you may encounter are hierarchical databases and network databases.

Relational databases consist of one or more related tables. In fact, large relational databases can consist of dozens or hundreds of related tables. Although relational databases may consist of many tables, you create relationships within the database by working with two tables at a time. One table in a relationship is always considered to be the primary table, whereas the other table is considered to be the related table. A primary table is the main table in a relationship that is referenced by another table. A related table (also called a child table) references a primary table in a relational database. Tables in a relationship are connected using primary and foreign keys. A primary key is a field that contains a unique identifier for each record in a primary table. A foreign key is a field in a related table that refers to the primary key in a primary table. Primary and foreign keys link records across multiple tables in a relational database.

There are three basic types of relationships within a relational database: one-to-one, one-to-many, and many-to-many. A one-to-one relationship exists between two tables when a related table contains exactly one record for each record in the primary table. You create one-to-one relationships when you want to break information into multiple, logical sets. It is important to understand that information in the tables in a one-to-one relationship can usually be placed within a single table. However, you may want to break the information into multiple tables to better organize the information into logical sets. Another reason for using one-to-one relationships is that they allow you to make the information in one of the tables confidential and accessible only by certain individuals. For example, you might want to create a personnel table that contains basic information about an employee, similar to the information in the table in Figure 12-1. Yet, you might also want to create a payroll table that contains confidential information about each employee’s salary, benefits, and other types of compensation, and that can be accessed only by the Human Resources and Accounting departments. Figure 12-2 shows two tables, Employees and Payroll, with a one-to-one relationship. The primary table is the employee information table from Figure 12-1. The related table is a payroll table that contains confidential salary and compensation information. Notice that each table contains an identical number of records; one record in the primary table corresponds to one record in the related table. The relationship is achieved by adding a primary key to the Employees table and a foreign key to the Payroll table.
A one-to-many relationship exists in a relational database when one record in a primary table has many related records in a related table. You create a one-to-many relationship in order to eliminate redundant information in a single table. Primary and foreign keys are the only pieces of information in a relational database table that should be duplicated. Breaking tables into multiple related tables in order to reduce redundant and duplicate information is called normalization. The elimination of redundant information (normalization) reduces the size of a database and makes the data easier to work with. For example, consider the table in Figure 12-3. The table lists every programming language in which the programmer is proficient. Notice that each programmer’s name is repeated for each programming language with which he or she is familiar. This repetition is an example of redundant information that can occur in a single table.

A one-to-many relationship provides a more efficient and less redundant method of storing this information in a database. Figure 12-4 shows the same information organized into a one-to-many relationship.

In some databases, the table containing multiple records for one entity (for example, the programming language table in Figure 12-4) is the primary table. In these cases, the relationship is often referred to as a many-to-one relationship.
Figure 12-3  Table with redundant information

**Employees table ("one" side)**

<table>
<thead>
<tr>
<th>employee_id</th>
<th>last_name</th>
<th>first_name</th>
<th>address</th>
<th>city</th>
<th>state</th>
<th>zip</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>Blair</td>
<td>Dennis</td>
<td>204 Spruce Lane</td>
<td>Brookfield</td>
<td>MA</td>
<td>01506</td>
</tr>
<tr>
<td>102</td>
<td>Hernandez</td>
<td>Louis</td>
<td>68 Boston Post Road</td>
<td>Spencer</td>
<td>MA</td>
<td>01562</td>
</tr>
<tr>
<td>103</td>
<td>Miller</td>
<td>Erica</td>
<td>271 Baker Hill Road</td>
<td>Brookfield</td>
<td>MA</td>
<td>01515</td>
</tr>
<tr>
<td>104</td>
<td>Morinaga</td>
<td>Scott</td>
<td>17 Ashley Road</td>
<td>Brookfield</td>
<td>MA</td>
<td>01515</td>
</tr>
<tr>
<td>105</td>
<td>Picard</td>
<td>Raymond</td>
<td>1113 Oakham Road</td>
<td>Barre</td>
<td>MA</td>
<td>01531</td>
</tr>
</tbody>
</table>

**Languages table ("many" side)**

<table>
<thead>
<tr>
<th>employee_id</th>
<th>language</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>JavaScript</td>
</tr>
<tr>
<td>101</td>
<td>ASPNET</td>
</tr>
<tr>
<td>102</td>
<td>JavaScript</td>
</tr>
<tr>
<td>102</td>
<td>ASPNET</td>
</tr>
<tr>
<td>102</td>
<td>Java</td>
</tr>
<tr>
<td>103</td>
<td>JavaScript</td>
</tr>
<tr>
<td>103</td>
<td>ASPNET</td>
</tr>
<tr>
<td>103</td>
<td>Java</td>
</tr>
<tr>
<td>103</td>
<td>C++</td>
</tr>
<tr>
<td>104</td>
<td>JavaScript</td>
</tr>
<tr>
<td>104</td>
<td>ASPNET</td>
</tr>
<tr>
<td>104</td>
<td>Java</td>
</tr>
<tr>
<td>105</td>
<td>JavaScript</td>
</tr>
<tr>
<td>105</td>
<td>ASPNET</td>
</tr>
</tbody>
</table>

One record in the top table is linked to many records in the bottom table

Figure 12-4  One-to-many relationship
Although Figure 12-4 is an example of a one-to-many relationship, the tables are not normalized because the language field contains duplicate values. Recall that primary and foreign keys are the only pieces of information in a relational database that should be duplicated. To further reduce repetition, you could organize the Languages table in Figure 12-4 into another one-to-many relationship. However, a better choice is to create a many-to-many relationship. A many-to-many relationship exists in a relational database when many records in one table are related to many records in another table.

Consider the relationship between programmers and programming languages. Each programmer can work with many programming languages, and each programming language can be used by many programmers. To create a many-to-many relationship, you must use a junction table because most relational database systems cannot work directly with many-to-many relationships. A junction table creates a one-to-many relationship for each of the two tables in a many-to-many relationship. A junction table contains foreign keys from the two tables in a many-to-many relationship, along with any other fields that correspond to a many-to-many relationship. Figure 12-5 contains an example of a many-to-many relationship between the Employees table and a Languages table. The Employees table contains a primary key named employee_id, and the Languages table contains a primary key named language_id. A junction table named Experience contains two foreign keys, one corresponding to the employee_id primary key in the Employees table, and one corresponding to the language_id primary key in the Languages table. The Experience junction table also contains a field named years. You add records to the Experience junction table to build a list of the years that each programmer has been working with a particular programming language.

Database Management Systems

With a grasp of basic database design, you can now begin to consider how to create and manipulate databases. An application or collection of applications used to create, access, and manage a database is called a database management system, or DBMS. Database management systems run on many different platforms, ranging from personal computers, to client-server systems, to mainframes. Different database management systems exist for different types of database formats. A database management system that stores data in a flat-file format is called a flat-file database management system. A database management system that stores data in a relational format is called a relational database management system, or RDBMS. Other types of database management systems include hierarchical and network database management systems. Some of the more popular relational database management systems you may have heard of include Oracle, Sybase, and Informix for high-end computers such as UNIX systems, and DB2 for mainframes, and Access, FoxPro, and Paradox for PCs.
Database management systems perform many of the same functions as other types of applications with which you might have worked, such as word-processing and spreadsheet programs. For example, database management systems create new database files and contain interfaces that allow users to enter and manipulate data. One of the most important functions of a database management system is the structuring and preservation of the database file’s structure. Additionally, a database management system must ensure that data is stored correctly in a database’s tables, regardless of the database format (flat-file, relational, hierarchical, or network). In relational databases, the database management system ensures that the appropriate information is entered according to the relationship structure in the database tables. Many DBMS systems also have security features that can be used to restrict user access to specific types of data.

Figure 12-5  Many-to-many relationship

<table>
<thead>
<tr>
<th>employee_id</th>
<th>last_name</th>
<th>first_name</th>
<th>address</th>
<th>city</th>
<th>state</th>
<th>zip</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>Blair</td>
<td>Dennis</td>
<td>204 Spruce Lane</td>
<td>Brookfield</td>
<td>MA</td>
<td>01506</td>
</tr>
<tr>
<td>102</td>
<td>Hernandez</td>
<td>Louis</td>
<td>68 Boston Post Road</td>
<td>Spencer</td>
<td>MA</td>
<td>01562</td>
</tr>
<tr>
<td>103</td>
<td>Miller</td>
<td>Erica</td>
<td>271 Baker Hill Road</td>
<td>Brookfield</td>
<td>MA</td>
<td>01515</td>
</tr>
<tr>
<td>104</td>
<td>Morinaga</td>
<td>Scott</td>
<td>17 Ashley Road</td>
<td>Brookfield</td>
<td>MA</td>
<td>01515</td>
</tr>
<tr>
<td>105</td>
<td>Picard</td>
<td>Raymond</td>
<td>1113 Oakham Road</td>
<td>Barre</td>
<td>MA</td>
<td>01531</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>language_id</th>
<th>language</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>JavaScript</td>
</tr>
<tr>
<td>11</td>
<td>ASP.NET</td>
</tr>
<tr>
<td>12</td>
<td>Java</td>
</tr>
<tr>
<td>13</td>
<td>C++</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>employee_id</th>
<th>language_id</th>
<th>years</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>10</td>
<td>5</td>
</tr>
<tr>
<td>101</td>
<td>11</td>
<td>4</td>
</tr>
<tr>
<td>102</td>
<td>10</td>
<td>3</td>
</tr>
<tr>
<td>102</td>
<td>11</td>
<td>2</td>
</tr>
<tr>
<td>102</td>
<td>12</td>
<td>3</td>
</tr>
<tr>
<td>103</td>
<td>10</td>
<td>2</td>
</tr>
<tr>
<td>103</td>
<td>11</td>
<td>3</td>
</tr>
<tr>
<td>103</td>
<td>12</td>
<td>6</td>
</tr>
<tr>
<td>103</td>
<td>13</td>
<td>3</td>
</tr>
<tr>
<td>104</td>
<td>10</td>
<td>7</td>
</tr>
<tr>
<td>104</td>
<td>11</td>
<td>5</td>
</tr>
<tr>
<td>104</td>
<td>12</td>
<td>8</td>
</tr>
<tr>
<td>105</td>
<td>10</td>
<td>4</td>
</tr>
<tr>
<td>105</td>
<td>11</td>
<td>2</td>
</tr>
</tbody>
</table>
Two other important aspects of database management systems are their querying and reporting capabilities. A **query** is a structured set of instructions and criteria for retrieving, adding, modifying, and deleting database information. A **report** is the formatted, printed output of a database table or the results of a query. Most database management systems use a **data manipulation language**, or **DML**, for creating queries. Different database management systems support different data manipulation languages. However, **structured query language**, or **SQL** (pronounced *sequel*), has become somewhat of a standard data manipulation language among many database management systems.

Many database management systems make it easier for users to create queries by hiding the data manipulation language behind a user interface. Figure 12-6 shows an example of Access’s query design interface. Users can create queries by dragging fields from the table objects in the upper portion of the screen to the criteria grid in the bottom portion of the screen. Behind the scenes, Access creates the SQL code shown in Figure 12-7. SQL is Access’s data manipulation language.

![Access SQL code](image)

**Figure 12-6**  Access query design screen

![Access SQL code](image)

**Figure 12-7**  Access SQL code
Although working with an interface to design queries is fine for end users, to programmatically manipulate the data in a database, you must learn the database management system’s data manipulation language. For example, when accessing databases with ASP.NET, you must use a data manipulation language. Because SQL is the underlying data manipulation language for many database management systems, you will learn more about SQL later in this chapter so that you can better understand how ASP.NET communicates with database management systems.

A great way to quickly write—and learn—SQL code is to use the Access query design window to build and test your queries. You can then copy and paste the SQL string generated by the Access query design window into your program code.

Many database management systems also use a data definition language, or DDL, for creating databases, tables, fields, and other components of a database.

It is important to understand that even though many database management systems support the same database formats (flat-file, relational, hierarchical, or network), each database management system is an individual application that creates its own proprietary file types. For example, even though Access and Paradox are both relational database management systems, Access creates its database files in a proprietary format with an extension of .mdb, whereas Paradox creates its database files in a proprietary format with an extension of .db. Although both Paradox and Access contain filters that allow you to import each other’s file formats, the database files are not completely interchangeable between the two programs. The same is true for most database management systems; they can import each other’s file formats, but they cannot directly read each other’s files.

In today’s environment, it is often necessary for an application to access multiple databases created in different database management systems. For example, a company may need an ASP.NET application that simultaneously accesses a large legacy database written in dBase and a newer database written in Oracle. Converting the large dBase database to Oracle would be cost prohibitive. On the other hand, the company cannot continue using the older dBase database because its needs have grown beyond the older database’s capabilities. Still, the company must be able to access the data in both systems.

To allow easy access to data in various database formats, Microsoft established the open database connectivity standard. Open database connectivity, or ODBC, allows ODBC-compliant applications to access any data source for which there is an ODBC driver. ODBC uses SQL commands (known as ODBC SQL) to allow an ODBC-compliant application to access a database. Essentially, an ODBC application connects to a database for which there is an ODBC driver and then executes ODBC SQL commands. Then the ODBC driver translates the SQL commands into a format that the database can understand.
Structured Query Language

Programmers at IBM invented SQL in the 1970s as a way of querying databases for specific criteria. Since then, SQL has been adopted by numerous database management systems running on mainframes, minicomputers, and PCs. In 1986 the American National Standards Institute (ANSI) approved an official standard for the SQL language. In 1991, The X/Open and SQL Access Group created a standardized version of SQL known as the Common Applications Environment (CAE) SQL draft specification. Even with two major standards available, however, most database management systems use their own version of the SQL language. ODBC SQL corresponds to the X/Open and SQL Access Group’s CAE SQL draft specification. Therefore, an ODBC driver for a specific database management system must support ODBC SQL.

If you ever work directly with an individual database management system, keep in mind that the ODBC SQL you learn in this chapter may not correspond directly to that database management system's version of SQL.

SQL uses fairly easy-to-understand statements to execute database commands. SQL statements are composed of keywords that perform actions on a database. Table 12-1 lists several SQL keywords that are common to most versions of SQL.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DELETE</td>
<td>Deletes a row from a table</td>
</tr>
<tr>
<td>FROM</td>
<td>Specifies the tables from which to retrieve or delete records</td>
</tr>
<tr>
<td>INSERT</td>
<td>Inserts a new row into a table</td>
</tr>
<tr>
<td>INTO</td>
<td>Determines the table into which records should be inserted</td>
</tr>
<tr>
<td>ORDER BY</td>
<td>Sorts the records returned from a table</td>
</tr>
<tr>
<td>SELECT</td>
<td>Returns information from a table</td>
</tr>
<tr>
<td>UPDATE</td>
<td>Saves changes to fields in a record</td>
</tr>
<tr>
<td>WHERE</td>
<td>Specifies the conditions that must be met for records to be returned from a query</td>
</tr>
</tbody>
</table>

The simple SQL statement `SELECT * FROM Employees` selects all records (using the asterisk * wildcard) from the Employees table. The following code shows a more complex SQL statement that selects the last_name and first_name fields from the Employees table if the record’s city field is equal to “Spencer”. The results are then sorted by the last_name and first_name fields using the ORDER BY keyword.

```
SELECT last_name, first_name FROM Employees
WHERE city = "Spencer" ORDER BY last_name, first_name
```
SQL table or field names that include spaces are enclosed in brackets. For example, if the `last_name` and `first_name` field names in the preceding code included spaces instead of underscore characters, you would write the statement as follows:

```sql
SELECT [last name], [first name] FROM [Employees]
WHERE [city] = "Spencer" ORDER BY [last name], [first name]
```

Not all database management systems allow spaces in SQL table or field names. For this reason, many programmers prefer not to include spaces in SQL table or field names in order to make their databases and SQL code compatible with database management systems that do not allow spaces.

You will use several of the basic ODBC SQL keywords in this chapter. For in-depth information on ODBC SQL, visit the Microsoft Developer’s Network at http://msdn.microsoft.com.

**Connecting to Databases with ASP.NET**

With Active Server Pages, you use ActiveX Data Objects to access databases. ActiveX Data Objects, or ADO, is a Microsoft database connectivity technology that allows ASP and other Web development tools to access ODBC- and OLE DB-compliant databases. OLE DB is a data source connectivity standard promoted by Microsoft as a successor to ODBC. One of the primary differences between OLE DB and ODBC is that ODBC supports access only to relational databases, whereas OLE DB provides access to both relational databases and nonrelational data sources, such as spreadsheet programs. The most recent version of ADO is ADO.NET, which allows you to access OLE DB-compliant data sources and XML. You can also use ADO.NET to directly access Microsoft SQL Server databases, without having to go through OLE DB.

In this chapter, you will use ADO.NET and OLE DB to access a Microsoft Access database. Although you are learning about databases, you should note that Access, and another popular DBMS, Paradox, are considered to be desktop database applications, not suited for large, enterprise-wide database systems that companies rely on for managing their businesses. Access and Paradox have their uses—both Access and Paradox databases are fairly easy to create and manage on a small scale. For mission-critical database applications, however, most companies use professional-strength, ODBC-compliant databases such as SQL Server, Oracle, Sybase, or Informix.

ADO and OLE DB are part of the Microsoft Universal Data Access strategy for providing access to data, regardless of its storage format. The components that make up the Universal Data Access technology are called the Microsoft Data Access Components, or MDAC. MDAC is installed with numerous Microsoft products, including Internet Explorer, Internet Information Server, Microsoft Visual Studio, and the Microsoft .NET Framework SDK. Most of these products, including Internet
Explorer, install MDAC automatically. If you are not sure if MDAC is installed on your system, you can download the most recent version from the Microsoft Data Access Technologies site at www.microsoft.com/data.

In this chapter you will work with an existing Microsoft Access database named WebAdventureCourses.mdb. The WebAdventureCourses.mdb database consists of two tables: **Students** and **Registration**. The **Students** table contains each student’s ID and name, along with other personal information. The **Registration** table contains a record for each class in which a student enrolls. The **Students** table is the primary table, and the **Student_ID** field acts as the primary key. The **Student_ID** field also acts as the foreign key in the **Registration** table. Because each student can enroll in more than one class, the relationship between the **Students** table and the **Registration** table is one-to-many; the **Students** table is the one side of the relationship, and the **Registration** table is the many side of the relationship. If you would like to examine the tables in the WebAdventureCourses.mdb database, you can open the database file in Access from your Chapter folder for Chapter 12.

Next, you will create the main Registration.html document. The Registration.html document is the first Web page students will see when they access the WebAdventure registration Web site. The document contains only text and XHTML elements, and does not contain any client-side JavaScript or ASP.NET code. The Registration.html document includes two forms with Submit buttons to call ASP.NET documents that access the database.

To create the Registration.html document:

1. Open your text editor and create a new document.

2. Type the `<!DOCTYPE>` declaration, `<html>` element, header information, and the `<body>` element. Use the strict DTD and “Registration” as the content of the `<title>` element. Your document should appear as follows:

```html
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html>
<head>
<title>Registration</title>
</head>
<body>
</body>
</html>
```

3. Add the following style section to the document head:

```css
body { font-family: "Trebuchet MS", Arial, Helvetica, sans-serif, serif }
h1 { font-size: 1.5em }
```

```css
h2 { font-size: 1.2em }
```
4. Add the following elements and text to the document body:

```html
<h1>WebAdventure Computer Training Registration</h1>
<h2>Welcome</h2>
<p>Welcome to Computer Training at WebAdventure! We offer a variety of computer training and technology courses that focus on the Web. To sign up for a course, please fill out the New Student Registration form and click the <strong>Get Student ID</strong> button to obtain a student ID. If you are a current student, enter your student ID number and click the Class Registration button to register for new classes or to review your current schedule.</p>

5. Add the following heading element and form to the end of the document body. The form calls a document named GetStudentID.aspx, which assigns student IDs.

```html
<h2>New Student Registration</h2>
<form method="post" action="GetStudentID.aspx">
<p>Last Name: <input type="text" name="last_name" size="30" /></p>
<p>First Name: <input type="text" name="first_name" size="30" /></p>
<p>Address: <input type="text" name="address" size="30" /> City, State, Zip: <input type="text" name="city" size="20" /></p>
<p><input type="text" name="state" size="2" maxlength="2" /></p>
<p><input type="text" name="zip" size="5" maxlength="5" /></p>
<p>E-Mail: <input type="text" name="email" size="50" /></p>
<p><input type="submit" name="submit" value="Get Student ID" /></p>
<input type="reset" /></form>
```

6. Now add the following heading element and form to the end of the document body. The form calls a document named CourseListing.aspx, which existing students use to register for new classes or to review their current schedule. Later in this chapter, you will create ASP.NET documents that are called by the forms.

```html
<h2>Returning Student Registration</h2>
<form method="post" action="CourseListing.aspx">
<p>Student ID: <input type="text" name="id" /></p>
<p><input type="submit" value="Class Registration" /></p>
</form>
```
7. Save the document as `Registration.html` in your Chapter folder for Chapter 12 and validate it with the W3C MarkUp Validation Service. Once the document is valid, close it in your text editor, and then open it in your Web browser from your IIS Web server by typing the following URL in the address box: `http://localhost/webadventure/Chapter.12/Chapter/Registration.html`. Figure 12-8 shows how the document appears. Do not click the Submit button yet because you still need to create the ASP.NET documents.

In order to open the URL in the preceding step, your IIS virtual directory for the WebAdventure Web site must be set to the path where your data files are located. You can find the instructions for setting your IIS virtual directory for the WebAdventure Web site in Chapter 11.

8. Close your Web browser window.

**ADO.NET Object Model**

ADO.NET technology is based on an object model that is used for accessing and manipulating data sources. Table 12-2 lists the core objects in the ADO.NET object model. You will work with several of the ADO.NET objects in this chapter.
This chapter provides only a brief overview of how to use ASP.NET and ADO.NET to access databases. For more information on ASP.NET database access with ADO.NET, visit the Microsoft Developer Network at http://msdn.microsoft.com/.

Before you learn how to access databases with ADO.NET, you need to create the CourseListing.aspx file that students will use to select the courses they want to take.

To create the CourseListing.aspx file:

1. Create a new document in your text editor.

2. Type the `<!DOCTYPE>` declaration, `<html>` element, header information, and the `<body>` element. Use the strict DTD and “Course Listing” as the content of the `<title>` element.

3. Add the following style section to the document head:

   `<style type="text/css">
   body { font-family: "Trebuchet MS", Arial, Helvetica, sans-serif, serif }
   h1 { font-size: 1.5em }
   h2 { font-size: 1.2em }
   p { font-size: .8em }
   </style>`

4. Add the following processing directive and code render block to the end of the document head. The if statement checks if the id property exists in the Request object Form collection. If the id property does not exist, it is assigned to the studentID property of the Session object Contents collection. The id property of the Request object Form collection will contain the value that students enter into the id field of the Registration form. You will use the studentID property throughout the registration program to keep track of users as they navigate through the pages that make up the program.

   `<%@ language="JScript" %>
   `<%`
if (parseInt(Request.Form("id")))
    Session.Contents("studentID") = parseInt(Request.Form("id"));
%

ASP.NET collection variables are saved as text data types. Therefore, you must use a data type conversion function when copying values from an ASP.NET collection variable to a variable that you intend to use in an expression.

5. Add the following heading element and form to the document body. The form contains two elements that are used for displaying a student's schedule. Submitting the form calls the ReviewSchedule.aspx document, which displays the student's schedule. The student’s ID is printed to the screen inside the form, using the output directive (<%= ).

```html
<h3>Course Registration Form</h3>
<form method="post" action="ReviewSchedule.aspx">
    <p><strong>Student ID: </strong><br />
       <%= Session.Contents("studentID") %></p>
    <input type="submit" value="Review Current Schedule " /></form>
</p>
</form>

6. Add the next form to the end of the document body. This form allows students to register for classes and is submitted to an ASP.NET script named RegisterStudent.aspx.

```html
<form method="post" action="RegisterStudent.aspx">
</form>
```

7. Add the following text and elements to the form, which create radio buttons for the available courses:

```html
<p><strong>Select the course you would like to take:</strong></p>
<input type="radio" name="course" value="Introduction to ASP.NET" />Introduction to ASP.NET<br />
<input type="radio" name="course" value="Introduction to JavaScript" />Introduction to JavaScript<br />
<input type="radio" name="course" value="Intermediate ASP.NET" />Intermediate ASP.NET<br />
<input type="radio" name="course" value="Intermediate to JavaScript" />Intermediate JavaScript<br />
<input type="radio" name="course" value="Advanced ASP.NET" />Advanced ASP.NET<br />
<input type="radio" name="course" value="Advanced JavaScript" />Advanced JavaScript</p>
```
8. Now add the following text and elements to the end of the form, which create two selection lists containing the available dates and times for the courses:

```html
<p><strong>Available Days and Times:</strong><br />
<select name="days">
<option selected="selected" value="Mondays and Wednesdays">Mondays and Wednesdays</option>
<option value="Tuesdays and Thursdays">Tuesdays and Thursdays</option>
<option value="Wednesdays and Fridays">Wednesdays and Fridays</option>
</select>
</select>

<select name="time">
<option selected="selected" value="9 a.m. - 11 a.m.">9 a.m. - 11 a.m.</option>
<option value="1 p.m. - 3 p.m.">1 p.m. - 3 p.m.</option>
<option value="6 p.m. - 8 p.m.">6 p.m. - 8 p.m.</option>
</select></p>

9. Finally, add the following submit and reset buttons to the end of the form:

```html
<p><input type="submit" value="Register" />
<input type="reset" /></p>
```

10. Save the document as CourseListing.aspx in your Chapter folder for Chapter 12, and then close it in your text editor. Before you can open the file, you need to write an ASP.NET script that generates new student IDs.

In order for your ASP.NET documents to access the ADO.NET object model, you must use the `import processing directive` to import a namespace. Namespaces are used for managing the various classes and other elements in the .NET Framework. There are numerous namespaces available to the .NET Framework, many of which are imported automatically into your ASP.NET programs. However, other namespaces, including the namespaces that give ASP.NET access to database connectivity classes, must be explicitly imported. To access the OLE DB namespace, which contains the classes you need for the exercises you create in this chapter, you must import the `System.Data.OleDb` namespace by adding the following statement to your ASP.NET documents:

```csharp
<%@ Import namespace="System.Data.OleDb" %>
```

### The ADO.NET Connection Object

With ADO.NET, you use a **Connection object** to access databases from ASP.NET. ADO.NET includes two **Connection objects**: the **SqlConnection object**, which connects to Microsoft SQL Server version 7.0 or later, and the **OleDbConnection object**, which connects to OLE DB data sources. You will work with the **OleDbConnection object** in this chapter. The **OleDbConnection** object contains various methods and properties for accessing and manipulating databases, as listed in Tables 12-3 and 12-4.
The first step in working with a database in ASP.NET is to create an instance of the `OleDbConnection` object using the following syntax:

```csharp
var object = new OleDbConnection("connection string");
```

The connection string that you pass to the `OleDbConnection` constructor must include the `Provider` and `DataSource` name=value pairs. You assign to `Provider` the name of the .NET data provider, which identifies the relational database system you want to access. At the time of this writing, the data providers listed in Table 12-5 have been tested with ADO.NET.

### Table 12-5  .NET data providers

<table>
<thead>
<tr>
<th>Provider</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SQLOLEDB</td>
<td>Microsoft OLE DB provider for SQL Server</td>
</tr>
<tr>
<td>MSDAORA</td>
<td>Microsoft OLE DB provider for Oracle</td>
</tr>
<tr>
<td>Microsoft.Jet.OLEDB.4.0</td>
<td>OLE DB provider for Microsoft Jet</td>
</tr>
</tbody>
</table>

The connection string that you pass to the `OleDbConnection` constructor must include the `Provider` and `DataSource` name=value pairs. You assign to `Provider` the name of the .NET data provider, which identifies the relational database system you want to access. At the time of this writing, the data providers listed in Table 12-5 have been tested with ADO.NET.

### Table 12-4  `OleDbConnection` object properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ConnectionString</code></td>
<td>The string used to open a data source</td>
</tr>
<tr>
<td><code>ConnectionTimeout</code></td>
<td>The time to wait before abandoning a client database connection attempt</td>
</tr>
<tr>
<td><code>Database</code></td>
<td>The name of the current database to use after a connection has been established</td>
</tr>
<tr>
<td><code>DataSource</code></td>
<td>The location and filename of the data source</td>
</tr>
<tr>
<td><code>Provider</code></td>
<td>The name of the OLE DB .NET data provider</td>
</tr>
<tr>
<td><code>ServerVersion</code></td>
<td>The version of the server to which the database is connected</td>
</tr>
<tr>
<td><code>State</code></td>
<td>A string indicating the current status of the database connection</td>
</tr>
</tbody>
</table>

### Table 12-3  `OleDbConnection` object methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>BeginTransaction()</code></td>
<td>Begins a transaction</td>
</tr>
<tr>
<td><code>ChangeDatabase()</code></td>
<td>Changes the currently opened database</td>
</tr>
<tr>
<td><code>Close()</code></td>
<td>Closes a data source connection</td>
</tr>
<tr>
<td><code>CreateCommand()</code></td>
<td>Creates and returns a <code>Command</code> object associated with the <code>OleDbConnection</code> object</td>
</tr>
<tr>
<td><code>GetOleDbSchemaTable()</code></td>
<td>Returns schema information from the data source</td>
</tr>
<tr>
<td><code>Open()</code></td>
<td>Opens a data source connection</td>
</tr>
<tr>
<td><code>ReleaseObjectPool()</code></td>
<td>Clears the <code>OleDbConnection</code> object pool</td>
</tr>
</tbody>
</table>
You use the `Microsoft.Jet.OLEDB.4.0` data provider to connect to Access databases. You assign to `DataSource` the path and filename of your Access database file. The following statement creates an `OleDbConnection` object named `dbConnection` and specifies `Microsoft.Jet.OLEDB.4.0` as the data provider and `C:\aspfiles\orders.aspx` as the data source:

```
var dbConnection = new OleDbConnection(
    "Provider=Microsoft.Jet.OLEDB.4.0;
    Data Source=C:\aspfiles\orders.aspx");
```

**Opening and Closing a Data Source**

Once you create an instance of the `OleDbConnection` object, you must use the `Open()` method to open a specific data source. One `OleDbConnection` object you should use whenever you open a database connection with the `Open()` method is the `Close()` method to disconnect the database connection. This is necessary because database connections do not close automatically when an ASP.NET program ends. If you do not close a database connection, it remains open indefinitely (at least until you reboot), and can possibly cause performance problems on the computer where the ASP.NET program is running. The following statement is an example of how to connect and disconnect to a database represented by a variable named `dbConnection`.

```
var dbConnection = new OleDbConnection(  
    "Provider=Microsoft.Jet.OLEDB.4.0;Data 
    Source=C:\aspfiles\employees.aspx");
    dbConnection.Open();
    additional statements;
    dbConnection.Close();
```

**Checking the Database Connection**

It is good practice to make sure your program has connected to a database successfully before it attempts to read, write, add, or modify records. The `State` property contains a string indicating the current status of the database connection. Table 12-6 lists the values that ADO.NET can assign to the `State` property.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Broken</td>
<td>The connection is broken</td>
</tr>
<tr>
<td>Closed</td>
<td>The connection is closed</td>
</tr>
<tr>
<td>Connecting</td>
<td>The Connection object is connecting to the data source</td>
</tr>
<tr>
<td>Executing</td>
<td>The connection is executing a command</td>
</tr>
<tr>
<td>Fetching</td>
<td>The connection is retrieving data</td>
</tr>
<tr>
<td>Open</td>
<td>The connection is open</td>
</tr>
</tbody>
</table>

Table 12-6  State property values
The following code adds an if statement that checks the State property after the statement that creates the new OleDbConnection object attempts to connect to a database. If the connection was unsuccessful, then the ASP.NET Response.Write() method returns a message to the client.

```javascript
var dbConnection = new OleDbConnection(  
  "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=C:\aspfiles\employees.aspx");
dbConnection.Open();
if (dbConnection.State != "Open")
  Response.Write("The database is not available.");
else {
  additional statements;
  dbConnection.Close();
}
```

Next, you will start creating the GetStudentID.aspx document, which generates new student IDs.

To create the GetStudentID.aspx document:

1. Create a new document in your text editor.

2. Type the <!DOCTYPE> declaration, <html> element, header information, and the <body> element. Use the strict DTD and “Student IDs” as the content of the <title> element.

3. Add the following style section to the document head:

   ```html
   <style type="text/css">
   body { font-family: "Trebuchet MS", Arial, Helvetica, sans-serif, serif }
   h1 { font-size: 1.5em }
   h2 { font-size: 1.2em }
   p { font-size: .8em }
   </style>
   ```

4. Add the following language, import processing directives, and code render block to the document body. The code locks and unlocks the Application object and generates a new student ID. If the idNum variable does not exist, it is created. If it does exist, then the current number is incremented by one and assigned to the projectID variable.

   ```javascript
   <%@ language="JScript" %>
   <%@ Import namespace="System.Data.OleDb" %>
   <%
   Application.Lock();
   if (!Application.Contents("idNum")) {
     Application.Contents("idNum") = 100;
     var curID = Application.Contents ("idNum");
   } else {
     curID = Application.Contents ("idNum");
     curID = curID + 1;
     Application.Contents ("idNum") = curID;
   }
   Response.Write("Student ID is: "+curID);
   Application.Lock();
   Application.Unlock();
   %
   ```
5. Add the following code to the end of the code render block, which opens a database connection to the WebAdventure.mdb database file:

```javascript
var dbConnection = new OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;Data Source=C:\JavaScript_Projects\Chapter.12\Chapter\WebAdventureCourses.mdb");
dbConnection.Open();
if (dbConnection.State != "Open")
  var responseString = "The database is not available."
```

6. Save the document as `GetStudentID.aspx` in your Chapter folder for Chapter 12.

**Executing SQL Commands through ASP.NET**

ADO.NET provides four primary objects for accessing and manipulating data sources: the Command, DataReader, DataSet, and DataAdapter objects. This section discusses the basics of how to work with the Command and DataReader objects. See the Microsoft Developer's Network at [http://msdn.microsoft.com](http://msdn.microsoft.com) for information on working with the DataSet and DataAdapter objects.

### The Command Object

The Command object executes a command, such as an SQL command, against a data source. ADO.NET includes two Connection objects: the SqlCommand object, which executes commands against Microsoft SQL Server version 7.0 or later, and the OleDbCommand object, which executes commands against an OLE DB data source. You will work with the OleDbCommand object in this chapter.

You must create an instance of the OleDbConnection object using the following syntax:

```javascript
var object = new OleDbCommand("command", connection);
```

The command parameter you pass to the OleDbCommand object is the SQL command you want to execute. The connection parameter represents the OleDbConnection object that represents the database connection.
The `OleDbCommand` object contains various methods and properties for executing commands against databases. The method you will study in this chapter is the `ExecuteNonQuery()` method, which executes commands against a database. A `Command` object is most useful for quickly inserting, updating, or deleting rows in a database. For example, the boldface code in the following code uses the SQL INSERT statement to add a new employee record to the `Employees` database table. If the connection is successful, the `else` clause executes necessary statements to perform the desired actions against the database before disconnecting. The SQL code is assigned to the `SQLString` variable, then passed to an `OleDbCommand` object named `empCommand`, along with the name of the database connection (`dbConnection`).

```csharp
var dbConnection = new OleDbConnection(
    "Provider=Microsoft.Jet.OLEDB.4.0;Data
Source=C:\aspfiles\employees.aspx");
dbConnection.Open();
if (dbConnection.State != "Open")
    Response.Write("The database is not available.");
else {
    var SQLString = "INSERT INTO Employees VALUES('106',
        'Mbuti', 'Pierre', '106 Flagg Road', 'Spencer',
        'MA', '01562');"
    var empCommand = new OleDbCommand(SQLString,
        dbConnection);
    empCommand.ExecuteNonQuery();
    dbConnection.Close();
}
```

The following code shows another example of the `ExecuteNonQuery()` method, which deletes a row from the `Employees` table:

```csharp
var dbConnection = new OleDbConnection(
    "Provider=Microsoft.Jet.OLEDB.4.0;Data
Source=C:\aspfiles\employees.aspx");
dbConnection.Open();
if (dbConnection.State != "Open")
    Response.Write("The database is not available.");
else {
    var SQLString = "DELETE FROM Employees
WHERE last_name = 'Miller'";
    var empCommand = new OleDbCommand(SQLString,
        dbConnection);
    empCommand.ExecuteNonQuery();
    dbConnection.Close();
}
```

The SQL string in the preceding code uses the `WHERE` clause to look for rows in the table where the `last_name` field is equal to Miller. Note that the preceding statement would actually delete all rows in the table where the `last_name` field is equal to Miller. The statement is safe with this example, because you know that there is only one record
that contains Miller in the last_name field. However, it’s important to understand exactly what records will be deleted before executing the DELETE statement. Also, be sure to include a WHERE clause when using the DELETE statement or all of the rows in the specified table will be deleted.

Next, you will add code to the GetStudentID.aspx file that writes records to the database, using the ExecuteNonQuery() method.

To add code to the GetStudentID.aspx file that writes records to the database, using the ExecuteNonQuery() method:

1. Return to the GetStudentID.aspx file in your text editor.

2. Add the following statements to the end of the code render block. The statements are contained in an else structure that executes only if the preceding if statement, which uses the State property to check if the database is connected, returns a value of “Open”.

```javascript
else {
    var SQLString = "INSERT INTO Students VALUES('"
    + curID + ", '", 
    + Request.Form("last_name") + ", '", 
    + Request.Form("first_name") + ", '", 
    + Request.Form("address") + ", '", 
    + Request.Form("city") + ", '", 
    + Request.Form("state") + ", '", 
    + Request.Form("zip") + ", '", 
    + Request.Form("email") + ");
    var dbCommand = new OleDbCommand(SQLString, 
dbConnection);
    dbCommand.ExecuteNonQuery();
}
```

3. Next, add the following statements to the end of the code render block, which builds a text string in the responseString variable, which will be returned as a response to the client. Be sure not to include any line breaks in the literal strings—the lines are broken below due to space limitations.

```javascript
var responseString = "<h2>WebAdventure Computer Training 
Registration</h2>";
responseString = responseString + "<p>Thanks " + 
Request.Form("first_name") + "! Your new student ID is <strong>" 
+ curID + "</strong>";
responseString = responseString + ". Click <a href='CourseListing.aspx'>Registration</a> to proceed to the 
course registration page.</p>
"
```

4. Close the database connection and the if...else structure.

```javascript
    dbConnection.Close();
}
```
5. Add the following statements to the document body, which return a response to the user:

```html
<p><%=ƒresponseString %></p>
```

6. Save GetStudentID.aspx, close it in your text editor, and then open the Registration.html file in your Web browser from your ASP.NET server. Fill out the registration information and click the Get Student ID button. You should receive a response similar to Figure 12-9. Write down the new Student ID; you need it for the next exercise.

![GetStudentID.aspx](image)

**Figure 12-9** GetStudentID.aspx

7. Close your Web browser window.

Next, you will create the RegisterStudent.aspx document.

To create the RegisterStudent.aspx document:

1. Create a new document in your text editor.

2. Type the `<!DOCTYPE>` declaration, `<html>` element, header information, and the `<body>` element. Use the strict DTD and “Register Student” as the content of the `<title>` element.

3. Add the following style section to the document head:

```html
<style type="text/css">
body { font-family: "Trebuchet MS", Arial, Helvetica, sans-serif, serif }
  h1 { font-size: 1.5em }
  h2 { font-size: 1.2em }
  p { font-size: .8em }
</style>
```
4. Add the following language, import processing directives, and code render block to the end of the document head. The code opens the database connection.

```javascript
<%@ language="JScript" %>
<%@ Import namespace="System.Data.OleDb" %>
<%
var dbConnection = new OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;Data Source=C:\JavaScript_Projects\Chapter.12\Chapter\WebAdventureCourses.mdb");
dbConnection.Open();
if (dbConnection.State != "Open")
    var responseString = "The database is not available.";
%
```

5. Next, add the following code to the end of the code render block, which executes the SQL statements using the `ExecuteNonQuery()` method:

```javascript
else {
    var SQLString = "INSERT INTO Registration VALUES('"
    + Session.Contents("studentID") + ", ", "
    + Request.Form("course") + ", ", "
    + Request.Form("days") + ", ", "
    + Request.Form("time") + ");
    var dbCommand = new OleDbCommand(SQLString, dbConnection);
    dbCommand.ExecuteNonQuery();
}
```

6. Add the following statements to the end of the code render block; these statements return a response to the user. Be sure to type the literal string within the parentheses on a single line.

```javascript
var responseString = "<h2>WebAdventure Computer Training Registration</h2>"
responseString = responseString + "<p>You are registered for "
    + Request.Form("course") + " on 
    + Request.Form("days") + ", 
    + Request.Form("time")
responseString = responseString + " To register for another course, click <a href='CourseListing.aspx'>Course Listing</a>. Or click <a href='ReviewSchedule.aspx'>Review Schedule</a> to review your current schedule.";
```

7. Close the database connection and the `if...else` statement.

```javascript
dbConnection.Close();
```
8. Add the following statements to the document body, which return a response to the user:

```html
<p>{$responseString}</p>
```

9. Save the file as `RegisterStudent.aspx` in your Chapter folder for Chapter 12, close it in your text editor, and then open `Registration.html` document from your ASP.NET server. Enter the student ID you created previously and click the **Class Registration** button to open `CourseListing.aspx`. Figure 12-10 shows how the document should appear.

![Course Listing](image)

**Figure 12-10**  CourseListing.aspx

10. Fill out the course form and click the **Register** button. Figure 12-11 shows an example of the response returned from `RegisterStudent.aspx`. 
11. Close your Web browser window.

**The DataReader Object**

A **DataReader object** retrieves read-only, forward-only data from a data source. **Forward-only** means that the program can only move forward through the record in the returned data, and not backward or to specific records. In other words, your program can sequentially read the records from the first record to the last record, but cannot sequentially read the records from the last record to the first record, or go to a specific record. You use a **DataReader object** when you want to read data from a database, but not add, delete, or modify records. ADO.NET includes two **DataReader objects**: the **SqlDataReader object**, which retrieves data from Microsoft SQL Server version 7.0 or later, and the **OleDbDataReader object**, which retrieves data from OLE DB data sources. You will work with the **OleDbDataReader object** in this chapter.

You use the **ExecuteReader() method** of the **OleDbCommand object** to create an **OleDbDataReader object**. The syntax is virtually identical to the **ExecuteNonQuery() method**, except that you must assign the object returned from the **ExecuteReader() method** to a variable that represents the new object. The boldface code in the following code demonstrates how to create an **OleDbDataReader object** named **empRecords** that contains the **last_name** and **first_name** fields from the **Employees** table. The SQL statement uses the SELECT keyword to return the data.

```javascript
var dbConnection = new OleDbConnection(  
  "Provider=Microsoft.Jet.OLEDB.4.0;Data  
Source=C:\aspfiles\employees.aspx");
dbConnection.Open();
if (dbConnection.State != "Open")
  Response.Write("The database is not available.");
else {
```
The OleDbDataReader object contains various properties and methods for reading the returned data. The one method of the OleDbDataReader object you will study in this chapter is the `Read()` method, which advances the OleDbDataReader object to the next record. When you work with an OleDbDataReader object, your position within the record set is called the cursor. When an OleDbDataReader object is first created, the cursor is initially placed before the first row in the record set. Figure 12-12 shows an example of where the cursor is placed when the Employees table is first opened in an OleDbDataReader object.

![Figure 12-12  Initial cursor position in an OleDbDataReader object](image)

You never actually see the record set in an OleDbDataReader object as it is shown in Figure 12-12. The illustration in Figure 12-12 is for demonstration purposes only.

The first time you use the `Read()` method, it places the cursor in the first row of the record set. For example, the following code creates a new OleDbDataReader object named `empRecords`, and then moves the cursor to the first record in the resulting record set:

```javascript
var SQLString = "SELECT * FROM Employees ORDER BY last_name, first_name";
var empCommand = new OleDbCommand(SQLString, dbConnection);
empRecords = empCommand.ExecuteReader();
empRecords.Read();
```

When you work with record sets and the `Read()` method, you can never be certain if there is another record following the current position of the cursor, or even if any records were returned at all from your SQL SELECT statement. To determine if a next
record is available, you can use the `Read()` method, which returns a value of true if it finds a next row in the record set or a value of false if it does not find a next row in the record set. The following code shows how to use an `if` statement to check the value returned by the `Read()` method before moving the cursor. Notice that the `Read()` method is executed as the conditional expression of the `if` statement. Also notice that the conditional expression does not include a comparison operator. The `Read()` method returns a value of true or false automatically as it is executed, eliminating the need for a comparison operator.

```csharp
var SQLString = "SELECT * FROM Employees
    ORDER BY last_name, first_name";
var empCommand = new OleDbCommand(SQLString, dbConnection);
var empRecords = empCommand.ExecuteReader();
if (empRecords.Read()) {
    statements;
} else
    Response.Write("Your query returned no records.");
```

The field names in a database table are assigned as variables in an `OleDbDataReader` object collection. For example, if you instantiate an `OleDbDataReader` object named `empRecords` for the `Employees` database, then you can refer to the `First_Name` field by using a statement similar to `empRecords("Last_Name")`. Be aware that whenever you use the `Read()` method, the content of each variable in an `OleDbDataReader` object changes to reflect the contents of the record at the current location of the cursor. The following code shows a simple program that returns the name of each programmer in the `Employees` table to the client along with the name of the city where each programmer lives. The program uses a `do...while` statement to move through the records in the table. Figure 12-13 shows the results returned to a client.

```csharp
var SQLString = "SELECT * FROM Employees
    ORDER BY last_name, first_name";
var empCommand = new OleDbCommand(SQLString, dbConnection);
var empRecords = empCommand.ExecuteReader();
if (empRecords.Read()) {
    do {
        Response.Write(empRecords("First_Name")
            + empRecords("Last_Name") + " lives in "
            + empRecords("City") + ", "
            + empRecords("State") + ".<br/>");
    } while (empRecords.Read());
} else
    Response.Write("Your query returned no records.");
```
When you are through working with an OleDbDataReader object, you must close it with the Close() method, the same way you close a database connection. This allows you to reuse the OleDbDataReader object to retrieve other record sets. Also keep in mind that an OleDbDataReader object has exclusive access to the database Connection object. This means you cannot execute any other commands against the database until the OleDbDataReader object is closed. To close the empRecords object, use the following statement:

```
empRecords.Close();
```

Next, you will create the ReviewSchedule.aspx document, which displays courses a student is registered for.

To create the ReviewSchedule.aspx document:

1. Create a new document in your text editor.
2. Type the `<!DOCTYPE>` declaration, `<html>` element, header information, and the `<body>` element. Use the strict DTD and “Review Schedule” as the content of the `<title>` element.
3. Add the following style section to the document head:

```
<STYLE TYPE="TEXT/CSS">
BODY { FONT-FAMILY: "Trebuchet MS", Arial, Helvetica, sans-serif, serif }
H1 { FONT-SIZE: 1.5EM }
H2 { FONT-SIZE: 1.2EM }
P { FONT-SIZE: .8EM }
</STYLE>
```
4. Add the following language, import processing directives, and code render block to the end of the document head. The code opens the database connection.

```html
<%@ language="JScript" %>
<%@ Import namespace="System.Data.OleDb" %>
<%
    var dbConnection = new OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;Data Source=C:\JavaScript_Projects\Chapter.12\Chapter\WebAdventureCourses.mdb");
    dbConnection.Open();
    if (dbConnection.State != "Open")
        var responseString = "The database is not available."
;
%
```

5. Next, add the following code to the end of the code render block to create and execute the SQL statement and the `OleDbDataReader` object. The SQL statement restricts the records returned to just those records that match the student ID.

```javascript
else {
    var SQLString = "SELECT * FROM Registration WHERE Student_ID = '" + Session.Contents("studentID") + "'";
    var dbCommand = new OleDbCommand(SQLString, dbConnection);
    var classData = dbCommand.ExecuteReader();
}
```

6. Add the following statements to the end of the code render block, which build the `responseString` variable, which returns a response to the student:

```javascript
var responseString = "<h2>This is your current schedule</h2>";
if (classData.Read()) {
    do {
        responseString = responseString
            + classData("Course") + ", ",
            + classData("Days") + ", ",
            + classData("Time") + "<br />";
    } while (classData.Read());
} else
    var responseString = "Your query returned no records."
;
7. Add the following statements to the end of the code render block to close the database connection the `OleDbDataReader` object:

```javascript
    classData.Close();
    dbConnection.Close();
```
8. Add the following statements to the document body, which return a response to the user:

\[
\text{<p><%=ƒresponseString %></p>}
\]

9. Save the file as \texttt{ReviewSchedule.aspx} in your Chapter folder for Chapter 12, close it in your text editor, and then open the \texttt{Registration.html} document from your ASP.NET server. Enter an existing student ID that is already registered for some classes and click the \texttt{Class Registration} button to open \texttt{CourseListing.aspx}. From the \texttt{CourseListing.aspx} document, click the \texttt{Review Current Schedule} button. Figure 12-14 shows how the document should appear.

![ReviewSchedule.aspx](image)

\textbf{Figure 12-14} \hspace{1em} \texttt{ReviewSchedule.aspx}

10. Close your Web browser and text editor.

\textbf{CHAPTER SUMMARY}

- A database is an ordered collection of information from which a computer program can quickly access information.
- A record in a database contains a single, complete set of related information.
- Fields are the individual categories of information stored in a record.
- A flat-file database stores information in a single table.
- A relational database stores information across multiple related tables.
- A primary table is the main table in a relationship that is referenced by another table.
- A related table (also called a child table) references a primary table in a relational database.
A primary key is a field that contains a unique identifier for each record in a primary table.

A foreign key is a field in a related table that refers to the primary key in a primary table.

A one-to-one relationship exists between two tables when a related table contains exactly one record for each record in the primary table.

A one-to-many relationship exists in a relational database when one record in a primary table has many related records in a related table.

Breaking tables into multiple related tables in order to reduce redundant and duplicate information is called normalization.

A many-to-many relationship exists in a relational database when many records in one table are related to many records in another table.

An application or collection of applications used to create, access, and manage a database is called a database management system, or DBMS.

A database management system that stores data in a flat-file format is called a flat-file database management system.

A database management system that stores data in a relational format is called a relational database management system, or RDBMS.

Structured query language, or SQL, has become a standard data manipulation language among many database management systems.

ActiveX Data Objects, or ADO, is a Microsoft database connectivity technology that allows ASP and other Web development tools to access ODBC- and OLE DB-compliant databases.

With ADO.NET, you use a Connection object to access databases from ASP.NET. ADO.NET includes two Connection objects: the SqlConnection object, which connects to Microsoft SQL Server version 7.0 or later, and the OleDbConnection object, which connects to OLE DB data sources.

The Command object executes a command, such as a SQL command, against a data source. ADO.NET includes two Command objects: the SqlCommand object, which executes commands against Microsoft SQL Server version 7.0 or later, and the OleDbCommand object, which executes commands against an OLE DB data sources.

A DataReader object retrieves read-only and forward-only data from a data source. Forward-only means that the program can only move forward through the record in the returned data, and not backward or to specific records.

ADO.NET includes two DataReader objects: the SqlDataReader object, which retrieves data from Microsoft SQL Server version 7.0 or later, and the OleDbDataReader object, which retrieves data from OLE DB data sources.
REVIEW QUESTIONS

1. A flat-file database consists of a single table. True or false?
2. Explain how relational databases are organized.
3. What is the correct term for the individual pieces of information that are stored in a database record?
   a. element
   b. field
   c. section
   d. container
4. What is the name of one table’s primary key when it is stored in another table? (Choose all that apply.)
   a. key symbol
   b. record link
   c. foreign key
   d. unique identifier
5. Breaking tables into multiple related tables in order to reduce redundant and duplicate information is called _________________.
   a. normalization
   b. redundancy design
   c. splitting
   d. simplification
6. Suppose you have a relational database for a dry cleaning company. Each customer of the dry cleaning company can have multiple items in a cleaning order. What type of relationship is this?
   a. one-to-one
   b. one-to-many
   c. many-to-one
   d. many-to-many
7. ________________ has become somewhat of a standard data manipulation language among many database management systems.
   a. Java
   b. SQL
   c. ASP.NET
   d. PERL
8. Which of the following is the correct string for a filter that narrows a record set to include only records where the State field is equal to Massachusetts?
   a. "WHERE [State] = 'Massachusetts'"
   b. "State = 'Massachusetts'"
   c. "WHERE [State] = Massachusetts"
   d. "[State] = 'Massachusetts'"

9. Files created by different database management systems are completely interchangeable. True or false?

10. What is one of the primary differences between ODBC and OLE DB?
    a. ODBC can be used only on Windows platforms.
    b. OLE DB functions only with relational databases.
    c. OLE DB provides access to both relational databases and nonrelational data sources.
    d. ODBC functions only with nonrelational databases.

11. Which of the following objects are ADO.NET Connection objects? (Choose all that apply.)
    a. OleDbConnection
    b. DbConnection
    c. OleDbConnection
    d. SqlConnection

12. Which of the following are .NET data providers? (Choose all that apply.)
    a. SQLOLEDB
    b. MSDAORA
    c. MDAC
    d. Microsoft.Jet.OLEDB.4.0

13. What is the correct syntax for creating a database object in ASP?
    a. var object = new OleDbConnection("connection string");
    b. var object = new (OleDbConnection);
    c. var OleDbConnection = new OleDbConnection("connection string");
    d. var OleDbConnection = new OleDbConnection();

14. Explain the steps involved in opening and closing a data source.
15. What is the value returned by the State property if a connection to a database has been successfully established?
   a. Closed
   b. Open
   c. Connecting
   d. Ready

16. Which of the following objects are ADO.NET Command objects? (Choose all that apply.)
   a. OleDbCommand
   b. DbCommand
   c. OleDbCommand
   d. SqlCommand

17. What is the second parameter that you must pass to a Command object?
   a. the SQL command you want to execute
   b. the value assigned to the Connection object’s State property
   c. the name of the data provider
   d. the Connection object that represents the database connection

18. Which of the following are methods of the Command object? (Choose all that apply.)
   a. Execute()
   b. ExecuteReader()
   c. ExecuteQuery()
   d. ExecuteNonQuery()

19. Which of the following objects are ADO.NET DataReader objects? (Choose all that apply.)
   a. OleDbDataReader
   b. DbDataReader
   c. OleDbDataReader
   d. SqlDataReader

20. Where is the cursor placed when you first create a DataReader object?
   a. before the first record
   b. on the first record
   c. on the last record
   d. after the last record
**Hands-on Projects**

**Project 12-1**
Redesign the table below to create a one-to-many relationship. Use any word-processing program, such as Microsoft Word, that allows you to create tables.

<table>
<thead>
<tr>
<th>Employee_ID</th>
<th>Last_Name</th>
<th>First_Name</th>
<th>Hourly_Pay</th>
<th>Department</th>
</tr>
</thead>
<tbody>
<tr>
<td>EMP001</td>
<td>Smith</td>
<td>Lucille</td>
<td>$32.50</td>
<td>Marketing</td>
</tr>
<tr>
<td>EMP002</td>
<td>Perez</td>
<td>Frank</td>
<td>$40.00</td>
<td>Legal</td>
</tr>
<tr>
<td>EMP003</td>
<td>Okayabashi</td>
<td>Mike</td>
<td>$22.00</td>
<td>Accounting</td>
</tr>
<tr>
<td>EMP004</td>
<td>Korso</td>
<td>Anthony</td>
<td>$28.00</td>
<td>Accounting</td>
</tr>
<tr>
<td>EMP005</td>
<td>Singh</td>
<td>Tasneem</td>
<td>$37.00</td>
<td>Legal</td>
</tr>
</tbody>
</table>

**Project 12-2**
Redesign the table below to create a many-to-many relationship. Use any word-processing program, such as Microsoft Word, that allows you to create tables.

<table>
<thead>
<tr>
<th>Employee_ID</th>
<th>Last_Name</th>
<th>First_Name</th>
<th>Project_ID</th>
<th>Project_Name</th>
<th>Hours_On_Project</th>
</tr>
</thead>
<tbody>
<tr>
<td>EMP001</td>
<td>Smith</td>
<td>Lucille</td>
<td>100-002</td>
<td>Ad campaign</td>
<td>14</td>
</tr>
<tr>
<td>EMP001</td>
<td>Smith</td>
<td>Lucille</td>
<td>100-003</td>
<td>Marketing brochure</td>
<td>9</td>
</tr>
<tr>
<td>EMP002</td>
<td>Perez</td>
<td>Frank</td>
<td>200-056</td>
<td>Vendor contracts</td>
<td>23</td>
</tr>
<tr>
<td>EMP005</td>
<td>Singh</td>
<td>Tasneem</td>
<td>200-056</td>
<td>Vendor contracts</td>
<td>17</td>
</tr>
<tr>
<td>EMP003</td>
<td>Okayabashi</td>
<td>Mike</td>
<td>300-010</td>
<td>Accounts receivable integration</td>
<td>8</td>
</tr>
<tr>
<td>EMP004</td>
<td>Korso</td>
<td>Anthony</td>
<td>300-010</td>
<td>Accounts receivable integration</td>
<td>12</td>
</tr>
<tr>
<td>EMP003</td>
<td>Okayabashi</td>
<td>Mike</td>
<td>300-012</td>
<td>Year-end tax returns</td>
<td>56</td>
</tr>
</tbody>
</table>

**Project 12-3**
New student IDs in the registration program you created in this chapter are generated from the `idNum` property of the `Application` object `Contents` collection. Generating new student IDs from a property in the `Application` object is not necessarily the best
method of generating new student IDs because the value in idNum is reinitialized each time you restart the registration program. In this project, you will modify the registration program so that new student IDs are generated from the Students table in the database instead of from the idNum property of the Application object Contents collection.

1. Open the GetStudentID.aspx document from your Chapter folder for Chapter 12 in your text editor.

2. Delete the following statements from the document. These statements generate student IDs from the idNum property of the Application object Contents collection. You do not need them because you will be adding code that generates new student IDs generated from the Students table in the database.

   ```
   Application.Lock();
   if (!Application.Contents("idNum")) {
     Application.Contents("idNum") = 100;
     var curID = Application.Contents("idNum");
     Session.Contents("studentID") = curID;
   }
   else {
     var curID = Application.Contents("idNum");
     ++curID;
     Session.Contents("studentID") = curID;
     Application.Contents("idNum") = curID;
   }
   Application.UnLock();
   ```

3. Next, add the following code to the else statement, immediately above the statement that declares the SQLString variable. The code uses the ExecuteReader() command to check whether any records exist in the Students table. If records do exist, then the while statement within the if statement navigates to the last record in the record set and copies the value of the Student_ID to the curID variable, which is then incremented by one. If no records exist, then the curID variable’s default value of 100 is used.

   ```
   var checkID = "SELECT * FROM Students ORDER BY Student_ID";
   var idCommand = new OleDbCommand(checkID, dbConnection);
   var idInfo = idCommand.ExecuteReader();
   var curID = 100;
   if (idInfo.Read()) {
     do {
       curID = idInfo("Student_ID");
     } while (idInfo.Read());
     ++curID;
   }
   idInfo.Close();
   Session.Contents("studentID") = curID;
   Application.Contents("idNum") = curID;
   ```
4. Save the `GetStudentID.aspx` document and then open the `Registration.html` document from your ASP.NET server and create several new student IDs to see if they are generated automatically. Close your Web browser and then reopen the `Registration.html` document and create another student ID. The new student ID should be one more than the last ID you created.

5. Close your Web browser window.

**Project 12-4**

In this project, you will create an ASP.NET program that saves a Web site hit list counter to a database. Your Projects folder for Chapter 12 contains a subfolder named HitCounter that contains an Access database, HitCounter.mdb, that you can use for this project.

1. Create a new document in your text editor.

2. Type the `<!DOCTYPE>` declaration, `<html>` element, header information, and the `<body>` element. Use the strict DTD and “Hit Counter” as the content of the `<title>` element.

3. Add the following processing directives and code render block to the end of the document body.

   ```html
   < %@ language="JScript" %>
   < %@ Import namespace="System.Data.OleDb" %>
   < %>
   ```

4. Add the following statements to the code render block to open a connection to the `HitCounter.mdb` database. Modify the path as necessary to access the HitCounter.mdb database on your computer.

   ```javascript
   var dbConnection = new OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;Data Source=C:\JavaScript_Projects\Chapter.12\Projects\HitCounter\HitCounter.mdb");
   dbConnection.Open();
   ```

5. Add the following `if...else` statement. The conditional expression in the `if` statements checks to see if the database connection opened successfully. If so, the `else` statement executes and updates the HitCounter.mdb database with the new hit number.

   ```javascript
   if (dbConnection.State != "Open")
   
   Response.Write("The database is not available.");

   else {
   
   var SQLString = "SELECT Hits FROM Hit_Count";
   
   var checkCounter = new OleDbCommand(SQLString, dbConnection);
   ```
var curGuestNum = 1;
var counterInfo = checkCounter.ExecuteReader();
counterInfo.Read();
curGuestNum = counterInfo("Hits");
++curGuestNum;
counterInfo.Close();
SQLString = "DELETE * FROM Hit_Count";
var updateCounter = new OleDbCommand(SQLString,
    dbConnection);
updateCounter.ExecuteNonQuery();
SQLString = "INSERT INTO Hit_Count VALUES(" + curGuestNum + ")"
updateCounter = new OleDbCommand(SQLString, dbConnection);
updateCounter.ExecuteNonQuery();
Response.Write("<h1>There have been " + curGuestNum + " hits to this page!</h1>");
dbConnection.Close();
}

6. Save the document as HitCounter.aspx in the HitCounter folder located in the Projects folder for Chapter 12, and then open it from your ASP.NET server. The first time you open the document, the hit counter should tell you there has been one hit to the page. Refresh your Web browser window and the hit counter should increment by one.

7. Close your Web browser window.

Project 12-5

In this project, you will create an ASP.NET program that saves a Web site guest book to a database. Your Projects folder for Chapter 12 contains a subfolder named GuestBook that contains an Access database, GuestBook.mdb, which you can use for this project.

1. Create a new document in your text editor. This document will be the main Web page where users can enter their names to “sign” the guest book.

2. Type the <!DOCTYPE> declaration, <html> element, header information, and the <body> element. Use the strict DTD and “Guest Book” as the content of the <title> element.

3. Add the following text and elements to the document body. Users can use the form to submit their names to the guest book.

   <h2>Enter your name to sign our guest book</h2>
   <form method="post" action="GuestRegister.aspx">
   <p>First Name <input type="text" name="first_name" /></p>
   <p>Last Name <input type="text" name="last_name" /></p>
   <p><input type="submit" value="Submit" /></p>
   </form>
4. Save the document as **GuestBook.html** in the GuestBook folder located in the Projects folder for Chapter 12, and then validate it with the W3C MarkUp Validation Service.

5. Create another document in your text editor. This document will be the ASP.NET program that writes user names that are submitted by the document to the database.

6. Type the `<!DOCTYPE>` declaration, `<html>` element, header information, and the `<body>` element. Use the strict DTD and “Guest Book” as the content of the `<title>` element.

7. Add the following processing directives and code render block to the end of the document body:

   ```
   < %@ language="JScript" %>
   < %@ Import namespace="System.Data.OleDb" %>
   < %
   ```

8. Add the following statements to the code render block to open a connection to the **GuestBook.mdb** database. Modify the path as necessary to access the GuestBook.mdb database on your computer.

   ```
   var dbConnection = new OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;Data Source=C:\JavaScript_Projects\Chapter.12\Projects\GuestBook\GuestBook.mdb");
   dbConnection.Open();
   ```

9. Add the following `if...else` statement. The conditional expression in the `if` statements checks to see if the database connection opened successfully. If so, the `else` statement executes and updates the GuestBook.mdb database with the guest information.

   ```
   if (dbConnection.State != "Open")
       Response.Write("The database is not available.");
   else {
       var SQLString = "INSERT INTO Guests VALUES('"
       + Request.Form("first_name") + "," + Request.Form("last_name") + ");"
       var updateGuestBook = new OleDbCommand(SQLString, dbConnection);
       updateGuestBook.ExecuteNonQuery();
       Response.Write("<h2>Thanks for registering, " + Request.Form("first_name") + "!</h2>");
   }
   dbConnection.Close();
   ```
10. Save the document as **GuestRegister.aspx** in the GuestBook folder located in the Projects folder for Chapter 12, and then open the **GuestBook.html** document from your ASP.NET server. Enter your first name and last name and click the Submit button. You should receive a response thanking you for registering. Click your browser’s **Back** button and enter several more names into the guest book; you will need to have some records entered into the database for the next project.

11. Close your Web browser window.

**Project 12-6**

In this project you will add a document to the Guest Book program you created in Project 12-5. This document will display the entries in the guest book.

1. Create a new document in your text editor.

2. Type the `<!DOCTYPE>` declaration, `<html>` element, header information, and the `<body>` element. Use the strict DTD and “Guest Book” as the content of the `<title>` element.

3. Add the following processing directives and code render block to the end of the document body:

```html
<%@ language="JScript" %>
<%@ Import namespace="System.Data.OleDb" %>
<% 
```

4. Add the following statements to the code render block to open a connection to the **GuestBook.mdb** database. Modify the path as necessary to access the GuestBook.mdb database on your computer.

```javascript
var dbConnection = new OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;Data Source=C:\JavaScript_Projects\Chapter.12\Projects\GuestBook\GuestBook.mdb");
dbConnection.Open();
```

5. Add the following if...else statement. The conditional expression in the if statement checks to see if the database connection opened successfully. If so, the else statement executes and displays the names in the guest book.

```javascript
if (dbConnection.State != "Open")
   Response.Write("The database is not available.");
else {
   var SQLString = "SELECT * FROM Guests";
   var dbCommand = new OleDbCommand(SQLString, dbConnection);
   var guestData = dbCommand.ExecuteReader();
   if (guestData.Read()) {
```
Response.Write("<p><strong>Here is the guest list</strong></p>");
    do {
        Response.Write(guestData("First_Name") + " " + guestData("Last_Name") + "<br />");
    } while(guestData.Read());
} else
    Response.Write("<p><strong>Nobody has signed the guest book!</strong></p>");
guestData.Close();
dbConnection.Close();
}

6. Save the document as **ShowGuestBook.aspx** in the GuestBook folder located in the Projects folder for Chapter 12.

7. Open the **GuestBook.html** document in your text editor and add the following simple form to the end of document body. The form opens the ShowGuestBook.aspx document.

   <form method="post" action="ShowGuestBook.aspx">
   <p><input type="submit" value="Show Guest Book" /></p>
   </form>

8. Open the **GuestBook.html** document from your ASP.NET server and click the **Show Guest Book** button. You should see the names you entered in the last project.


---

**CASE PROJECTS**

For the following projects, use a database management system such as Access, Paradox, or SQL Server to create the database file where you will store the data. Save the files you create in your Cases folder for Chapter 12.

**Case Project 12-1**

Create a database application that stores airline surveys. Include fields for the date and time of the flight, flight number, and so on. Also, include radio buttons for the following questions:

- Friendliness of customer staff?
- Space for luggage storage?
- Comfort of seating?
Cleanliness of aircraft?
Noise level of aircraft?
Wait time for check-in?

Each radio button group should include the following buttons that allow users to select a rating for each question: No Opinion, Poor, Fair, Good, or Excellent. Each record in the database you create should store the results of a single survey. Include a View Past Survey Results button on the main survey page that displays a list of past survey results.

Case Project 12-2
Create a telephone directory application that saves entries to a database. You should include standard telephone directory fields in the database such as name, address, city, state, zip, telephone number, and so on. Create a document as a main “directory” where you can select and retrieve records. Also, create one document that you can use to add new entries to your database and another document that you can use to edit entries.

Case Project 12-3
Create a shopping cart application for an online bookstore. Use the course registration program as a model. Instead of adding course registrations to the database, you add purchase information. Use different pages in the application for different types of books. The shopping cart should build a list of books that users want to purchase and provide a checkout mechanism that writes the information to a database and returns a response to the user.

Case Project 12-4
Create a database application for storing software development bug reports. Include fields such as product name and version, type of hardware, operating system, frequency of occurrence, and proposed solutions. Include two buttons on the main page that allow you to create a new bug report and update an existing bug report.