Visual Studio.NET Database Projects

CHAPTER

8

IN THIS CHAPTER

- Creating a Database Project 294
- Database References 296
- Scripts 297
- Queries 312

The database project is a special type of Visual Studio.NET project. Its purpose is to create and manage SQL database scripts. If you're developing database applications with Visual Studio.NET, you will want to know about the tools available for making your work with databases easier and faster. This version of Visual Studio features many new tools and contains significant improvements to others that existed in previous versions.

The Visual Database tools allow us to view, design, modify, and test database objects (for example, tables, views, queries, stored procedures, and so on) quickly without having to jump from the Visual Studio environment to a different toolset. The main advantage is in design and development productivity, although licensing and installation issues also have been greatly simplified.

We have already worked with some of these tools in previous chapters, and their use is usually quite intuitive. They should also seem very familiar to you if you've had experience using similar tools (even MS Access qualifies). If this is your first time working with such tools, the Visual Studio help topics will be useful.

In this chapter we continue using some of the Visual Database tools to demonstrate how to utilize the tools and features available in a VS.NET database project.

Creating a Database Project

A special type of VS.NET project of particular interest is the database project. A database project is not specific to any .NET programming language, such as Visual Basic or C#, but is meant to be used to design, test, run, and manage SQL scripts and queries. When you design your application in multiple layers, as you should, this project type helps you to develop and manage the database layer closest to the database and the database itself.

In some applications, a database project may be part of the actual application code, and in other applications, it may be part of a separate administration solution used to set up and maintain the application's database.

To add a new database project to a solution, do the following.

1. Launch the Add New Project dialog from either the main File menu or from the context menu displayed by right-clicking on the solution in the Solution Explorer.
2. The left panel of this dialog box displays a list of folders containing different project types. Expand the Other Projects folder (click on the "+").
3. Select the Database Projects folder. It displays a Database Project template in the right panel of the dialog box.
4. Specify a project name of NoveltyData and a path for saving the project files and then click on the OK button.

5. If you don't currently have any database connections defined in the Server Explorer, the Data Link Properties dialog box will be displayed, allowing you to define a new database connection.
6. If you have at least one database connection defined in the Server Explorer, the Add Database Reference dialog box is displayed. From this dialog choose the database connection that you want to use (you can add new ones later). Alternatively, you can click on the Add New Reference button to display the Data Link Properties dialog box, allowing you to define a new database connection.
7. Select the connection to the Novelty database on the SQL Server and click on the OK button. If for some reason the connection doesn't exist, create it from the Data Link Properties dialog box.
8. Figure 8.1 shows the project and its folders in the Solution Explorer.

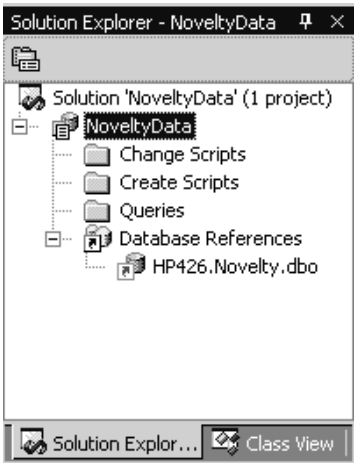

Figure 8.1 *The NoveltyData database project shown in the Solution Explorer*

Note that the created database project contains the following folders:

- Change Scripts
- Create Scripts
- Queries
- Database References

Let's take at look at these folders and what they contain. We start with Database References because they're the prerequisite for everything else.

Database References

A database reference is a pointer to a database. However, this reference doesn't allow you to access and view the database objects—that's the job of the database connection.

A database reference is stored as part of the database project that is saved to disk. Whenever the project is opened, the project scans the list of database connections currently defined in the Server Explorer to see if a connection to the referenced database already exists. If such a connection doesn't exist, the database project automatically creates one.

The first database reference added to a project becomes the default reference for the project. That is, unless specified otherwise, the scripts and queries that will be designed and/or run will be against that database reference.

NOTE

The icon for a database reference is a shortcut icon on top of a database icon. For the default database reference, the shortcut icon is red, green, and white. For the other references, the shortcut icon is black and white.

You can change the default reference for the project by right-clicking on the reference that you would like to be the default and selecting Set as Project Default from the context menu displayed.

You can add database references by right-clicking on the Database References node and selecting New Database Reference from the context menu displayed.

TIP

You can add another database reference and make it the default all at once. Right-click on the database project name (main node) in the Solution Explorer and then select the Set Default Reference menu item from the displayed context menu. Doing so displays the Set Default Reference dialog box. If you choose a database connection from the list of connections, it is added to the project and set to be the default database reference.

Scripts

The database project template automatically creates two folders for holding SQL scripts. The Create Scripts folder is meant to hold scripts that are used to re-create the database from scratch, or to re-create a portion of the database that has undergone changes.

The Change Scripts folder is meant to contain SQL scripts that reflect desired changes that you haven't yet made to the database. Changes may be "placed on hold" because, as a developer, you don't have sufficient security privileges to modify a production database (sometimes not a bad idea!) or because changes from multiple sources are to be combined and applied all at once as a unit.

NOTE

The folder names given are by convention only. You can change them to something different that you prefer or find more meaningful. Of course, being able to change them also means that you have the flexibility to mess things up, or at least confuse others (or even yourself) who need to use your project.

You can also add other folders to a database project. For example, you may want a separate folder to store scripts required for upgrading the database to a specific release version or for other maintenance activities. You may want to have several different query folders, each containing different types of queries. You can add a new folder to the project by right-clicking on the project and selecting the New Folder menu item from the context menu that is displayed.

You can create any SQL script manually, by right-clicking on a folder (or the project) node in the Solution Explorer and then selecting the Add New Item or the Add SQL Script menu item from the displayed context menu. After selecting one of the standard script templates, shown in Figure 8.2, you can edit it manually in the Visual Studio editor.

What is special about the Create and Change scripts is that you can have them generated automatically.

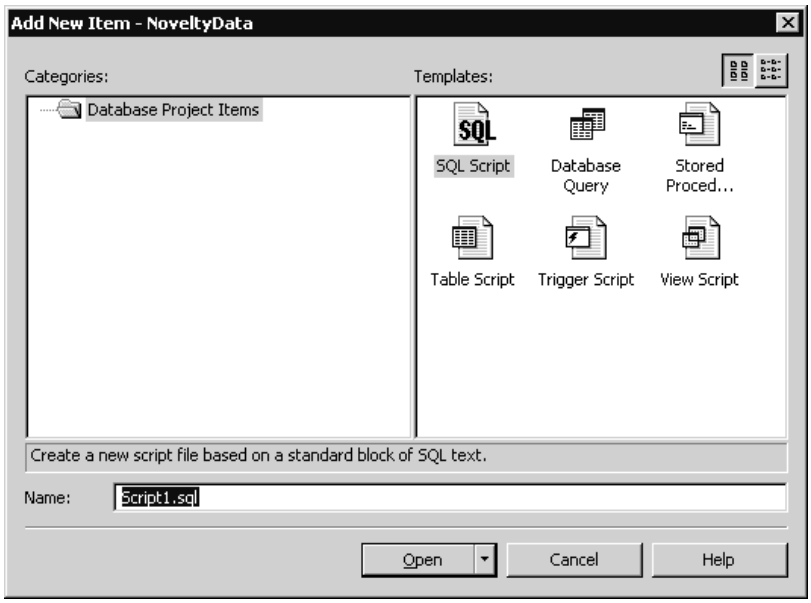

Figure 8.2 *The standard script templates displayed in the Add New Item dialog box*

Create Scripts

As we said previously, Create Scripts are SQL scripts that create new database objects, including tables, views, stored procedures, and constraints. They are normally used to set up an installation or revert an existing site (such as a development server) to its pristine state.

NOTE

You can generate Create Scripts only if you are using SQL Server 7.0 or SQL Server 2000. Moreover, to use this feature, the client tools for SQL Server must be installed on the same machine as Visual Studio. The reason is that Visual Studio utilizes the same tools and dialogs.

To generate Create Scripts, do the following.

1. Open the Server Explorer and right-click on the item for which you want to generate the script. This item can be either the entire database or an individual database object (table, view, stored procedure, or function). For this example, select the entire Novelty database.

2. Select the Generate Create Script menu item from the context menu displayed. After you successfully enter your security credentials in the SQL Server Login dialog box, the Generate Create Scripts dialog box is displayed.
3. If you select a specific database object, the Generate Create Scripts dialog box appears, configured for only that object. If you select the entire database, as in Figure 8.3, the dialog box

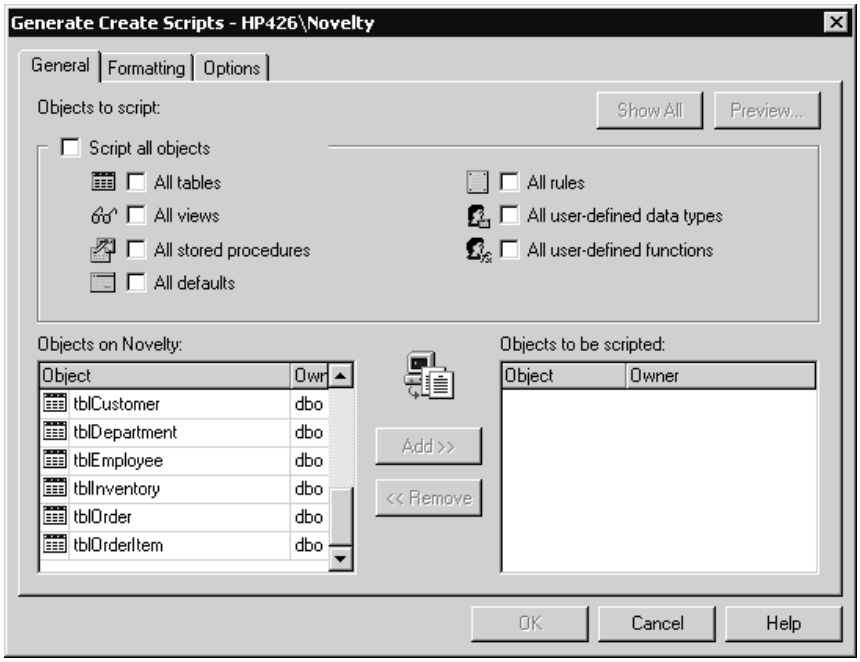

Figure 8.3 *The Generate Create Scripts dialog box for the entire Novelty database*

TIP

You can also display the Generate Create Scripts dialog box by dragging and dropping from the Server Explorer to a folder in the Solution Explorer. Here, too, you can drag either the entire database or individual database objects. If you select one or more individual objects, such as a table and its relevant stored procedures, the Generate Create Scripts dialog appears configured for just the selected object(s). However, if you select the entire database, or even a single folder within the database, such as the Tables or Views folder, it in fact behaves as if you dragged the entire database. If you want to script more than just a single object, you can also start by right-clicking on the Create Scripts folder and selecting the Generate Create Script menu item from there.

appears, showing all the objects available on the database but without any of them selected for scripting. This result is shown in Figure 8.3.

4. On the General tab, you can specify which objects are to be included in the generated script by selecting one or more objects from the Objects on the Novelty panel and adding them to the Objects to be scripted panel by either double-clicking on an object or the Add button. Entire groups of objects, such as all tables or all views (or even all the database objects), can be specified by checking one or more of the checkboxes in the top portion of the tab. For this example, check the Script all objects checkbox.
5. Select the Formatting tab. In addition to the default settings, check the Include descriptive headers in the script files checkbox.
6. Click on the OK button. Doing so displays the Browse for Folder dialog box so that you can specify where to save the script file(s). Note that it defaults to the Create Scripts directory of the current project, but you can change this setting to whatever you want. Accept the default by clicking on the OK button.
7. Because we had you choose to script all the database objects, many script files are created and added to the project, as shown in Figure 8.4. We could have had you choose to have all the scripts in a single file by selecting the Create one file option on the Options tab of the Generate Create Scripts dialog box.

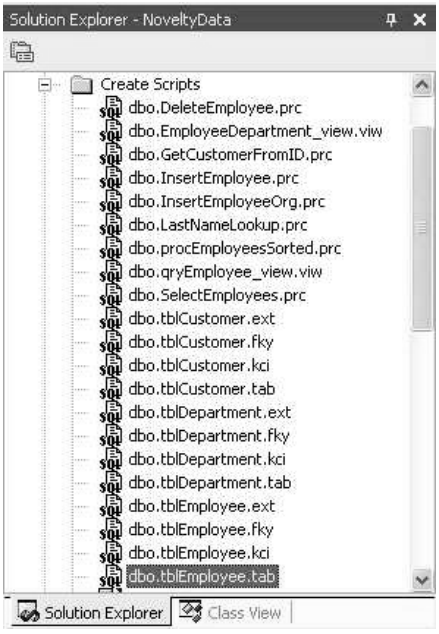

Figure 8.4 *The Solution Explorer filled with Create Scripts for the Novelty database*

TIP

You can even generate the script for creating the database itself by checking the Script database option on the Options tab of the Generate Create Scripts dialog box. Our database project, NoveltyData, now contains a set of scripts that we can run to create all the database objects for the Novelty database. In the Running the Scripts section, we show how to do so.

8. You can view (and modify) the contents of a script file by double-clicking on it in the Solution Explorer. Figure 8.5 shows the script dbo.tblOrder.tab that creates the tblOrder table in the Novelty database.

Note, however, that these scripts create only the database schema and do not populate the newly created tables with any data. In the Command Files section, we show how to copy a table's data.

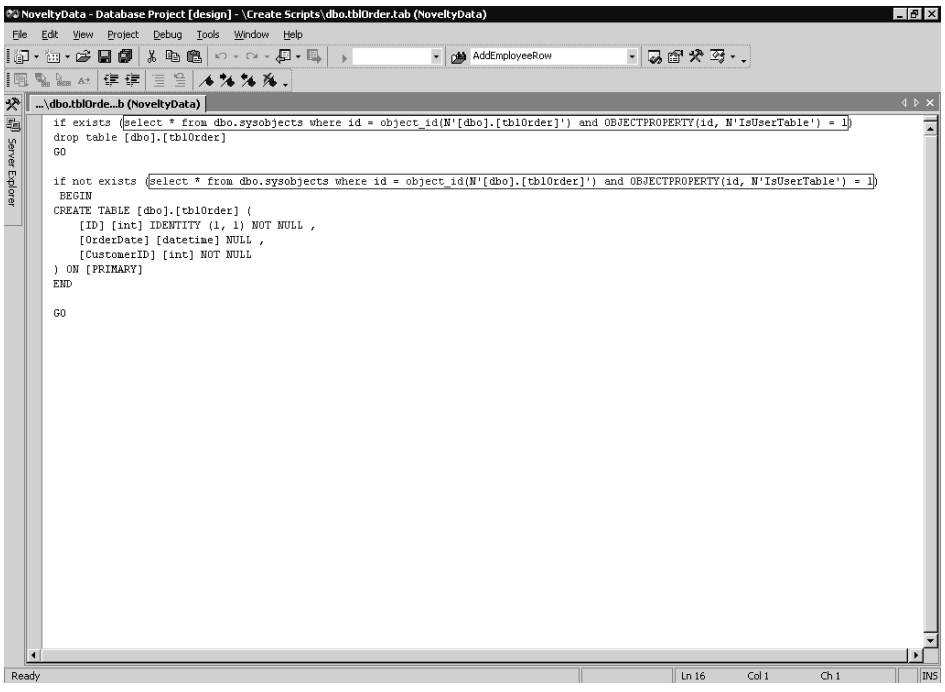

Figure 8.5 *The generated script to create the tblOrder table.*

Change Scripts

Change scripts are used to apply changes to an existing database schema. Although these scripts could be written manually, it is preferable to use a tool to generate them. When we use the Visual Database tools that we used in Chapters 1 and 2, Visual Studio automatically maintains a script of any changes made to the database schema that haven't yet been applied to the database.

Let's say that you want to add a new field, StartDate, to the tblEmployee table to track the start date of each employee. Do the following.

1. Open the Server Explorer, expand the Tables node of the Novelty database, and right-click on tblEmployee.
2. Select the Design Table menu item from the context menu to open tblEmployee in the Table Designer.
3. Add a new column named StartDate, of type datetime, as shown in Figure 8.6.

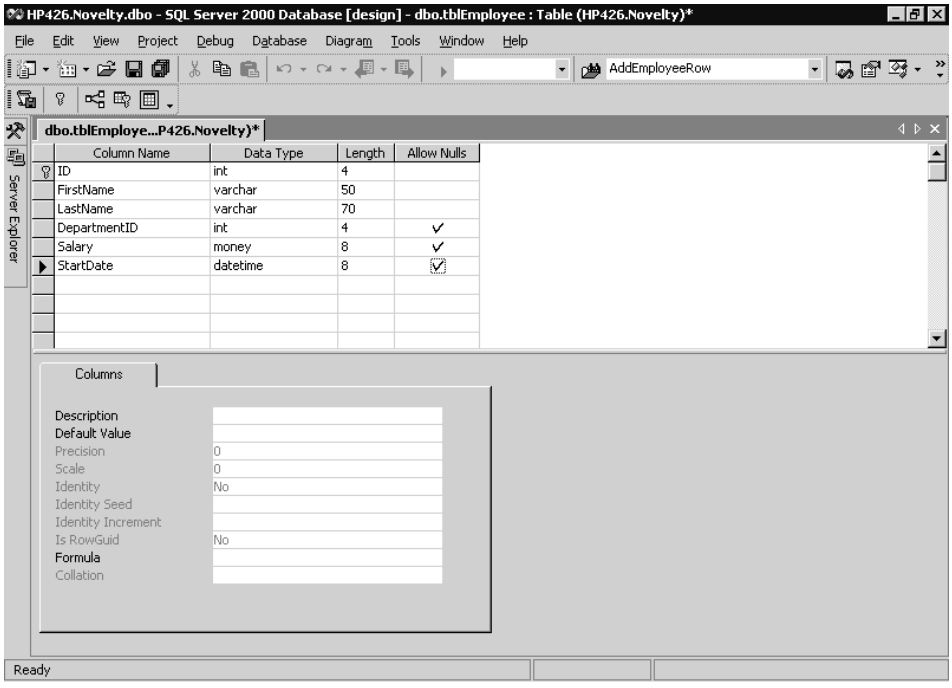

Figure 8.6 *Adding the StartDate column to the tblEmployee table*

Because you will want to apply this change to all the databases already installed and deployed at different sites in the field, you need to create a change script for what you just did.

1. Select the Generate Change Script menu item from the main Diagram menu or click on the Generate Change Script button on the Table toolbar. Doing so displays the Save Change Script dialog box, with a preview of the script, as shown in Figure 8.7.

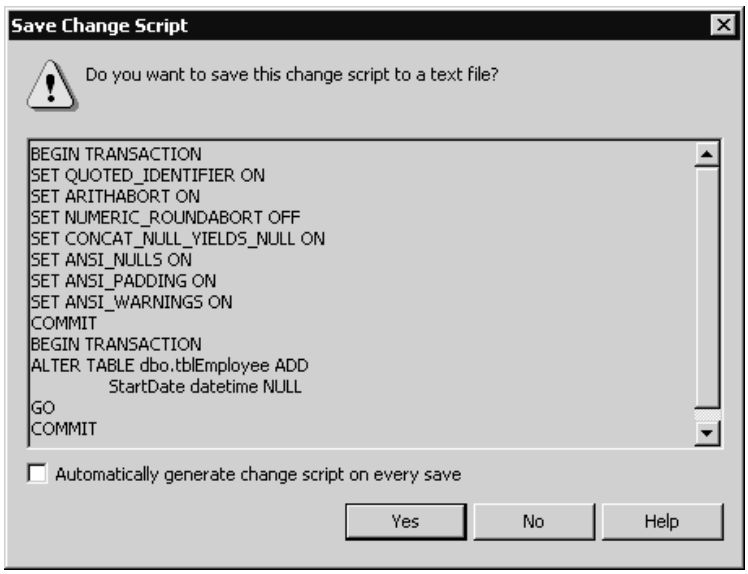

Figure 8.7 *The Save Change Script dialog box, showing the script to add the StartDate column to tblEmployee*

2. Click on the Yes button. The standard Save As dialog is displayed and defaults to the Change Scripts folder of the current database project.
3. Click on the Save button to save the script with the default name of tblEmployee.
4. Close the table designer where you modified tblEmployee. When prompted to save changes to the table, click on the No button. In the next section we show you how to apply the changes by running the SQL script that you just created.
5. Double-click on tblEmployee.sql in the Solution Explorer to view the saved script, as shown in Figure 8.8.

```
/*

   Friday, February 08, 2002 4:17:29 PM

   User: sa

   Server: HP426

   Database: Novelty

   Application:

*/



BEGIN TRANSACTION
SET QUOTED_IDENTIFIER ON
SET ARITHABORT ON
SET NUMERIC_ROUNDABORT OFF
SET CONCAT_NULL_YIELDS_NULL ON
SET ANSI_NULLS ON
SET ANSI_PADDING ON
SET ANSI_WARNINGS ON
COMMIT
BEGIN TRANSACTION
ALTER TABLE dbo.tblEmployee ADD
    StartDate datetime NULL
GO
COMMIT
```

Figure 8.8 *Viewing the tblEmployee.sql script.*

Running the Scripts

You can run a script directly within the Solution Explorer. The easiest way to do so is to drag the script that you want to run and drop it on the database reference that you want to run it against. Alternatively, you can right-click on the script that you want to run. The context menu displayed has both a Run and a Run On menu items. Selecting the Run item executes the script against the default database reference. Selecting the Run On menu item, as shown in Figure 8.9, allows you to specify a database other than the current default. Note that you can choose from existing database references or use this opportunity to add a new one.

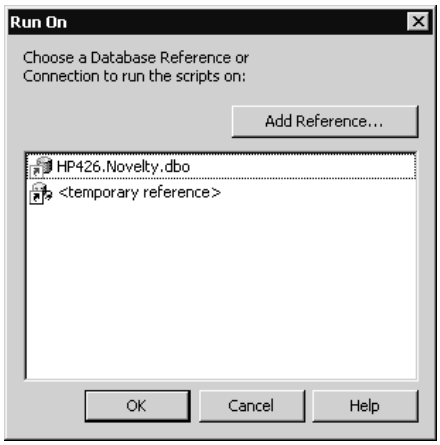

Figure 8.9 *The Run On dialog box*

NOTE

You can also choose to define a temporary database reference to run this script on. Double-clicking on the last item in the list, <temporary reference>, displays the familiar Data Link Properties dialog for you to use to define the connection. However, this reference won't be added to the project or to the Server Explorer.

To apply the changes that you previously designed and saved in the script tblEmployee.sql, do the following.

1. Verify that the tblEmployee table does not contain the StartDate field. In the Server Explorer, expand the node for tblEmployee to list the fields of the table, as shown in Figure 8.10.
2. Expand the Change Scripts folder in the Solution Explorer and select the tblEmployee.sql script.
3. Drag and drop the script onto the reference for the Novelty database in the Solution Explorer.
4. The Execute Scripts or Queries dialog box is shown, giving you a chance to confirm that you want to run the script on the specified database. Click on the Yes button to continue and to run the script.

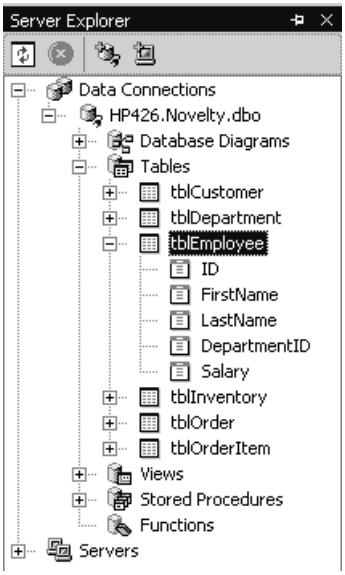

Figure 8.10 *Displaying the fields of tblEmployee*

5. Repeat step 1 to again display the fields of the table (or just click on the Refresh button on Server Explorer toolbar if it remained open). The newly added StartDate field now appears.

Command Files

Now that you have created all these scripts to create and modify the different database objects, wouldn't it be nice if you could organize multiple scripts into a logical group to be run as a single unit? Yes it would be, and VS.NET can create command files to do just that. These command files, which have the .cmd extension, are meant to be used on the Windows 2000 or Windows XP operating systems, which recognize and can execute such files. These files can also load a newly created table with data that we exported from an existing database.

NOTE

The ability to easily and automatically create a script that loads table data in addition to creating database schema and objects is a VS.NET feature not found in the SQL Server Enterprise Manager.

Let's say that we want to create a single command file that will automatically run all the Create Scripts that we need to create a brand new version of our Novelty database on another computer. Although this new system will have its own customers, employees, and orders, the inventory information in tblInventory will be the same. We therefore want to populate the new database's tblInventory table with the data currently in our existing tblInventory table.

Because you will want to have the command file load the inventory data from the existing database to the newly created one, you must first export the data and then continue with the process of creating the command file, as follows.

1. In the Server Explorer, right-click on tblInventory and select the Export Data menu item from the context menu displayed.
2. The Browse for Folder dialog box appears and defaults to the Create Scripts folder of the database project. Click on the OK button to accept this folder.
3. After proceeding through the SQL Server Login dialog, the script dbo.tblInventory.dat is created.
4. Decide which folder in the database project to use to store the new command file. Again use the Create Scripts folder.
5. In the Solution Explorer, right-click on the Create Scripts folder and select the Create Command File menu item from the pop-up menu. Doing so displays the Create Command File dialog box shown in Figure 8.11.

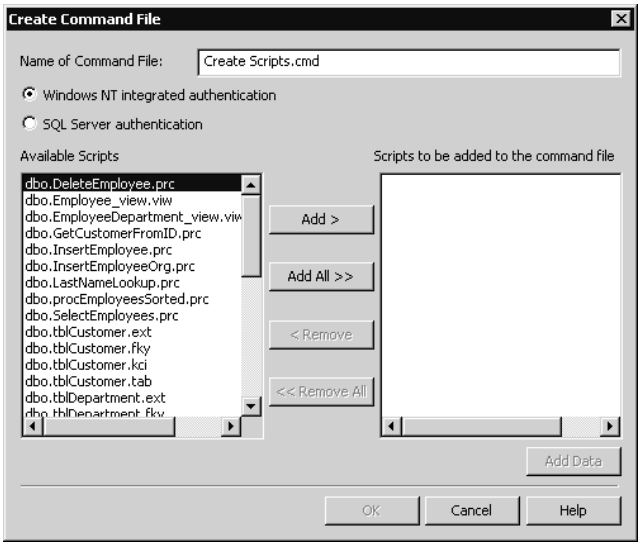

Figure 8.11 *The Create Command File dialog box for the Novelty database.*

6. The Available Scripts list of the Create Command File dialog box contains all the SQL scripts in the selected folder that can be included in the command file. You can add all the scripts, or just individual ones, to the list of Scripts to be added to the command file. Click on the Add All button to add all the Create Scripts to the command file.
7. Because at least one Create Table Script (with the .tab extension) was included in the list of scripts to be added to the command file—and there is at least one exported data file in the folder—the Add Data button is enabled on the Create Command File dialog box.
8. Click on the Add Data button to display the Add Data dialog box shown in Figure 8.12. This dialog lists all the Create Table Scripts that were selected to be added to the command file and allows choosing the corresponding data file for each script.

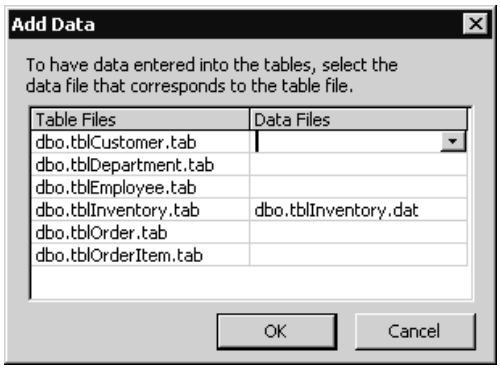

Figure 8.12 *The Add Data dialog box*

9. The dialog recognizes and automatically matches the tblInventory data file with the script that creates the tblInventory table. Click on OK to return to the Create Command File dialog box.
10. Now that the scripts and the exported data files have been specified, click on the OK button to generate the command file. The Create Scripts.cmd command file is added to the Create Scripts folder and its contents are displayed, as shown in Listing 8.1.

LISTING 8.1 The contents of the Create Scripts.cmd command file

```
@echo off
REM: Command File Created by Microsoft Visual Database Tools
REM: Date Generated: 08-Feb-02
REM: Authentication type: Windows NT
REM: Usage: CommandFilename [Server] [Database]
```

```
if '%1' == '' goto usage
if '%2' == '' goto usage

if '%1' == '/?' goto usage
if '%1' == '-?' goto usage
if '%1' == '?' goto usage
if '%1' == '/help' goto usage

osql -S %1 -d %2 -E -b -i "dbo.tblCustomer.tab"
if %ERRORLEVEL% NEQ 0 goto errors
osql -S %1 -d %2 -E -b -i "dbo.tblDepartment.tab"
if %ERRORLEVEL% NEQ 0 goto errors
osql -S %1 -d %2 -E -b -i "dbo.tblEmployee.tab"
if %ERRORLEVEL% NEQ 0 goto errors
osql -S %1 -d %2 -E -b -i "dbo.tblInventory.tab"
if %ERRORLEVEL% NEQ 0 goto errors
bcp "%2.dbo.tblInventory" in "dbo.tblInventory.dat" -S %1 -T -k -n -q
if %ERRORLEVEL% NEQ 0 goto errors
osql -S %1 -d %2 -E -b -i "dbo.tblOrder.tab"
if %ERRORLEVEL% NEQ 0 goto errors
osql -S %1 -d %2 -E -b -i "dbo.tblOrderItem.tab"
if %ERRORLEVEL% NEQ 0 goto errors
osql -S %1 -d %2 -E -b -i "dbo.tblCustomer.kci"
if %ERRORLEVEL% NEQ 0 goto errors
osql -S %1 -d %2 -E -b -i "dbo.tblDepartment.kci"
if %ERRORLEVEL% NEQ 0 goto errors
osql -S %1 -d %2 -E -b -i "dbo.tblEmployee.kci"
if %ERRORLEVEL% NEQ 0 goto errors
osql -S %1 -d %2 -E -b -i "dbo.tblInventory.kci"
if %ERRORLEVEL% NEQ 0 goto errors
osql -S %1 -d %2 -E -b -i "dbo.tblOrder.kci"
if %ERRORLEVEL% NEQ 0 goto errors
osql -S %1 -d %2 -E -b -i "dbo.tblOrderItem.kci"
if %ERRORLEVEL% NEQ 0 goto errors
osql -S %1 -d %2 -E -b -i "dbo.tblCustomer.fky"
if %ERRORLEVEL% NEQ 0 goto errors
osql -S %1 -d %2 -E -b -i "dbo.tblDepartment.fky"
if %ERRORLEVEL% NEQ 0 goto errors
osql -S %1 -d %2 -E -b -i "dbo.tblEmployee.fky"
if %ERRORLEVEL% NEQ 0 goto errors
osql -S %1 -d %2 -E -b -i "dbo.tblInventory.fky"
if %ERRORLEVEL% NEQ 0 goto errors
osql -S %1 -d %2 -E -b -i "dbo.tblOrder.fky"
if %ERRORLEVEL% NEQ 0 goto errors
osql -S %1 -d %2 -E -b -i "dbo.tblOrderItem.fky"
if %ERRORLEVEL% NEQ 0 goto errors
```

```
osql -S %1 -d %2 -E -b -i "dbo.tblCustomer.ext"
if %ERRORLEVEL% NEQ 0 goto errors
osql -S %1 -d %2 -E -b -i "dbo.tblDepartment.ext"
if %ERRORLEVEL% NEQ 0 goto errors
osql -S %1 -d %2 -E -b -i "dbo.tblEmployee.ext"
if %ERRORLEVEL% NEQ 0 goto errors
osql -S %1 -d %2 -E -b -i "dbo.tblInventory.ext"
if %ERRORLEVEL% NEQ 0 goto errors
osql -S %1 -d %2 -E -b -i "dbo.tblOrder.ext"
if %ERRORLEVEL% NEQ 0 goto errors
osql -S %1 -d %2 -E -b -i "dbo.tblOrderItem.ext"
if %ERRORLEVEL% NEQ 0 goto errors
osql -S %1 -d %2 -E -b -i "dbo.Employee_view.viw"
if %ERRORLEVEL% NEQ 0 goto errors
osql -S %1 -d %2 -E -b -i "dbo.EmployeeDepartment_view.viw"
if %ERRORLEVEL% NEQ 0 goto errors
osql -S %1 -d %2 -E -b -i "dbo.DeleteEmployee.prc"
if %ERRORLEVEL% NEQ 0 goto errors
osql -S %1 -d %2 -E -b -i "dbo.GetCustomerFromID.prc"
if %ERRORLEVEL% NEQ 0 goto errors
osql -S %1 -d %2 -E -b -i "dbo.InsertEmployee.prc"
if %ERRORLEVEL% NEQ 0 goto errors
osql -S %1 -d %2 -E -b -i "dbo.InsertEmployeeOrg.prc"
if %ERRORLEVEL% NEQ 0 goto errors
osql -S %1 -d %2 -E -b -i "dbo.LastNameLookup.prc"
if %ERRORLEVEL% NEQ 0 goto errors
osql -S %1 -d %2 -E -b -i "dbo.procEmployeesSorted.prc"
if %ERRORLEVEL% NEQ 0 goto errors
osql -S %1 -d %2 -E -b -i "dbo.SelectEmployees.prc"
if %ERRORLEVEL% NEQ 0 goto errors
osql -S %1 -d %2 -E -b -i "dbo.UpdateEmployee.prc"
if %ERRORLEVEL% NEQ 0 goto errors

goto finish

REM: How to use screen
:usage
echo.
echo Usage: MyScript Server Database
echo Server: the name of the target SQL Server
echo Database: the name of the target database
echo.
echo Example: MyScript.cmd MainServer MainDatabase
echo.
echo.
goto done
```

```
REM: error handler
:errors
echo.
echo WARNING! Error(s) were detected!
echo ————————————————————
echo Please evaluate the situation and, if needed,
echo restart this command file. You may need to
echo supply command parameters when executing
echo this command file.
echo.
pause
goto done

REM: finished execution
:finish
echo.
echo Script execution is complete!
:done
@echo on
```

NOTE

The command file makes use of the osql and bcp command line utilities that are part of the SQL Server installation. The osql utility allows you to execute SQL statements, system procedures, and script files. The bcp is a bulk copy program that copies data to and from a data file and an instance of SQL Server.

You can run this command file from within the Solution Explorer by right-clicking on it and then selecting the Run menu item. You can also invoke it externally, independent of Visual Studio, so long as all the scripts exist together with the command file.

TIP

Remember that running this command file against a database will delete all the data that currently exists in that database!

Queries

Similar to what we mentioned regarding the Create and Change scripts, you can also create SQL queries manually within the Visual Studio environment. However, except for the most trivial queries, using the Query Designer to design the queries graphically is much more efficient and less error-prone.

We can demonstrate use of the Query Designer by creating a parameter update query that updates the wholesale prices of all of the products in our inventory by a specified percentage.

1. Open the Solution Explorer and right-click on any folder node other than Database References. From the context menu displayed, select the Add Query menu item. Doing so displays the Add New Item dialog box, shown previously in Figure 8.2.
2. Select the Database Query template, set the name to UpdateWholesale.dtq, and click on the Open button. The Query Designer is now displayed, as shown in Figure 8.13.

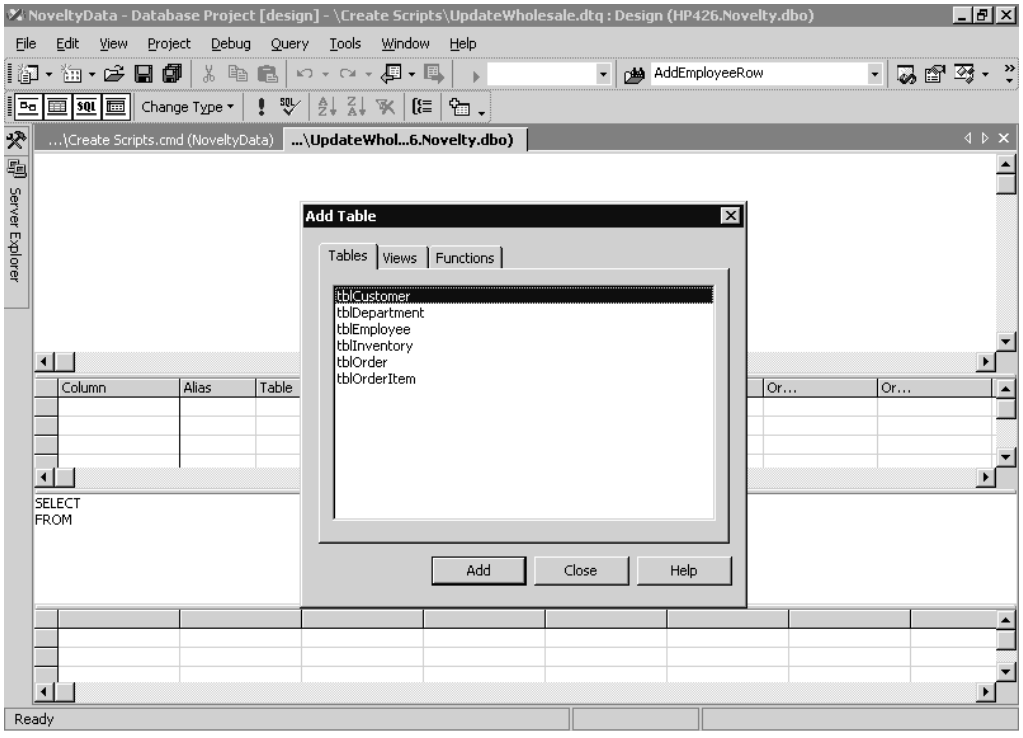

Figure 8.13 *The Query Designer opened to create a new database query*

3. From the Add Table dialog, add the tblInventory table and then click on the Close button to dismiss the dialog.
4. We need to change the query type from a Select to an Update query. We do so by selecting the Change Type menu item from the main Query menu, or by clicking on the Change Type button on the Query toolbar, and then selecting the Update menu item.
5. In the Diagram pane of the Query Designer, click on the checkbox for the WholesalePrice field, as that is the field we're going to update.
6. In the Grid pane, enter the following formula in the New Value column in the row for the WholesalePrice field that we just added:

```
WholesalePrice * (1 + ? / 100)
```

7. This formula accepts a parameter value, which is the percentage that the wholesale price should be increased by. When the query is executed, the question mark in the formula will be replaced by the parameter value that is provided. The Query Designer should look like that shown in Figure 8.14.

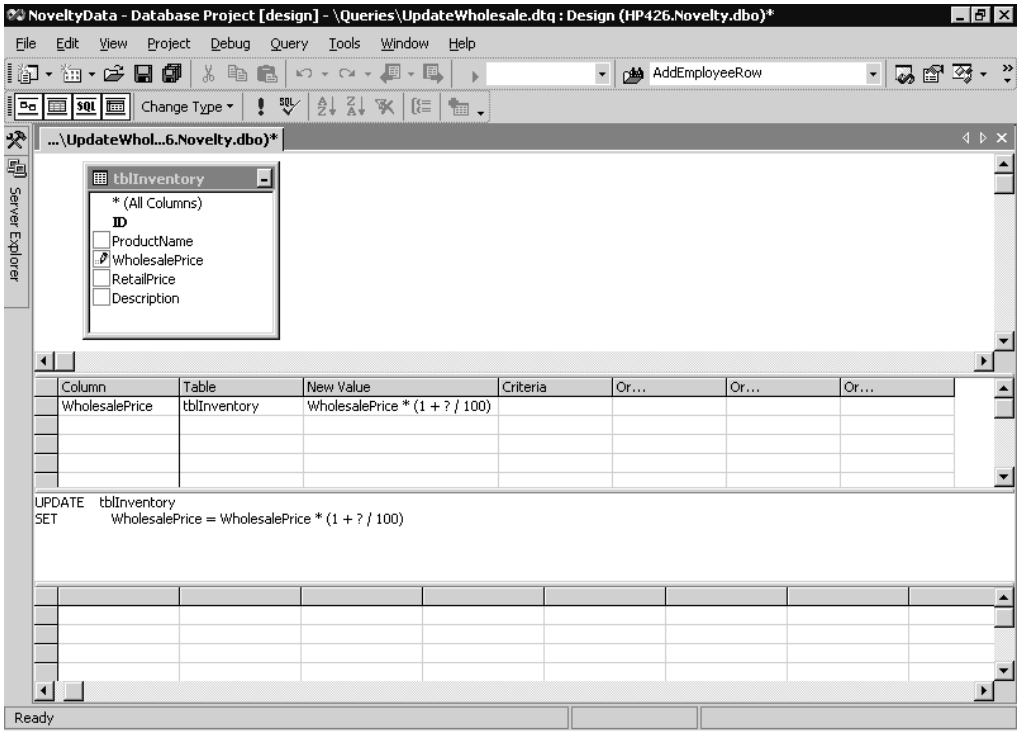

Figure 8.14 *The query to update the WholesalePrice field in the tblInventory table*

8. Although we could now run and test our query within the Query Designer, we will (soon) do it from the Solution Explorer.
9. Close the Query Designer and click on the Yes button when prompted to save the changes to the UpdateWholesale.dtq query.
10. Double-click on tblInventory in the Server Explorer to display all the current data in that table. These are the values before we run the query script to update the wholesale price.

TIP

You may want to take a snapshot of the data so that you can easily verify the new prices after running the update query. You can do so by selecting all the rows that are being displayed and then copying and pasting them in a Notepad file (or any other tool of your liking).

11. Similar to what we showed previously with a script, we can run a query by dragging and dropping it onto a database reference. We can also run it on the default reference by right-clicking on the query that we want to run and selecting the Open menu item.

NOTE

The context menu displayed when you right-click on a query also contains Design and Design On menu items that open the Query Designer against the default or a specified database reference.

12. Drag and drop the UpdateWholesale query onto the reference for the Novelty database in the Solution Explorer.
13. The Execute Scripts or Queries dialog box is shown, giving us a chance to confirm that we want to run the script on the specified database. Click on the Yes button to continue and to run the query.
14. The Define Query Parameters dialog box is shown because our query contains a parameter. Enter a value—say, 10—as the percentage increase for the wholesale prices. Click on the OK button to continue.
15. A message box is displayed, stating the number of rows affected by the query. Click on OK to dismiss it.
16. Repeat step 9 to again display the data in the tblInventory table. The table should contain the modified wholesale prices, as shown in Figure 8.15.

	ID	ProductName	WholesalePrice	RetailPrice	Description
▶	1	Rubber Chicken	0.308	2.99	The quintessential rubber chicken.
	2	Joy Buzzer	2.244	9.99	They will get a real shock out of this.
	3	Seltzer Bottle	3.432	15.24	Seltzer sold separately.
	4	Ant Farm	3.795	14.99	Watch ants where they live and bree
	5	Wind-Up Robot	6.534	29.99	Giant robot: attack toybox!
	6	Rubber Eyeballs	0.132	0.99	Peek-a-boo!
	7	Doggy Mess	0.462	1.99	Yechhh!
	8	Mini-Camera	2.299	9.99	
	9	Glow Worms	0.044	1.99	
	10	Insect Pops	0.198	0.99	
	11	Alien Alarm Clock	10.549	45.99	
	12	Cinnamon Toothpic	0.44	1.99	
*					

Figure 8.15 *The data in the tblInventory table, after increasing the WholesalePrice field by 10 percent*

Summary

In this chapter we focused on the Visual Studio.NET database project type. This project type doesn't include any Visual Basic code; rather, it is meant to be used to create, test, run, and manage SQL database scripts, queries, and command files. These scripts and commands allow you to create new database schemas, make changes to existing schemas, and query and update existing database data. These are important, time-saving tools that should be used as much as possible during both the development and deployment phases of a project.

In general, you probably won't bother getting involved with database project scripts and queries unless you're developing, maintaining, and enhancing a real-world database application. Furthermore, as an application programmer, you may be used to leaving the tasks discussed in this chapter to database analysts (DBAs). However, more and more application programmers are assuming many of the tasks traditionally performed by DBAs. Even if that isn't the case in your situation, you may still need to perform many of these tasks for your own private development environment.

Questions and Answers

Q: I see that many of the same or very similar tools exist in both Visual Studio and the SQL Server Enterprise Manager. Which ones should I use?

A: The short answer: whichever one(s) you prefer. For many operations, the tools are the same in either toolset, so you can go with whichever you prefer. However, some operations are easier, or can only be done, with one tool or the other. In all likelihood, if you are a DBA,

you will do most of your work from within the SQL Server Enterprise Manager. If you are a programmer, you will most likely want to do as much as possible within the Visual Studio environment. However, if your database is not SQL Server, you will need other tools in order to design or modify your database objects. The Visual Studio tools allow you to browse and query such databases but not to modify them.