



Part **1**

Introduction to ADO.NET and Databases

- ◆ **Chapter 1: Introduction to Database Programming with ADO.NET**
- ◆ **Chapter 2: Introduction to Databases**
- ◆ **Chapter 3: Introduction to the Structured Query Language**
- ◆ **Chapter 4: Introduction to Transact-SQL Programming**
- ◆ **Chapter 5: Overview of the ADO.NET Classes**
- ◆ **Chapter 6: Introducing Windows Applications and ADO.NET**



Chapter 1

Introduction to Database Programming with ADO.NET

A *DATABASE* IS AN organized collection of information that is divided into *tables*. Each table is further divided into *rows* and *columns*; these columns store the actual information. You access a database using *Structured Query Language* (SQL), which is a standard language supported by most database software including SQL Server, Access, and Oracle.

In this chapter, you'll see a C# program that connects to a SQL Server database, retrieves and displays the contents stored in the columns of a row from a table, and then disconnects from the database. You'll also see programs that connect to Access and Oracle databases.

You'll also learn about Microsoft's rapid application development (RAD) tool, Visual Studio .NET (VS .NET). VS .NET enables you to develop, run, and debug programs in an integrated development environment. This environment uses all the great features of Windows, such as the mouse and intuitive menus, and increases your productivity as a programmer.

In the final sections of this chapter, you'll see how to use the extensive Microsoft documentation that comes with the .NET Software Development Kit (SDK) and VS .NET. You'll find this documentation invaluable as you become an expert with ADO.NET and C#. You'll also learn how to use the SQL Server documentation.

Featured in this chapter:

- ◆ Obtaining the required software
- ◆ Developing your first ADO.NET program
- ◆ Connecting to Access and Oracle databases
- ◆ Introducing Visual Studio .NET
- ◆ Using the .NET documentation
- ◆ Using the SQL Server documentation

Obtaining the Required Software

Before you can develop C# programs, you'll need to install either the .NET Software Development Kit (SDK) or VS .NET. You can download the .NET SDK at <http://msdn.microsoft.com/downloads> (search for the Microsoft .NET Framework Software Development Kit). You can purchase a trial or full copy of VS .NET from Microsoft at <http://msdn.microsoft.com/vstudio>.

To install the .NET SDK, run the executable file you downloaded and follow the instructions on the screen to install it on your computer. To install VS .NET, run the `setup.exe` file on the disk and follow the instructions on the screen.

You'll also need a copy of the SQL Server database software. At time of writing, you can download a trial version of SQL Server from Microsoft at <http://www.microsoft.com/sql>. You can also purchase a trial or full copy of SQL Server from Microsoft's Web site.

This book uses the Developer Edition of the SQL Server 2000 software and uses a database named Northwind. This database contains the information for the fictitious Northwind Company, which sells food products to customers. Northwind is one of the example databases that you can install with SQL Server. Customer information in the Northwind database is stored in a table named `Customers`; you'll see the use of this table in the example program later in this chapter.

If you don't want to download or purchase a trial version of SQL Server, the .NET SDK (and VS .NET) comes with a stand-alone desktop database server known as the Microsoft SQL Server 2000 Desktop Engine (MSDE 2000). MSDE 2000 has a version of the Northwind database that you can use instead of the SQL Server Northwind database—although you won't get all of the graphical administration tools that come with SQL Server. If you're using the .NET SDK and want to install MSDE 2000, select `Start > Microsoft .NET Framework SDK > Samples and QuickStart Tutorials`. If you're using VS .NET and want to install MSDE 2000, run the `setup.exe` program that you use to install VS .NET and select MSDE 2000 as a new feature to install.

NOTE You can learn more about MSDE 2000 at <http://www.microsoft.com/sql/techinfo/development/2000/msde2000.asp>.

Developing Your First ADO.NET Program

In this section you'll plunge into ADO.NET programming and see a C# program that performs the following tasks:

1. Connects to the SQL Server Northwind database
2. Retrieves a row from the `Customers` table
3. Displays the columns from the row
4. Closes the database connection

You'll be introduced to many concepts in this section that are fully explored in later chapters. Don't be too concerned about all the details of the concepts at this stage; you'll learn those details in the later chapters.

Listing 1.1 shows the example program, which is contained in the file `FirstExample.cs`.

NOTE You can download all the source files for the programs featured in this book from the Sybex Web site at www.sybex.com. You'll find instructions on downloading these files in the introduction of this book. Once you've downloaded the files, you can follow along with the examples without having to type in the program listings.

LISTING 1.1: FIRSTEXAMPLE.CS

```
/*
FirstExample.cs illustrates how to:
1. Connect to the SQL Server Northwind database.
2. Retrieve a row from the Customers table using
   a SQL SELECT statement.
3. Display the columns from the row.
4. Close the database connection.
*/

using System;
using System.Data.SqlClient;

class FirstExample
{
    public static void Main()
    {
        try
        {
            // step 1: create a SqlConnection object to connect to the
            // SQL Server Northwind database
            SqlConnection mySqlConnection =
                new SqlConnection(
                    "server=localhost;database=Northwind;uid=sa;pwd=sa"
                );

            // step 2: create a SqlCommand object
            SqlCommand mySqlCommand = mySqlConnection.CreateCommand();

            // step 3: set the CommandText property of the SqlCommand object to
            // a SQL SELECT statement that retrieves a row from the Customers table
            mySqlCommand.CommandText =
                "SELECT CustomerID, CompanyName, ContactName, Address " +
                "FROM Customers " +
                "WHERE CustomerID = 'ALFKI'";

            // step 4: open the database connection using the
            // Open() method of the SqlConnection object
            mySqlConnection.Open();
        }
    }
}
```

```

// step 5: create a SqlDataReader object and call the ExecuteReader()
// method of the SqlCommand object to run the SELECT statement
SqlDataReader mySqlDataReader = mySqlCommand.ExecuteReader();

// step 6: read the row from the SqlDataReader object using
// the Read() method
mySqlDataReader.Read();

// step 7: display the column values
Console.WriteLine("mySqlDataReader[\"CustomerID\"] = " +
    mySqlDataReader["CustomerID"]);
Console.WriteLine("mySqlDataReader[\"CompanyName\"] = " +
    mySqlDataReader["CompanyName"]);
Console.WriteLine("mySqlDataReader[\"ContactName\"] = " +
    mySqlDataReader["ContactName"]);
Console.WriteLine("mySqlDataReader[\"Address\"] = " +
    mySqlDataReader["Address"]);

// step 8: close the SqlDataReader object using the Close() method
mySqlDataReader.Close();

// step 9: close the SqlConnection object using the Close() method
mySqlConnection.Close();
}
catch (SqlException e)
{
    Console.WriteLine("A SqlException was thrown");
    Console.WriteLine("Number = " + e.Number);
    Console.WriteLine("Message = " + e.Message);
    Console.WriteLine("StackTrace:\n" + e.StackTrace);
}
}
}

```

Let's go through the lines in `FirstExample.cs`. The first set of lines is a comment that indicates what the program does:

```

/*
FirstExample.cs illustrates how to:
1. Connect to the SQL Server Northwind database.
2. Retrieve a row from the Customers table using
   a SQL SELECT statement.
3. Display the columns from the row.
4. Close the database connection.
*/

```

The next two lines indicate the namespaces being referenced in the program with the `using` statement:

```
using System;
using System.Data.SqlClient;
```

The `System` namespace is the root namespace and is referenced so that we can simply use `Console.WriteLine()` calls in the program, rather than the fully qualified `System.Console.WriteLine()` call. The `System.Data.SqlClient` namespace contains the ADO.NET classes for use with SQL Server, including the `SqlConnection`, `SqlCommand`, and `SqlDataReader` classes that are used later in the program. You'll be introduced to these classes shortly, and you'll learn the full details of the ADO.NET classes as you progress through this book.

You handle exceptions that might be thrown in your code by placing the code within a `try/catch` block. You'll notice that the nine steps are placed within a `try/catch` block in the `Main()` method, with the `catch` block handling a `SqlException` object that might be thrown by the code in the `try` block. You'll learn more about this later in the section "Handling Exceptions" after I've discussed the nine steps in the following sections.

Step 1: Create a *SqlConnection* Object to Connect to the Database

You use an object of the `SqlConnection` class to connect to a SQL Server database. Step 1 in the `Main()` method creates a `SqlConnection` object named `mySqlConnection` to connect to the SQL Server Northwind database:

```
SqlConnection mySqlConnection =
    new SqlConnection(
        "server=localhost;database=Northwind;uid=sa;pwd=sa"
    );
```

The string passed to the `SqlConnection` constructor is known as the *connection string* and contains the following elements:

server Specifies the name of the computer on which SQL Server is running—`localhost` in this example; `localhost` is a common name that refers to the computer on which your program runs. If your database is running on a computer other than the one your program is running on, then you'll need to replace `localhost` with the name of that computer.

database Specifies the name of the database—`Northwind` in this example.

uid Specifies the name of the database user account—`sa` in this example; `sa` is a common database user account used by the database administrator (DBA). You can use any database user account as long as it has access to the Northwind database.

pwd Specifies the password for the user. The password for the `sa` user in my database is also `sa`. You'll need to change `pwd` to the password for your `sa` account, or whichever account you specified in `uid`.

You'll need to change the settings of some or all of the previous elements in your connection string. You might need to speak with your DBA to get the various elements that make up your connection string. Once you have the correct values, you should make the changes to the connection string in your copy of `FirstExample.cs`.

NOTE A database administrator (DBA) is responsible for performing tasks such as installing the database software, backing up the databases, and so on.

Step 2: Create a *SqlCommand* Object

Step 2 creates a `SqlCommand` object named `mySqlCommand` that is used later to send a `SELECT` statement to the database for execution.

```
SqlCommand mySqlCommand = mySqlConnection.CreateCommand();
```

Step 3: Set the *CommandText* Property of the *SqlCommand* Object

You use SQL to work with the information stored in a database. SQL is an industry standard language supported by SQL Server, Access, and Oracle. You use the SQL `SELECT` statement for retrieving information from a database. You'll learn the basics of SQL in Chapter 3, "Introduction to the Structured Query Language."

Step 3 sets the `CommandText` property of `mySqlCommand` created in the previous step to a `SELECT` statement. This statement will retrieve the `CustomerID`, `CompanyName`, `ContactName`, and `Address` columns from the row in the `Customers` table whose `CustomerID` is `ALFKI`:

```
mySqlCommand.CommandText =  
    "SELECT CustomerID, CompanyName, ContactName, Address " +  
    "FROM Customers " +  
    "WHERE CustomerID = 'ALFKI'";
```

Step 4: Open the *SqlConnection* Object

Step 4 opens the database connection using the `Open()` method of the `SqlConnection` object created in step 1:

```
mySqlConnection.Open();
```

Once the connection to the database is open, you can send commands to the database for execution.

Step 5: Run the *SELECT* Statement

You run the `SELECT` statement previously set in `mySqlCommand` by calling the `ExecuteReader()` method. This method returns a `SqlDataReader` object that you then use to read the row data returned by the `SELECT` statement.

Step 5 creates a `SqlDataReader` object and calls the `ExecuteReader()` method of `mySqlCommand` object to run the `SELECT` statement:

```
SqlDataReader mySqlDataReader = mySqlCommand.ExecuteReader();
```


Step 6: Read the Row Using the *SqlDataReader* Object

Step 6 reads the row in *mySqlDataReader* using the *Read()* method:

```
mySqlDataReader.Read();
```

Step 7: Display the Column Values from the *SqlDataReader* Object

You can read the value for a column from *mySqlDataReader* by passing the name of the column in square brackets. For example, *mySqlDataReader["CustomerID"]* returns the value of the *CustomerID* column.

Step 7 displays the column values for the *CustomerID*, *CompanyName*, *ContactName*, and *Address* column values:

```
Console.WriteLine("mySqlDataReader[\"CustomerID\"] = " +
    mySqlDataReader["CustomerID"]);
Console.WriteLine("mySqlDataReader[\"CompanyName\"] = " +
    mySqlDataReader["CompanyName"]);
Console.WriteLine("mySqlDataReader[\"ContactName\"] = " +
    mySqlDataReader["ContactName"]);
Console.WriteLine("mySqlDataReader[\"Address\"] = " +
    mySqlDataReader["Address"]);
```

Step 8: Close the *SqlDataReader* Object

When you're finished reading rows from a *SqlDataReader* object, close it using the *Close()* method. Step 8 calls the *Close()* method for *mySqlDataReader*:

```
mySqlDataReader.Close();
```

Step 9: Close the *SqlConnection* Object

When you're finished accessing the database, close your *SqlConnection* object using the *Close()* method. Step 9 calls the *Close()* method for *mySqlConnection*:

```
mySqlConnection.Close();
```

Handling Exceptions

You handle exceptions that might be thrown in your code by placing the code within a *try/catch* block. You'll notice that the nine steps are placed within a *try/catch* block, with the catch block handling a *SqlException* object that might be thrown by the code in the *try* block. The *SqlException* class is specifically for use with code that accesses a SQL Server database.

The following example shows how to structure a *try/catch* block:

```
try
{
    /* code that might throw a SqlException */
}
catch (SqlException e)
{
}
```

```

    Console.WriteLine("A SQLException was thrown");
    Console.WriteLine("Number = " + e.Number);
    Console.WriteLine("Message = " + e.Message);
    Console.WriteLine("StackTrace:\n" + e.StackTrace);
}

```

The properties displayed in the catch block are as follows:

Number The error number

Message A string containing a description of the error

StackTrace A string containing the name of the class and the method from which the exception was thrown

The two most common examples of when a `SQLException` object is thrown are as follows:

- ◆ Your `SqlConnection` object is unable to connect to the database. If this happens, you should check the connection string that specifies how to connect to your database.
- ◆ Your `SELECT` statement contains a mistake in the spelling of a table or column.

The following example output shows what happens when the `SqlConnection` object in `FirstExample.cs` is unable to connect to the database because the database is currently down:

```

A SQLException was thrown
Number = -2
Message = Timeout expired. Possible reasons: the timeout period elapsed prior
to completion of the operation, the server is not responding,
or the maximum pool size was exceeded.
Please see the documentation for further details.
StackTrace:
    at System.Data.SqlClient.SqlConnection.Open()
    at FirstExample.Main()

```

You can use the output from your catch block to determine the problem. If the database is down, contact your DBA.

NOTE For brevity, the only program to use a try/catch block in this book is `FirstExample.cs`. You should use try/catch blocks in your own programs to catch exceptions. For more details on handling exceptions, I recommend the book *Mastering Visual C# .NET* from Sybex (2002).

In the next section you'll see how to compile `FirstExample.cs` and run it.

Compiling and Running *FirstExample.cs*

You can compile the `FirstExample.cs` program using either the command-line tool that comes with the .NET SDK or VS .NET. In this section, you'll see how to use the command-line version of the compiler for `FirstExample.cs` program. Later in this chapter, in the section "Introducing Visual Studio .NET," you'll see how to use VS .NET to compile and run a program.

You run the command-line version of the compiler by entering `csc` in the Command Prompt tool, followed by the name of your program source file. For example, to compile `FirstExample.cs`, you would enter the following command in the Command Prompt tool:

```
csc FirstExample.cs
```

If you want to follow along with the examples, start the Command Prompt tool by selecting **Start > Programs > Accessories > Command Prompt**.

NOTE *If you're using Windows XP rather than Windows 2000, start the Command Prompt tool by selecting **Start > All Programs > Accessories > Command Prompt**.*

Next, you need to change directories to where you copied the `FirstExample.cs` file. To do this, you first enter the partition on your hard disk where you saved the file. For example, let's say you saved the file in the `ADO.NET\book\ch01\programs` directory of the C partition of your hard disk. To access the C partition, you enter the following line into the Command Prompt tool and then you press the Enter key:

```
C:
```

Next, to move to the `ADO.NET\book\ch01\programs` directory, you enter `cd` followed by `ADO.NET\book\ch01\programs`:

```
cd ADO.NET\book\ch01\programs
```

To compile `FirstExample.cs` using `csc`, you enter the following command:

```
csc FirstExample.cs
```

Notice that the name of the program source file follows `csc`; in this case, it's `FirstExample.cs`.

If you get an error when running `csc`, you'll need to add the directory where you installed the SDK to your Path environment variable. The Path environment variable specifies a list of directories that contain executable programs. Whenever you run a program from the command prompt, the directories in the Path variable are searched for the program you want to run. Your current directory is also searched. To set your Path environment variable, do the following:

1. Select **Start > Settings > Control Panel**. Then double-click **System** and select the **Advanced** tab.
2. Click the **Environment Variables** button and double-click **Path** from the system variables area at the bottom.
3. Add the directory where you installed the SDK to your Path environment variable.
4. Click **OK** to save your change, and then click **OK** again on the next dialog.
5. Restart Command Prompt so that your change is picked up. You should then be able to run `csc` successfully.

The compiler takes the `FirstExample.cs` file and compiles it into an executable file named `FirstExample.exe`. The `.exe` file contains instructions that a computer can run, and the `.exe` file extension indicates the file is an executable file.

You run an executable file using the Command Prompt tool by entering the name of that executable file. For example, to run the `FirstExample.exe` file, you enter the following line in the Command Prompt tool and then you press the Enter key:

```
FirstExample
```

When you run the program, you should see the following text displayed in your Command Prompt window:

```
mySqlDataReader["CustomerID"] = ALFKI
mySqlDataReader["CompanyName"] = Alfreds Futterkiste
mySqlDataReader["ContactName"] = Maria Anders
mySqlDataReader["Address"] = Obere Str. 57
```

If you encounter an exception—such as your program can't connect to the database—you should check the connection string set in step 1 of `FirstExample.cs`, and speak with your DBA if necessary.

Connecting to Access and Oracle Databases

In this section you'll see examples of connecting to both an Access and an Oracle database. To interact with either of these databases in your program, you use classes from the `System.Data.OleDb` namespace. This namespace contains classes for use with databases that support object linking and embedding for databases (OLE DB) such as Access or Oracle. You'll learn more about the `System.Data.OleDb` namespace in Chapter 5, "Overview of the ADO.NET Classes."

Connecting to an Access Database

You connect to an Access database using an `OleDbConnection` object—rather than a `SqlConnection` object—with a connection string of the following format:

```
provider=Microsoft.Jet.OLEDB.4.0;data source=databaseFile
```

where *databaseFile* is the directory and filename of your Access database. Notice that you specify the provider in the connection string, which is set to `Microsoft.Jet.OLEDB.4.0`.

The following example creates a string named `connectionString` with the appropriate format to connect to the Access Northwind database stored in the `Northwind.mdb` file:

```
string connectionString =
    "provider=Microsoft.Jet.OLEDB.4.0;" +
    "data source=F:\\Program Files\\Microsoft" +
    "Office\\Office\\Samples\\Northwind.mdb";
```

NOTE Notice the use of two backslash characters in the `data source` part of the connection string. The first backslash is used to specify that the second backslash is to be treated literally; therefore `\\` is treated as `\` in the connection string. You'll need to locate the `Northwind.mdb` file on your hard disk and set your connection string appropriately.

Assuming the `System.Data.OleDb` namespace has been imported, the following example creates an `OleDbConnection` object, passing `connectionString` (set in the previous line of code) to the constructor:

```
OleDbConnection myOleDbConnection =
    new OleDbConnection(connectionString);
```

Listing 1.2 illustrates how to connect to the Northwind Access database using an `OleDbConnection` object and retrieve a row from the `Customers` table. Notice that you use an `OleDbCommand` and `OleDbDataReader` object to run a SQL statement and read the returned results from an Access database.

LISTING 1.2: OLEDBCONNECTIONACCESS.CS

```
/*
 OleDbConnectionAccess.cs illustrates how to use an
 OleDbConnection object to connect to an Access database
 */

using System;
using System.Data;
using System.Data.OleDb;

class OleDbConnectionAccess
{
    public static void Main()
    {
        // formulate a string containing the details of the
        // database connection
        string connectionString =
            "provider=Microsoft.Jet.OLEDB.4.0;" +
            "data source=F:\\Program Files\\Microsoft
Office\\Office\\Samples\\Northwind.mdb";

        // create an OleDbConnection object to connect to the
        // database, passing the connection string to the constructor
        OleDbConnection myOleDbConnection =
            new OleDbConnection(connectionString);

        // create an OleDbCommand object
        OleDbCommand myOleDbCommand = myOleDbConnection.CreateCommand();

        // set the CommandText property of the OleDbCommand object to
        // a SQL SELECT statement that retrieves a row from the Customers table
        myOleDbCommand.CommandText =
            "SELECT CustomerID, CompanyName, ContactName, Address " +
            "FROM Customers " +
            "WHERE CustomerID = 'ALFKI'";
```

```

// open the database connection using the
// Open() method of the OleDbConnection object
myOleDbConnection.Open();

// create an OleDbDataReader object and call the ExecuteReader()
// method of the OleDbCommand object to run the SELECT statement
OleDbDataReader myOleDbDataReader = myOleDbCommand.ExecuteReader();

// read the row from the OleDbDataReader object using
// the Read() method
myOleDbDataReader.Read();

// display the column values
Console.WriteLine("myOleDbDataReader[\"CustomerID\"] = " +
    myOleDbDataReader["CustomerID"]);
Console.WriteLine("myOleDbDataReader[\"CompanyName\"] = " +
    myOleDbDataReader["CompanyName"]);
Console.WriteLine("myOleDbDataReader[\"ContactName\"] = " +
    myOleDbDataReader["ContactName"]);
Console.WriteLine("myOleDbDataReader[\"Address\"] = " +
    myOleDbDataReader["Address"]);

// close the OleDbDataReader object using the Close() method
myOleDbDataReader.Close();

// close the OleDbConnection object using the Close() method
myOleDbConnection.Close();
}
}

```

The output from this program is as follows:

```

myOleDbDataReader["CustomerID"] = ALFKI
myOleDbDataReader["CompanyName"] = Alfreds Futterkiste
myOleDbDataReader["ContactName"] = Maria Anders
myOleDbDataReader["Address"] = Obere Str. 57

```

Connecting to an Oracle Database

You connect to an Oracle database using an `OleDbConnection` object with a connection string of the following format:

```

provider=MSDAORA;data source=OracleNetServiceName;user
id=username;password=password

```

where

OracleNetServiceName Specifies the Oracle Net service name for the database. Oracle Net is a software component that allows you to connect to a database over a network. You'll need to speak with your DBA to get the Oracle Net service name.

username Specifies the name of the database user you want to connect to the database as.

password Specifies the password for the database user.

The following example creates a connection string named `connectionString` with the correct format to connect to an Oracle database:

```
string connectionString =
    "provider=MSDAORA;data source=ORCL;user id=SCOTT;password=TIGER";
```

NOTE The user ID of SCOTT with a password of TIGER is the default for accessing one of the example databases that comes with Oracle. This database contains a table called `emp` that contains sample employee data.

Assuming the `System.Data.OracleClient` namespace has been imported, the following example creates an `OracleConnection` object, passing `connectionString` to the constructor:

```
OracleConnection myOracleConnection =
    new OracleConnection(connectionString);
```

Listing 1.3 illustrates how to connect to an Oracle database using an `OracleConnection` object and retrieve a row from the `emp` table. Notice that you use an `OracleCommand` and `OracleDataReader` object to run a SQL statement and read the returned results from an Oracle database.

LISTING 1.3: ORACLECONNECTIONORACLE.CS

```
/*
    OracleConnectionOracle.cs illustrates how to use an
    OracleConnection object to connect to an Oracle database
*/

using System;
using System.Data;
using System.Data.OracleClient;

class OracleConnectionOracle
{
    public static void Main()
    {
        // formulate a string containing the details of the
        // database connection
        string connectionString =
            "provider=MSDAORA;data source=ORCL;user id=SCOTT;password=TIGER";
```

```

// create an OleDbConnection object to connect to the
// database, passing the connection string to the constructor
OleDbConnection myOleDbConnection =
    new OleDbConnection(connectionString);

// create an OleDbCommand object
OleDbCommand myOleDbCommand = myOleDbConnection.CreateCommand();

// set the CommandText property of the OleDbCommand object to
// a SQL SELECT statement that retrieves a row from the emp table
myOleDbCommand.CommandText =
    "SELECT empno, ename, sal " +
    "FROM emp " +
    "WHERE empno = 7369";

// open the database connection using the
// Open() method of the SqlConnection object
myOleDbConnection.Open();

// create an OleDbDataReader object and call the ExecuteReader()
// method of the OleDbCommand object to run the SELECT statement
OleDbDataReader myOleDbDataReader = myOleDbCommand.ExecuteReader();

// read the row from the OleDbDataReader object using
// the Read() method
myOleDbDataReader.Read();

// display the column values
Console.WriteLine("myOleDbDataReader[\"empno\"] = " +
    myOleDbDataReader["empno"]);
Console.WriteLine("myOleDbDataReader[\"ename\"] = " +
    myOleDbDataReader["ename"]);
Console.WriteLine("myOleDbDataReader[\"sal\"] = " +
    myOleDbDataReader["sal"]);

// close the OleDbDataReader object using the Close() method
myOleDbDataReader.Close();

// close the OleDbConnection object using the Close() method
myOleDbConnection.Close();
}
}

```

The output from this program is as follows:

```
myOleDbDataReader["empno"] = 7369
myOleDbDataReader["ename"] = SMITH
myOleDbDataReader["sal"] = 800
```

Introducing Visual Studio .NET

In the previous sections, you saw programs that connect to various databases, retrieve a row from a table, and display the column values for that row on your computer screen. This type of program is known as a *console application* because it displays output directly on the screen on which the program is running.

You can use Visual Studio .NET (VS .NET) to create console applications, as well as the following types of applications:

Windows Applications These take advantage of the visual controls offered by the Windows operating system, such as menus, buttons, and editable text boxes. Windows Explorer, which you use to navigate the file system of your computer, is one example. You'll learn about Windows programming in Chapter 6, "Introducing Windows Applications and ADO.NET."

ASP.NET Applications These run over the Internet. You access an ASP.NET application using a Web browser, such as Internet Explorer. Examples of ASP.NET applications would be online banking, stock trading, or auction systems. You'll learn about ASP.NET programming in Chapter 15, "Introducing Web Applications: ASP.NET."

ASP.NET Web Services These also run over the Internet. Also known as XML Web services, the difference is that you can use them to offer a service that could be used in a distributed system of interconnected services. For example, Microsoft's Passport Web service offers identification and authentication of Web users you could then use in your own Web application. You'll learn about Web services in Chapter 17, "Web Services."

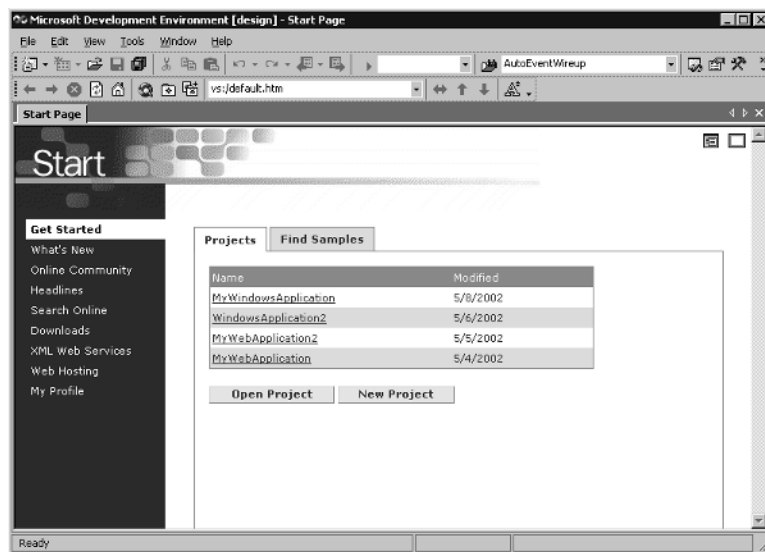
This is not an exhaustive list of the types of applications you can develop with VS .NET, but it does give you flavor for the broad range of VS .NET's capabilities.

In the rest of this section, you'll see how to develop and run a console application using VS .NET. If you've installed VS .NET on your computer, you'll be able to follow along with the example. If you don't have VS .NET, don't worry; you'll still be able to see what's going on from the figures provided.

Starting Visual Studio .NET and Creating a Project

All of your work in VS .NET is organized into *projects*. Projects contain the source and executable files for your program, among other items. If you have VS .NET installed, start it by selecting Start ➤ Programs ➤ Microsoft Visual Studio .NET ➤ Microsoft Visual Studio .NET. Once VS .NET has started, you'll see the Start page (see Figure 1.1).

FIGURE 1.1
The Start page



From the Start page, you can see any existing projects you've created. You can open and create projects using the Open Project and New Project buttons, respectively. You'll create a new project shortly.

USING THE VS .NET LINKS

As you can see from Figure 1.1, VS .NET contains a number of links on the left of the Start page. Some of these links provide access to useful information on the Internet about .NET; the links are as follows:

Get Started Open the Start page.

What's New View any updates for VS .NET or Windows. You can also view upcoming training events and conferences.

Online Community Get in touch with other members of the .NET community. Includes links to Web sites and newsgroups.

Headlines View the latest news on .NET.

Search Online Search the MSDN Online Library for technical material such as published articles on .NET.

Downloads Download trial applications and example programs from the Web sites featured here.

XML Web Services Find registered XML Web services that you can then use in your own programs. XML Web services are also known as ASP.NET Web services. You'll learn more about Web services in Chapter 17.

Web Hosting A Web hosting company can take your program and run it for you. It takes care of the computers on which your program runs. Use the Web Hosting link to view companies that provide these services.

My Profile Set items such as your required keyboard scheme and window layout.

Click these links and explore the information provided. As you'll see, there's a lot of information about .NET on the Internet.

CREATING A NEW PROJECT

When you're finished examining the information in the previous links, create a new project by clicking the New Project button on the Get Started page.

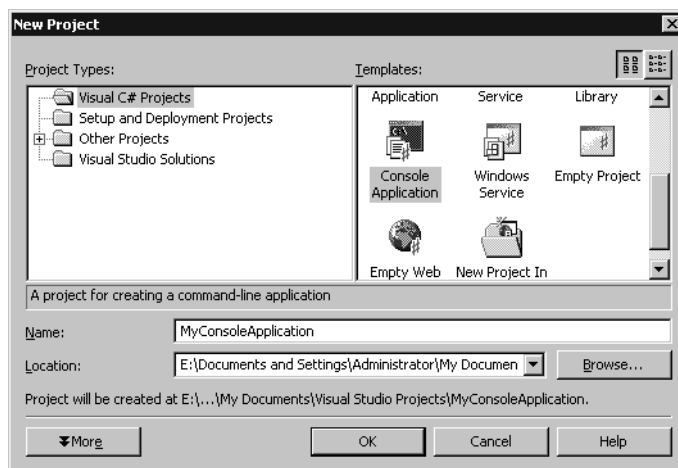
NOTE You can also create a new project by selecting *File > New > Project*, or by pressing *Ctrl+Shift+N* on your keyboard.

When you create a new project, VS .NET displays the New Project dialog box, which you use to select the type of project you want to create. You also enter the name and location of your new project; the location is the directory where you want to store the files for your project.

Because you're going to be creating a C# console application, select Visual C# Projects from the Project Types section on the left of the New Project dialog box, and select Console Application from the Templates section on the right. Enter **MyConsoleApplication** in the Name field, and keep the default directory in the Location field. Figure 1.2 shows the completed New Project dialog box with these settings.

FIGURE 1.2

The New Project dialog box with the appropriate settings for a C# console application

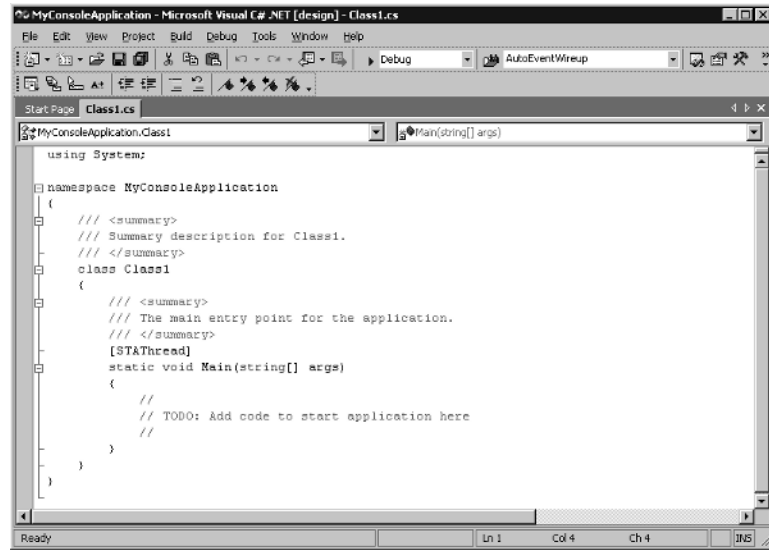


Click the OK button to create the new project.

WORKING IN THE VS .NET ENVIRONMENT

Once you've created a new project, the main development screen is displayed (see Figure 1.3). This screen is the environment in which you'll develop your project. As you can see, VS .NET has already created some starting code for you. This code is a skeleton for your program; you'll see how to modify it shortly. In this section, I'll give you a brief description of the different parts of the VS .NET environment.

FIGURE 1.3
The VS .NET
environment



NOTE Depending on your settings for VS .NET, your screen might look slightly different from that shown in Figure 1.3.

The VS .NET menu contains the following items:

File Open, close, and save project files.

Edit Cut, copy, and paste text from the Clipboard. The Clipboard is a temporary storage area.

View Hide and show different windows such as the Solution Explorer (which lets you see the files that make up your project), Class View (which lets you see the classes and objects in your project), Server Explorer (which lets you explore items such as databases), and the Properties window (which lets you set the properties of objects, such as the size of a button). You can also use the View menu to select the toolbars you want to display.

Project Add class files to your project and add Windows forms and controls.

Build Compile the source files in your project.

Debug Start your program with or without debugging. Debugging lets you step through your program line by line, looking for errors.

Tools Connect to a database and customize your settings for VS .NET. For example, set the colors used for different parts of your program lines or set the initial page displayed by VS .NET when you start it.

Window Switch between files you've opened and hide windows.

Help Open the documentation on .NET. You'll learn how to use this documentation later in this chapter in the section "Using the .NET Documentation."

The VS .NET toolbar contains a series of buttons that act as shortcuts to some of the menu options. For example, you can save a file or all files, cut and paste text from the Clipboard, and start a program using the debugger. You'll learn how to use some of these features later in this chapter.

The code shown in the window (below the toolbar) with the title `Class1.cs` is code that is automatically generated by VS .NET, and in the next section you'll modify this code.

MODIFYING THE VS .NET-GENERATED CODE

Once VS .NET has created your project, it will display some starting code for the console application with a class name of `Class1.cs`. You can use this code as the beginning for your own program. Figure 1.3, shown earlier, shows the starting code created by VS .NET.

The `Main()` method created by VS .NET is as follows:

```
static void Main(string[] args)
{
    //
    // TODO: Add code to start application here
    //
}
```

As you can see, this code contains comments that indicate where you add your own code. Replace the `Main()` method with the following code taken from the `Main()` method in `FirstExample.cs`, shown earlier in Listing 1.1:

```
public static void Main()
{
    try
    {
        // step 1: create a SqlConnection object to connect to the
        // SQL Server Northwind database
        SqlConnection mySqlConnection =
            new SqlConnection(
                "server=localhost;database=Northwind;uid=sa;pwd=sa"
            );

        // step 2: create a SqlCommand object
        SqlCommand mySqlCommand = mySqlConnection.CreateCommand();

        // step 3: set the CommandText property of the SqlCommand object to
        // a SQL SELECT statement that retrieves a row from the Customers table
```

```

mySqlCommand.CommandText =
    "SELECT CustomerID, CompanyName, ContactName, Address " +
    "FROM Customers " +
    "WHERE CustomerID = 'ALFKI'";

// step 4: open the database connection using the
// Open() method of the SqlConnection object
mySqlConnection.Open();

// step 5: create a SqlDataReader object and call the ExecuteReader()
// method of the SqlCommand object to run the SELECT statement
SqlDataReader mySqlDataReader = mySqlCommand.ExecuteReader();

// step 6: read the row from the SqlDataReader object using
// the Read() method
mySqlDataReader.Read();

// step 7: display the column values
Console.WriteLine("mySqlDataReader[\"CustomerID\"] = " +
    mySqlDataReader["CustomerID"]);
Console.WriteLine("mySqlDataReader[\"CompanyName\"] = " +
    mySqlDataReader["CompanyName"]);
Console.WriteLine("mySqlDataReader[\"ContactName\"] = " +
    mySqlDataReader["ContactName"]);
Console.WriteLine("mySqlDataReader[\"Address\"] = " +
    mySqlDataReader["Address"]);

// step 8: close the SqlDataReader object using the Close() method
mySqlDataReader.Close();

// step 9: close the SqlConnection object using the Close() method
mySqlConnection.Close();
}
catch (SqlException e)
{
    Console.WriteLine("A SqlException was thrown");
    Console.WriteLine("Number = " + e.Number);
    Console.WriteLine("Message = " + e.Message);
    Console.WriteLine("StackTrace:\n" + e.StackTrace);
}
}

```

NOTE You'll also need to add the following line near the start of your class: `using System.Data.SqlClient;`

Once you've added the previous code, your next steps are to compile and run your program.

Compiling and Running the Program Using VS .NET

As always, you must first compile your program before you can run it. Because programs in VS .NET are organized into projects, you must compile the project; this is also known as *building* the project. To build your project, select Build ➤ Build Solution. This compiles the `Class1.cs` source file into an executable file.

TIP You can also press `Ctrl+Shift+B` on your keyboard to build your project.

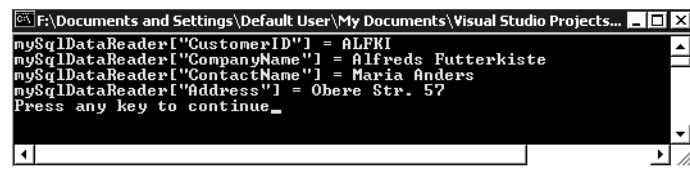
Finally, you can now run your program. Select Debug ➤ Start Without Debugging. When you select Start Without Debugging, the program will pause at the end, allowing you to view the output.

TIP You can also press `Ctrl+F5` on your keyboard to run your program.

When you run your program, VS .NET will run the program in a new Command Prompt window, as shown in Figure 1.4. Your program is run in this window because it is a console application.

FIGURE 1.4

The running program



```
F:\Documents and Settings\Default User\My Documents\Visual Studio Projects...
mySqlDataReader["CustomerID"] = ALFKI
mySqlDataReader["CompanyName"] = Alfreds Futterkiste
mySqlDataReader["ContactName"] = Maria Anders
mySqlDataReader["Address"] = Obere Str. 57
Press any key to continue_
```

To end the program, press any key. This will also close the Command Prompt window.

You've barely scratched the surface of VS .NET in this section. You'll explore some of the other features of VS .NET later in this book. In the next section, you'll learn how to use the extensive documentation that comes with .NET.

Using the .NET Documentation

Both the .NET SDK and VS .NET come with extensive documentation, including the full reference to all the classes in .NET. As you become proficient with C#, you'll find this reference documentation invaluable.

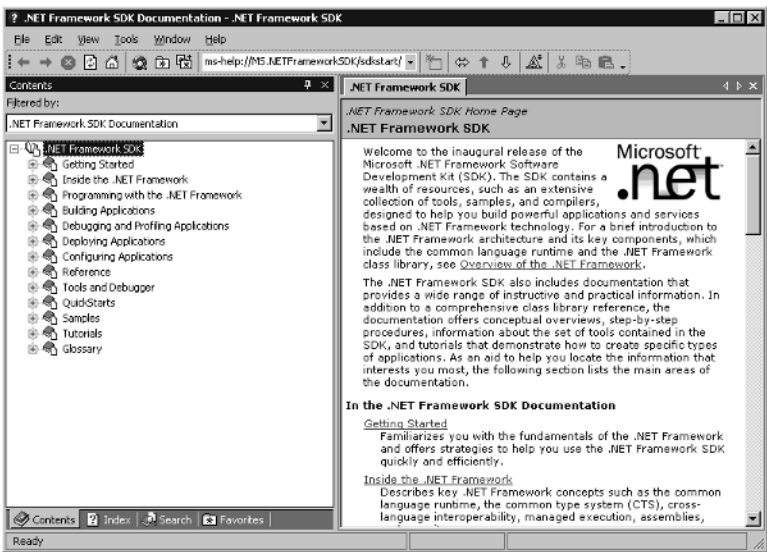
In the following sections, you'll see how to access and search the .NET documentation, and view some of the contents of the documentation. Depending on whether you're using the .NET SDK or VS .NET, you access the documentation in a slightly different way. You'll see how to use both ways to access the documentation in this section.

NOTE The documentation that comes with the .NET SDK is a subset of the documentation that comes with VS .NET.

Accessing the Documentation Using the .NET SDK

If you're using the .NET SDK, you access the documentation by selecting Start ➤ Programs ➤ Microsoft .NET Framework SDK ➤ Documentation. Figure 1.5 shows the .NET Framework SDK document home page; this is the starting page for the documentation.

FIGURE 1.5
The documentation
home page

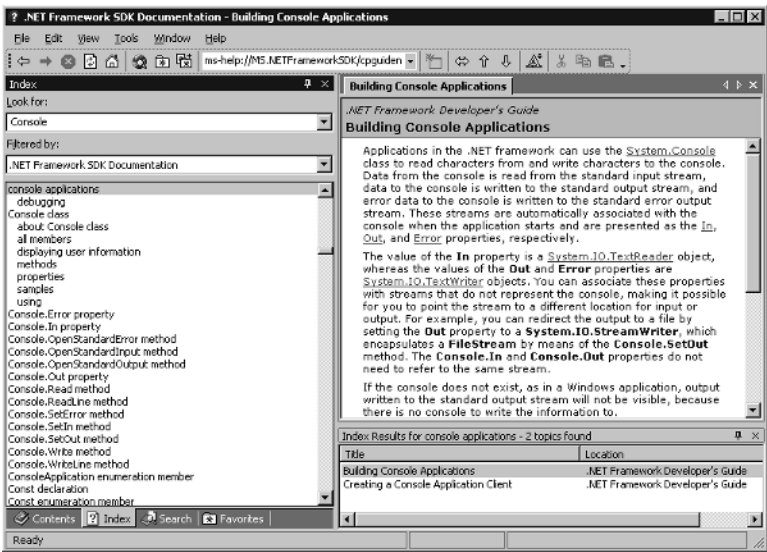


On the left of the page, you can see the various sections that make up the contents of the documentation. You can view the index of the documentation by selecting the Index tab at the bottom of the page.

TIP You can also view the Index window by selecting *Help > Index*, or by pressing *Ctrl+Alt+F2* on your keyboard.

You can search the index by entering a word in the Look For field of the Index tab. Figure 1.6 shows the results of searching for *Console*. Figure 1.6 also shows the text for the details on building console applications on the top right of the screen. I opened this overview by double-clicking the Building Console Applications link in the Index Results on the bottom right of the screen.

FIGURE 1.6
Searching the index
for the word *console*



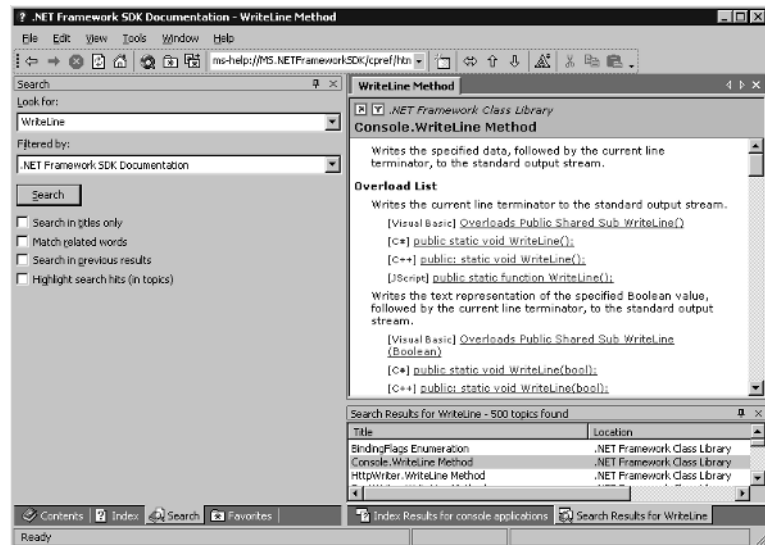
You can also search all pages in the documentation using the Search tab. You display the Search tab by selecting it from the bottom of the screen.

TIP You can also view the Search window by selecting *Help > Search*, or by pressing *Ctrl+Alt+F3* on your keyboard.

You enter the words you want to search for in the Look For field of the Search window. Figure 1.7 shows the search page and the search results returned by a search for *WriteLine*. When you run the search, the names of the pages that contain your required words are displayed in the Search Results window that appears at the bottom of the screen (you can see this window in Figure 1.7).

FIGURE 1.7

Searching all of the documentation for the word *WriteLine*



TIP You can also view the Search Results window by selecting *Help > Search results*, or by pressing *Shift+Alt+F3* on your keyboard.

You view the contents of a particular page shown in the Search Results window by double-clicking the appropriate line. For example, in Figure 1.7, we double-clicked the second line in the Search Results window. This line contained the page with the title “Console.WriteLine Method,” and as you can see, this page is displayed in the window above the Search Results in Figure 1.7.

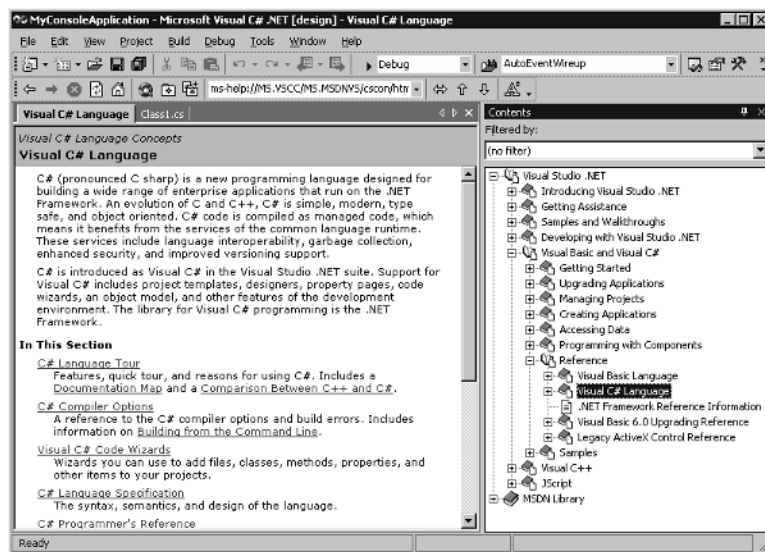
In the next section, you’ll see how to access the documentation using VS .NET.

Accessing the Documentation Using VS .NET

If you’re using VS .NET, you access the documentation using the Help menu. To access the contents of the documentation, you select *Help > Contents*. Figure 1.8 shows the contents displayed in VS .NET. Notice that the documentation is displayed directly in VS .NET, rather than in a separate window, as is done when viewing documentation with the .NET SDK.

FIGURE 1.8

The documentation contents viewed in VS.NET



NOTE The same keyboard shortcuts shown in the previous section also apply to VS.NET.

The Help menu also provides access to similar Index and Search windows as you saw in the previous section.

Using the SQL Server Documentation

SQL Server also comes with extensive electronic documentation. To access this documentation, you select Start > Programs > Microsoft SQL Server > Books Online. Figure 1.9 shows the SQL Server documentation home page.

You can browse the online books using the Contents tab, and you can search for specific information using the Index and Search tabs. Figure 1.10 shows some of the information for the SELECT statement, which is located in the Transact-SQL reference book.

NOTE Transact-SQL is Microsoft's implementation of SQL and contains programming extensions. You'll learn about Transact-SQL programming in Chapter 4.

You can see the information shown in Figure 1.10 yourself by opening Contents > Transact-SQL Reference > SELECT > SELECT Examples.

FIGURE 1.9
SQL Server
documentation
home page

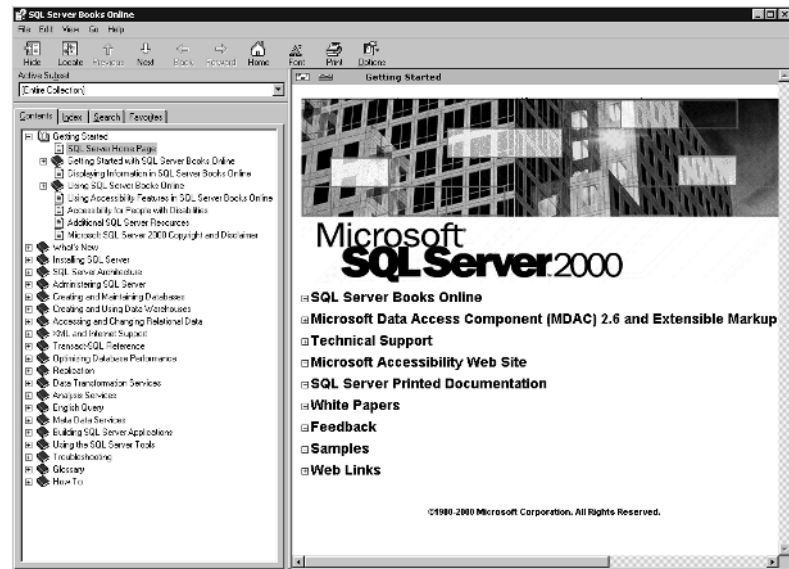
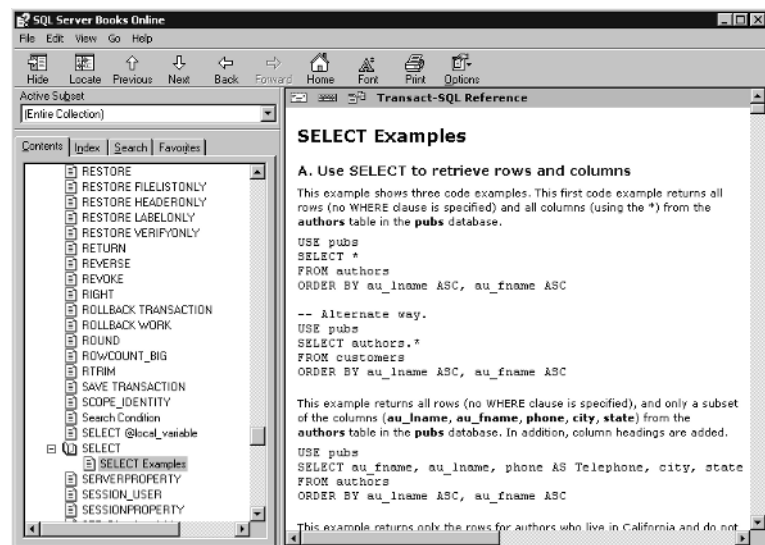


FIGURE 1.10
SELECT examples
documentation



Summary

A *database* is an organized collection of information that is divided into *tables*. Each table is further divided into *rows* and *columns*; these columns store the actual information. You access a database using the *Structured Query Language* (SQL), which is a standard language supported by most database software including SQL Server, Access, and Oracle.

You saw a C# program that connected to a SQL Server database, retrieved and displayed the contents stored in the columns of a row from a table, and then disconnected from the database. You also saw programs that connected to an Access and an Oracle database.

Microsoft's Rapid Application Development (RAD) tool is Visual Studio .NET (VS .NET). VS .NET enables you to develop and run programs in an integrated development environment. This environment uses all the great features of Windows, such as the mouse and intuitive menus, and increases your productivity as a programmer.

In the final sections of this chapter, you saw how to use the extensive documentation from Microsoft that comes with the .NET Software Development Kit (SDK) and VS .NET. You also saw how to use the SQL Server documentation.

In the next chapter, you'll learn more about databases.