CHAPTER 5

Creating a Data Warehouse

Plans are only good intentions unless they immediately degenerate into hard work.

—Peter F. Drucker

When all the planning is done, it is time to create the data warehouse. SQL Server makes this task quite simple and effortless.

In this chapter, we discuss how to use the SQL Server 2012 Management Studio (SSMS) application to create a data warehouse. We look at several techniques to accomplish this, including SQL code, a table designer, and a diagramming tool that lets you visually create all of the data warehouse objects. When you have completed this chapter, you will have a working data warehouse that is ready to be filled with data.

SQL Server Management Studio

SQL Server Management Studio is one of the most important applications of Microsoft's SQL Server 2012. It allows you to create and manage your databases as well as work with certain aspects of Microsoft's other BI servers (SSIS, SSAS, and SSRS).

■ **Note** At the time of this writing, SQL Server Data Tools (SSDT) has been newly released for SQL 2012. SSDT adds many, but not all, of the features found in SQL Server Management Studio (SSMS) to the Visual Studio environment. It does not replace SSMS, and it does not replace the way databases are created. It does provide added convenience to Visual Studio, but because this book is about BI development rather than the new features of SQL 2012, we use SQL Server Management Studio for our examples. For more information about SSDT, visit the author's website at `http://NorthwestTech.org/ProBISolutions/SSDTDemos`.

To launch SQL Server Management Studio, navigate to Start ➤ All Programs ➤ Microsoft SQL Server 2012 menu item and click the title to expand the selection, as shown in Figure 5-1. Then, right-click the SSMS menu item and select Run as Administrator to open the SSMS application. It is important to run SQL Server BI applications as a Windows administrator to avoid permission issues.

Figure 5-1. *Opening SQL Server Management Studio with adminstrator rights*

Connecting to Servers

When SQL Server Management Studio first opens, you will be asked to connect to a SQL Server installation using the Connect to Server dialog window. You can connect to several different types of servers including SQL Server's database server, the Integration Services Server, the Analysis Cube Server and the Reporting Services Server. We work with all of these servers and make the connections to them in future chapters, but for now, we are focusing on the SQL Server database engine.

■ **Note** When you first open SQL Server Management Studio, the Object Explorer window will not be populated with servers as shown in Figure 5-2 until after these connections have been made.

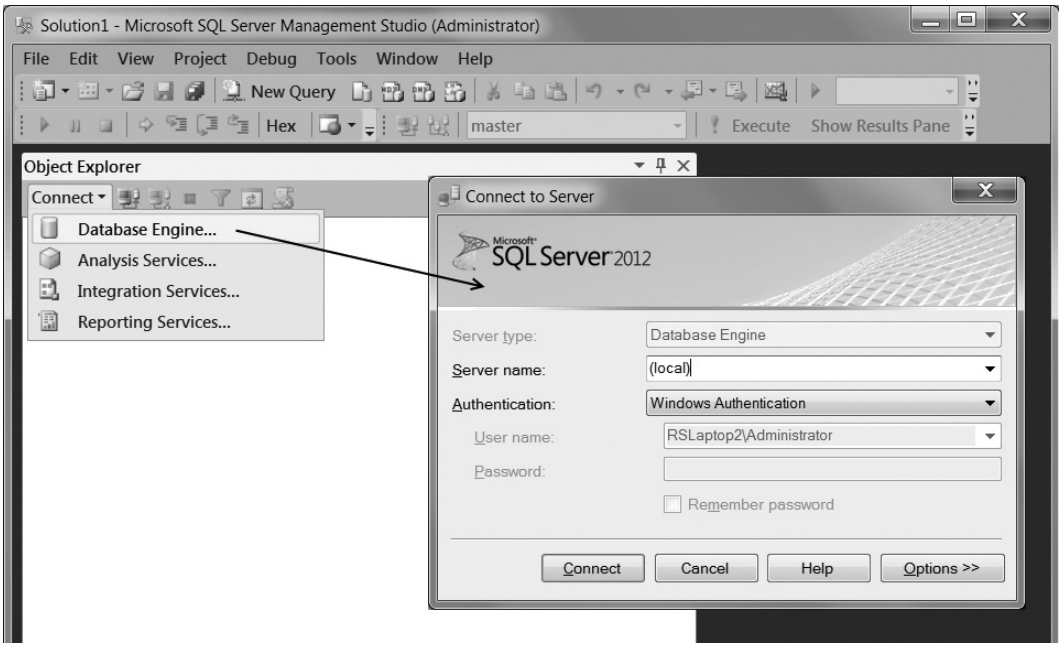

Figure 5-2. *Examples of the types of connections that can be made to the four BI servers*

To connect to a server, on the left of your SSMS window within the Object Explorer, click the Connect dropdown menu item and choose a server type. In Figure 5-2 we are selecting Database Engine. When the Connect to Server dialog window opens, enter the name of the server you want to connect to, as shown in Figure 5-2.

You can connect to more than one server at a time. In Figure 5-3, you can see that we have connected to all three of the BI servers from SQL Server Management Studio. When we did, Object Explorer indicated the type of server and the version number. Any server with a version number starting with 11 indicates SQL Server 2012.

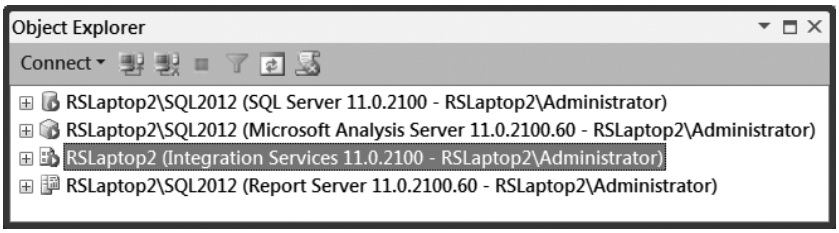

Figure 5-3. *Connecting to multiple servers at one time*

■ **Note** The screen shots from Randal's computer are using a named instance called RSLaptop2\SQL2012. The exception of this is the Integration Services (SSIS) connection. SSIS does not allow for named instances. We talk more about named instances in just a moment.

Server Aliases

When connecting to one of the four BI servers on your machine, depending upon your configurations, you can use several different methods to connect; all of which mean the same thing. For instance, you can use localhost, (local), 127.0.0.1 or your computer name. You can even type a single period and SQL Server Management Studio understands to connect to your local server installation. For demonstration purposes, in Figure 5-4, we illustrate these four methods of connecting; each time it is to the SQL Server database engine.

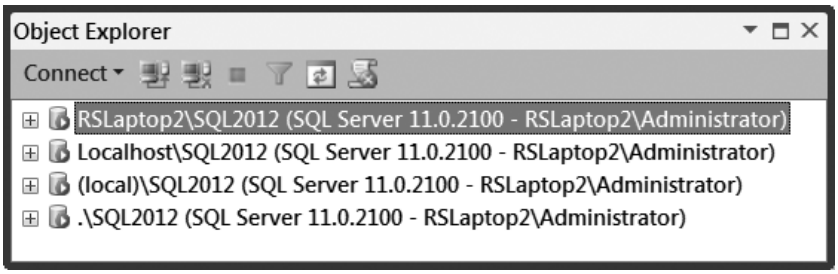

Figure 5-4. *Connecting to the database engine with various aliases*

It does not matter how you choose to connect, but it is important to be able to recognize each method. In this book, we use either *(local)* or *(local)\SQL2012* for most of our examples.

Use whichever works! Unfortunately, not all options work on all computers, so you will have to figure out which one is able to connect on your machine. For example; localhost allows Randal to connect to SSIS, but the alias (local) does not, even though they are supposedly equivalent. And Caryn connects in a different manner altogether, because her only installation of SQL Server is a named instance. We discuss how to handle named instances below.

To help you understand why some connection method's work and others do not, here is an overview of the differences:

- *Localhost*: Allows you to connect using the network protocol TCP/IP. As you may know, this is the protocol used throughout networks today. Localhost and the IP address 127.0.0.1 are equivalent to each other. They are both associated with TCP/IP, and each acts as an alias for your computer name. Using the name localhost to connect to your SQL Server works most of the time, but some configurations prevent this from working on all computers. Although there are ways to fix these issues, the easiest thing to do is try one of the other options.
- *(Local)*: Typing in the word **(local)** with parentheses will almost always connect when localhost does not. This name gives you access using an older protocol called NetBIOS. Microsoft originally used NetBIOS for networking and required NetBIOS on top of TCP/IP for many years. This requirement is no longer mandatory; however, legacy items remain, such as your ability to use (local) as an alias for your computer. Please note that (local) is the only computer alias that uses parentheses. This can confuse new developers into putting parentheses around aliases like *localhost*, leaving them unable to connect because *localhost*, with parentheses, has no significant meaning in NetBIOS or TCP/IP.
- *A period "."*: If, for some reason, you still cannot connect, another option is to type a single period as a server alias. This period symbol often works when no other selections will. It originates from an even older Microsoft protocol called *Named Pipes*. Named Pipes allows applications to talk to each other on a Windows machine and has been used extensively over the years.
- *Your computer name*: You can always just type in your computer name. Keep in mind that you can also connect to BI servers on a different computer by typing in its name. For example, let's say your company has a development server named DevServer; you could connect to it from your desk or laptop computer using its name, DevServer.
- *Computer Name or Alias\A_Named_Instance_on_Your_Server*: This allows you to connect to additional named instances installed on a single computer. An example of this would be (local)\SQLExpress or DevServer\SQLTestInstall. Keep in mind that you may only have named SQL Server instances installed on a computer. If this is the case, attempting to connect using the computer name is not enough. You must use the computer name or an alias such as (local) or localhost followed by the backslash "\" followed by your instance name—with no spaces in between.

WHAT IS UP WITH NAMED INSTANCES?

Named instances may be confusing, but they are very convenient. Here is a brief history of how and when they were added to SQL BI servers.

When SQL Server was first introduced by Microsoft, it could be installed only once per computer. In SQL 2000, this changed, and you could install the SQL database engine multiple times on the same machine.

To do this, each additional installation needed its own name. That is why it is called a *named instance*. Before SQL 2005, the first installation always installed without an additional identifier and was by definition the *default instance.* Only subsequent installations were given a unique name.

From 2005 on, the rules changed again, and you can now install SQL, SSAS, and SSRS with an additional identifier even on the first installation, without requiring a default instance. On your personal computer, if a default instance is not installed, you must qualify your SQL, SSAS, and SSRS servers with the full name; *Computer Name\named instance name*.

Another method of connecting is to use the Server Name dropdown menu. Select the < browse for more. . . > option, and then select your local machine or a network to connect to (Figure 5-5).

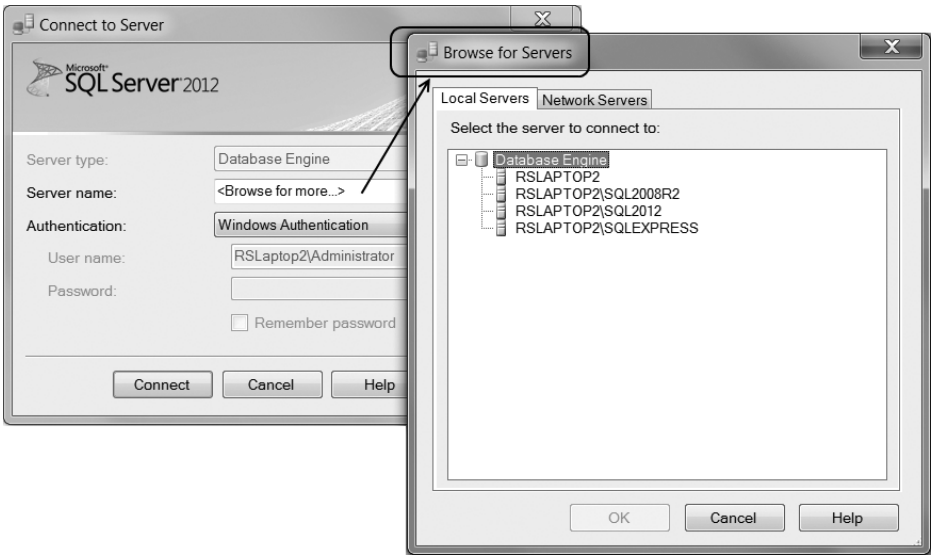

Figure 5-5. *Browsing for SQL Servers*

In Figure 5-5, you can see the local servers list on Randal's laptop. Selecting one of these creates a connection to the server as long as that database engine is running. Additionally, you can connect to remote SQL Servers as well, as long as it is running and remote connections to the other computer are allowed.

Configuration Manager

If you want to allow a remote connection to your server, you can enable this using the SQL Server Configuration Manager. You can also use this tool to make sure that an instance of SQL server is running on a given computer or to manage SQL Server, SSAS, SSIS, and SSRS start-up and service account settings.

Figure 5-6 lists SQL Server services and indicates that the named instance for (local)\SQLExpress is not currently running, but three other SQL servers are currently running. The SQLExpress named instance is set to start up manually, whereas the other three servers start automatically when the computer boots up. As it is now,

you could not connect to the SQLExpress named instance from Management Studio with first starting the server by right-clicking its name and selecting Start from the context menu.

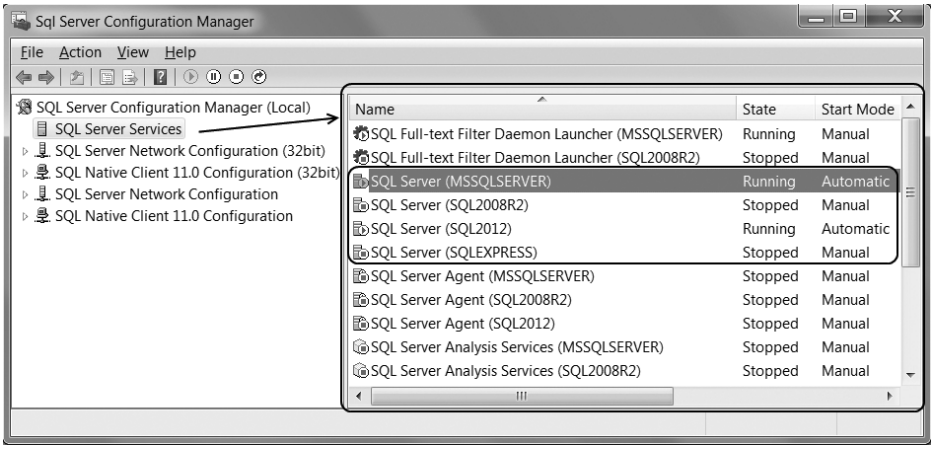

Figure 5-6. *Checking the SQL Servers with the Configuration Manager*

■ **Tip** In SQL Server Configuration Manager any server with the name (MSSQLSERVER) is the default instance and is accessed solely by the computer name or alias. No instance name is required. This is true for SQL Server, SSAS, and SSRS, but not for SSIS, which does not allow named instances.

To get to the SQL Server Configuration Manager, you need to navigate to the Start menu, select All Programs ➤ Microsoft SQL Server 2012 ➤ Configuration Tools, and right-click the SQL Server Configuration Manager menu option. From there, select the Run as Administrator option to launch the Configuration Manager.

When SQL Server Configuration Manager opens, you see a navigation tree on the left side of the screen. Selecting SQL Server Services shows you a window similar to the one in Figure 5-6. In this window, you can see which services are currently running and either stop or restart them by right-clicking the server name and selecting the Start or Restart option in the context menu. You can also access a number of server and start-up settings using the Properties option.

Clicking the SQL Server Network Configuration (32bit) node displays a window similar to Figure 5-7. Here you enable or disable the network protocols used for remote SQL connections. You will not be able to access the SQL BI servers remotely without TCP/IP enabled, and by default it is disabled.

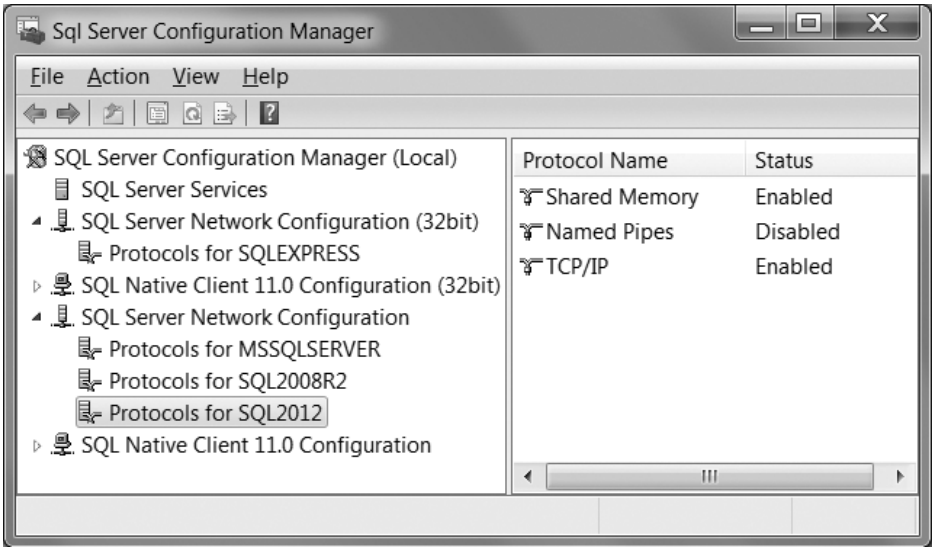

Figure 5-7. *Enabling TCP/IP*

To configure a network protocol, first expand the SQL Server Network Configuration node, and then click the "Protocols for [SQL Server instance name]" node for whichever instance you wish to configure.

If a protocol is disabled, you can right-click the protocol and then click Enable from the context menu to enable it. The status will change when the protocol is enabled, but you must restart the SQL Server service before applications can connect to the server. Additionally, Microsoft recommends restarting the SQL Server Browser service as well. Both of these services can be restarted under the SQL Server Services node (Figure 5-6).

Finally, you must use SQL Server Management Studio to allow remote access by checking the "Allow remote connections to this server" checkbox (Figure 5-8). This checkbox is found on the SQL Server Properties window under the Connection page.

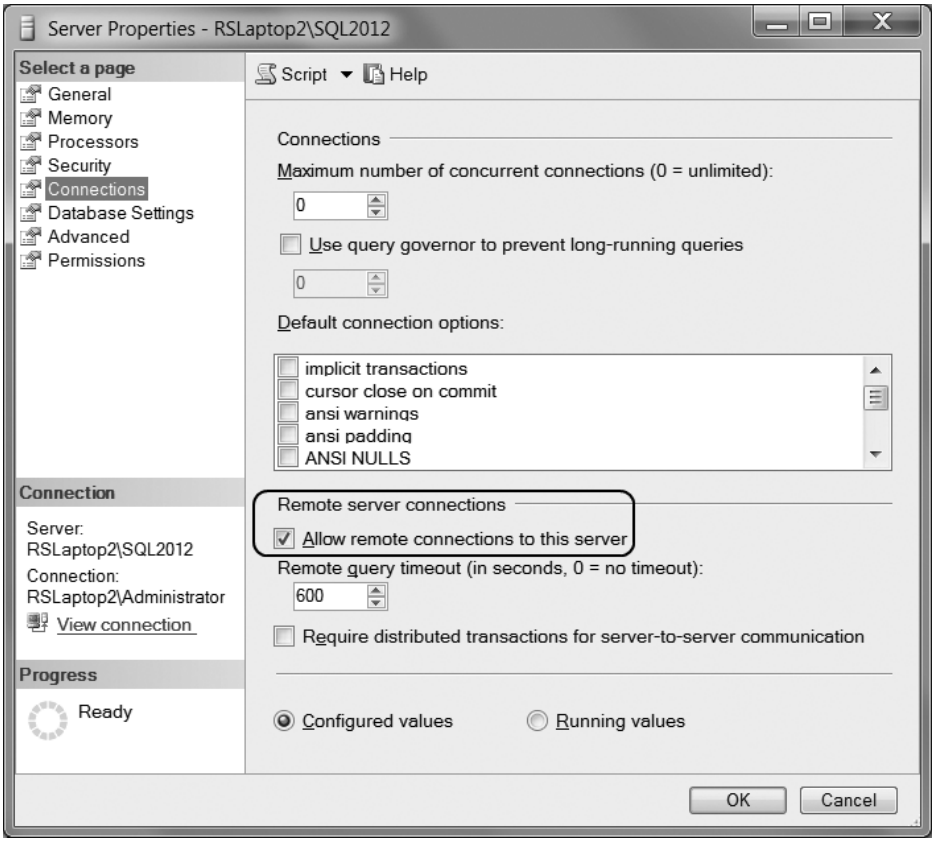

Figure 5-8. *Enabling remote connections for a SQL Server*

■ **Note** Our purpose is to allow you to become familiar with the network protocol settings, but there is no need to make changes at this time. No remote connections are necessary for the exercises in this book.

Management Studio Windows

SQL Server 2012 introduces a new SQL Server Management Studio user interface. The design may look different, but functionally it is very similar to previous versions, and it includes multiple windows. These windows can be repositioned and even moved outside Management Studio. This feature is great for use with multiple monitors. However, the majority of the time you will use only Object Explorer and a query window.

Object Explorer

Object Explorer is the first of the two main windows utilized in SQL Server Management Studio. In the default configuration, the Object Explorer window is displayed on the left side of the screen (Figure 5-9). Object Explorer allows you to view the various components of the database engine in a treeview format. By clicking the + next to each folder icon, you are able to expand the folder to access subcomponents.

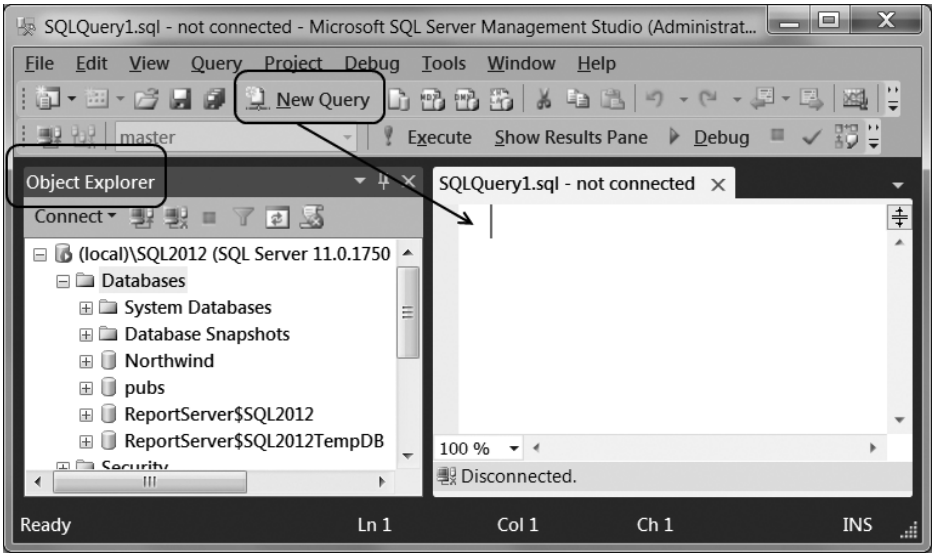

Figure 5-9. *The SQL Server Management Studio UI*

The Query Window

The second main window used on a regular basis is the query window. As indicated in Figure 5-9, there is a button on the toolbar called New Query which opens a new query window for you. This window is actually a set of windows, and just like modern web browsers, you can have many query windows open and each is accessed by its tab. Each tabbed query window has its own unique connection to a database engine.

By default each query window opens in the center of SQL Server Management Studio. Once opened, one or more SQL commands can be typed into a query window and then executed separately or collectively.

A query window must be connected in order to execute SQL code. In Figure 5-9, the disconnected status at the bottom of the screen is a clear indication that the query window is not yet connected. Disconnected queries are easily remedied by right-clicking anywhere on a query window, choosing the Connection option from the context menu, and selecting Connect.

It is important to note that the Object Explorer window represents a separate connection. You may be connected to a database engine in the Object Explorer, but you are not necessarily connected to a database engine in a query window. This can often be a source of confusion because developers expect one application to have only a single connection to a server at a time. Once you are able to get past the confusion, it is a very convenient feature nonetheless, since it allows you to work with multiple servers at once from a single application.

Changing the Query Window Focus

To focus on a particular database in Object Explorer, you need to expand the database folder and select a database from the expanded treeview. To focus on a particular database in the query window, use the *Available Databases* dropdown box (circled in Figure 5-10). This may be hard to spot because the name of the dropdown box is not shown, it can be repositioned on the toolbar, and the selected database may be different than expected. The Master database is typically displayed, but you can of course change this. In Figure 5-10, we have selected DWWeatherTracker instead, which causes the query window to be focused on that database.

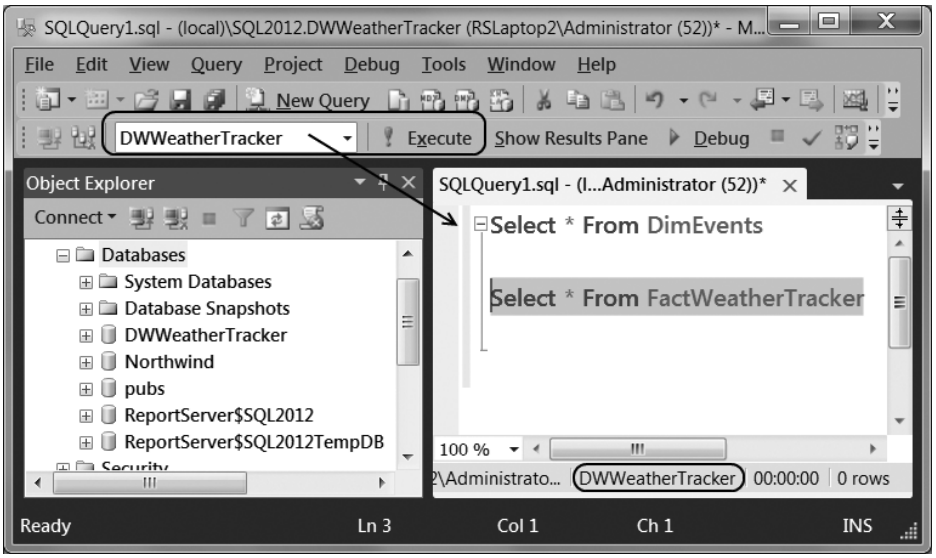

Figure 5-10. *Using the query window*

Although we are not showing it in Figure 5-10, you can also change the focus to a database by executing the command: `USE DWWeatherTracker.` Looking at the status panel at the bottom of the query window tells you which database is currently in focus.

Executing a Query

After you have added code to the query window, you are then able to execute that code by clicking the "! Execute" button, as shown in Figure 5-10. You can also execute a query using the keystrokes Alt + X or Ctrl + E.

■ **Important** We do *not* recommended using the Debug button (with the green arrow) to execute your code. A debugging session can start other processes and complicate what you are working on. If you are a .NET programmer, you know that this is contrary to how you normally run code in Visual Studio in C# or VB.NET applications. Nevertheless, it is important to remember that while working in SQL Server Management Studio, execute your code with the "! Execute" button rather than debugging it.

In many database applications, a query window may hold only a single statement or batch of statements at a time, but this is not true in SQL Server Management Studio. Instead, you can type in hundreds of lines of SQL code and execute them independently, as a batch, or as several batches by selecting whichever statements you want to run.

If no statements are highlighted, all the statements in the query window run sequentially. In Figure 5-10 you see two SQL statements have been typed into the query window, but only one statement has been selected. In this example, only the highlighted statement will be executed when "! Execute" is clicked.

■ **Tip** The term *SQL batch* is used to describe one or more SQL statements that are submitted to the database engine as a unit. Some statements, such as the `CREATE PROCEDURE` statement, must be the first statement in a batch, but most statements can be submitted in any order. If you want to create multiple procedures in one query window, you can use the batch separator keyword `GO` to divide the SQL statements into multiple batches. You may also use the `GO` keyword between any of the individual statements, but this is more of a stylistic choice than a programmatic one and is not necessary for your statements to execute correctly. Examples of this are shown throughout the code samples of this book.

SQL code can be submitted to the database engine from other applications as well as SQL Server Management Studio. Simple examples of this are executing code from SSIS and SSRS. We investigate both of these in later chapters of this book.

Creating Data Warehouse Database

With SQL Server Management Studio, you can easily create a database by right-clicking the database icon in Object Explorer, as shown in Figure 5-11. This launches the New Database dialog window. When the dialog window opens, there is a selection of pages on the left of the dialog window, and on the right are associated text boxes and grids. To create a database, select the general page and provide a name for the database such as DWPubSales; then indicate the owner of the database as the system administrator account, SA.

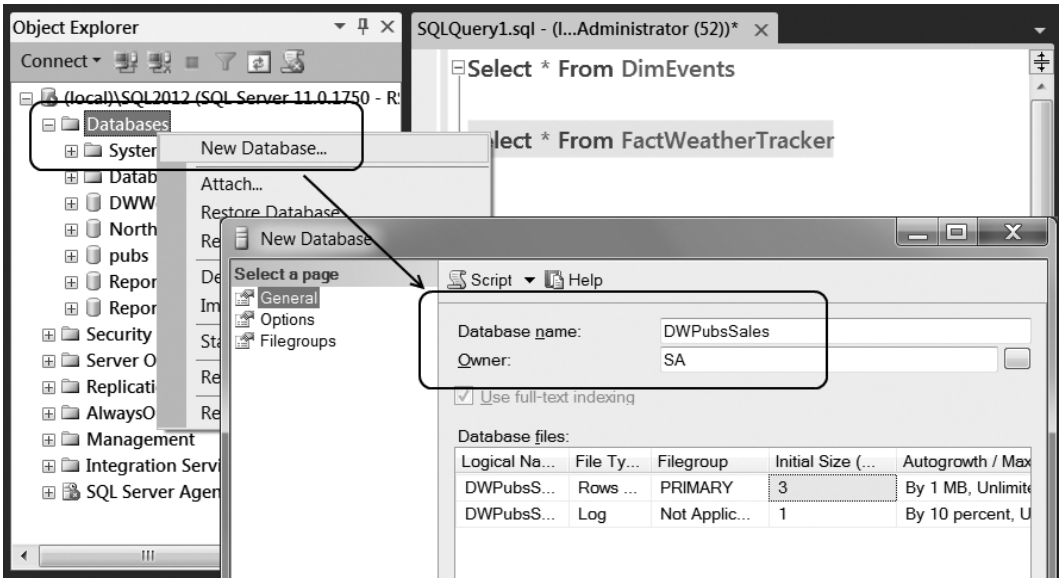

Figure 5-11. *Creating the data warehouse database*

■ **Note** Next, we discuss a number of advanced options in the database creation process, but you do not need to use these options for the exercises of this book.

Setting the Database Owner

Most of the time, the database owner setting has little impact on working with the database. An example of when it is necessary, however, is when using SQL Server's diagram features. If there is no owner assigned to the database or the one that was assigned is no longer valid, the diagramming tools will not work.

The database owner can map to either a Windows login account or a SQL Server login account. Both are similar, but a SQL Server login's name and password are passed onto the network as plain text. And although this network traffic can be encrypted, by default it is not. Because of this, many SQL administrators have stopped using SQL logins for most occasions and just use Windows logins instead. Thus, the standard choice for a database owner is an existing Windows login.

There are times, however, where using a SQL login for the database owner makes sense. For example, if you created a database using your personal login account as the owner of the database and then backed it up to send to a co-worker's computer, the database owner may not be valid on your co-worker's computer. When your co-worker restores the database on his or her own machine, if your personal login account does not exist on his or her machine, the database ownership is broken.

At first, this would probably go unnoticed, but if your co-worker tried to create a database diagram, which requires the use of valid owner, your co-worker will get the error message shown in Figure 5-12.

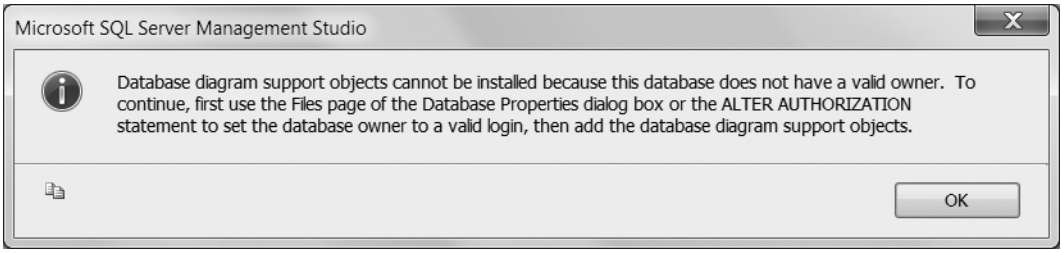

Figure 5-12. *This database is missing a valid owner*

Perhaps the easiest way to resolve this issue is to assign a valid owner using a SQL login that you know will be on all SQL Server installations. (The SA login was used in Figure 5-7.) The SA login is built into Microsoft SQL Server as a system administrator and works very nicely for this occasion. Although it is disabled by default, it still exists on each and every installation of SQL Server and by its mere existence can be used as a valid owner. When you create your database using the new database dialog window, the SA account is a simple and effective choice that can safely be used for nonproduction computers. On a production machine, you normally use a Windows domain account.

■ **Note** The authors, and the legal department, are really hoping that you are not using a production server to perform the exercises in this book, so the SA login should work just fine throughout the text! Please let the lawyers sleep at night and use only nonproduction servers. They get grouchy when they don't get their sleep!

Setting the Database Size

During this creation process, many decisions must be made; such as where the files for the database should go or what their initial size will be. In the examples from this book, you can safely leave these at the default settings because the samples are very small. In real life, however, you will probably want to put your data warehouse on its own separate hard drive and try to match the initial size as close as possible to its expected size.

Every database you create on SQL Server will have at least two files with which to be concerned. They are the data file and the log file. The data file holds all the data from your tables, and the log file records any changes to the data file. All changes are recorded to the log file before they are changed in the data file. A synchronization process, known as a *checkpoint,* occurs on a regular basis to record all the changes from the log file into the data file. This is performed automatically.

When estimating the size of the data warehouse, the two files should be adjusted according to their function. The data file needs to be large enough to hold all your data, but the log file can be quite small. In an OLTP database, the transaction log is expected to be receiving continual transaction entries, but the log file has much less work to do in an OLAP data warehouse. The log file records data entries while importing data from the OLTP source to the OLAP destination, but reporting statements are not logged. Microsoft recommends an OLTP database log file be set to 25% of the size of its associated data file. In an OLAP data warehouse, you can usually get by with a lot less: approximately 10% of the data file. So, if you made your data file 100 MB, the log file will then be 10 MB.

Listing 5-1 shows the code for creating a typical OLAP database. We have placed the files on a different hard drive to maximize performance and started with the D drive, so the Windows operating system is on a separate hard drive as well.

Listing 5-1. Creating a New Database

```
/* (Note the following code is expected to error out and is only a demo) */
USE [master]GO
CREATE DATABASE [DWWeatherTracker] ON PRIMARY
( NAME=N'DWWeatherTracker'
, FILENAME=N'D:\_BISolutions\DWWeatherTracker.mdf' -- On the D:\ hard drive
, SIZE=10MB
, MAXSIZE=1GB
, FILEGROWTH=10MB )
LOG ON
( NAME=N'DWWeatherTracker_log'
, FILENAME=N'F:\_BISolutions\DWWeatherTracker_log.LDF' -- On the F:\ hard drive
, SIZE=1MB
, MAXSIZE=1GB
, FILEGROWTH=10MB)
GO
EXEC [DWWeatherTracker].dbo.sp_changedbowner @loginame=N'SA', @map=false
GO
ALTER DATABASE [DWWeatherTracker] SET RECOVERY BULK_LOGGED
GO
```

Estimating the exact size of the data warehouse files can be tricky. You do not have to be exact, however, just close. SQL Server databases have the capacity to grow in size automatically as more data is added, but it is considered best practice not to rely on this feature. An easy way to estimate the data warehouse size is by looking at the source databases or source files to identify what is going to be imported and how much space it currently takes up.

If your source is a SQL Server OLTP database, the process is greatly simplified by a number of reports included in SQL Server Management Studio. To access these reports, right-click an object in the treeview of Object Explorer and look for the Reports menu option. From there you see a list of standard reports, a selection for custom reports and a list of currently used reports. Upon selecting a particular report, such as the one shown in Figure 5-13, a report is generated for you. You can estimate the size of your data warehouse tables by

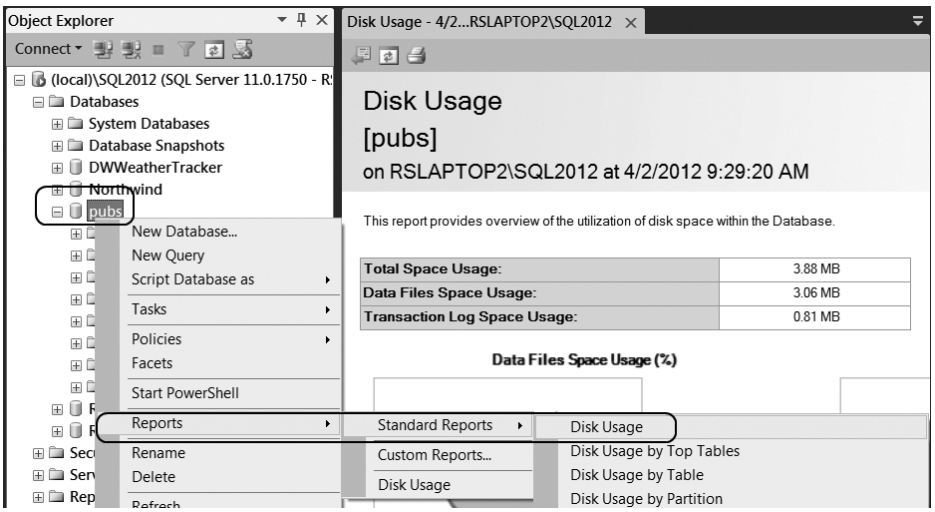

Figure 5-13. *Estimating the size of the data warehouse database*

determining which OLTP tables you intend to use as the source for your dimension and fact tables and then noting their current size.

■ **Note** SQL Server Management Studio reports are rendered using a miniature version of the Report Server engine and do not require that SSRS be installed or configured on your machine. For information on setting up custom reports, search the Web for "Customer Reports Management Studio."

This is a simple effective way to estimate the size of your data warehouse, but it will not work for all occasions. For instance, you may have to pull data from sources other than SQL Server, such as a set of flat files or an Oracle server. In those cases, you have to estimate the database size the old-fashioned way: by estimating the size of each row in a table and multiplying it by the number of rows in that table. You can find more information on estimating the size of a database on Microsoft's website at `http://msdn.microsoft.com/en-us/library/ms187445.aspx`.

Setting the Recovery Model

After you have selected the name, owner, and size of your data warehouse, you may want to move to the options page and change the recovery model. The recovery model controls the behavior of a database transaction log file. Setting this correctly can increase your data warehouse loading performance.

The three settings are Full, Simple, and Bulk-Logged. Each functions slightly differently. In both full and simple modes, the log file will record every row that you add to the table. The difference between these two modes is that in simple mode, the transaction log file is periodically cleared automatically. When the database is set to full mode, the log file continues to fill up until you back up the log file. In an OLTP database, it is best practice to put the database in full mode and do regular backups on the transaction log. This makes sense, given that the purpose of an OLTP database is to process transactions. Therefore, making sure that you have a backup of these transactions is important. In an OLAP database, however, the focus is not on transactions but on analysis of data. Because of this, using full mode is not necessary because you do not need to make backups of the individual transactions.

A SQL Server transaction will occur whenever you insert, update or delete data from a table. Consequently, you will still end up with transactional events each time you import to the data warehouse. In both simple and full mode, if you import a million rows, the log records a million insert transactions. Therefore, regardless of whether you use the simple or the full, the log file records each and every insert.

Using the full mode increases the size of the log files substantially because it is not automatically cleared for you but instead relies on regular backups. Using the simple mode, which has automatic truncations, will keep the log file size small, although it still has the extra overhead of tracking each and every insert. This brings us to the third option: bulk-logged.

The bulk-logged option may be a more logical choice for an OLAP database. Figure 5-14 shows this selection. With this setting, large imports of data are recorded with the minimal amount of transaction log entries, which provides additional performance during data warehouse loading and minimal impact on the size of the transaction log. You will, however, still need to perform regular backups on the log file to clear it out. Also, there will still be times when bulk-logged is not the best choice, such as when you have slow-changing columns that you could lose data from, as we discuss in a moment.

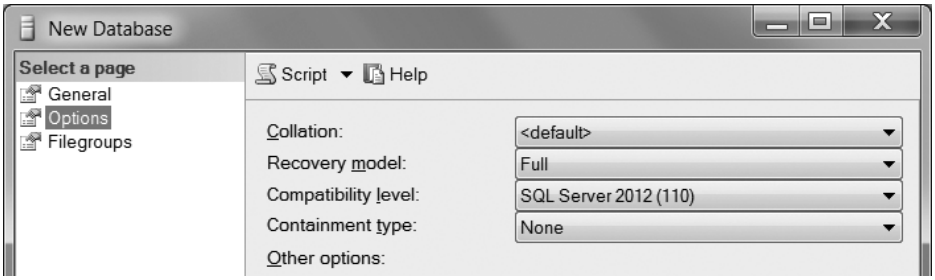

Figure 5-14. *Setting the recovery model*

Performing Database Backups

Backing up the transaction log is quite simple, and you can do it with a single line of SQL code. You cannot back up the log, however, unless you have already backed up the whole database at least once. You must do this even if you are not going to use the database backup, but you can reuse the same backup file multiple times by clearing its contents using the `INIT` option. Listing 5-2 shows a full database backup, followed by a transaction log backup to the same file.

Listing 5-2. Backing Up the Log File

```
BACKUP DATABASE [DWWeatherTracker] TO DISK=N'C:\_BISolutions\DWWeatherTracker.bak'
GO
BACKUP LOG [DWWeatherTracker] TO DISK=N'C:\_BISolutions\DWWeatherTracker.bak' WITH INIT
```

When you back up the transaction log, it clears out entries in the log file, freeing up space for more log entries. Performing regular backups of the transaction log keeps the transaction log file from growing on your hard drive. But, remember that the backup file(`.bak`) increases in size unless you clear this file each time you perform a backup. Initializing the backup file clears it, allowing you to maintain a smaller backup file. The `WITH INIT` command triggers initialization (Listing 5-2).

Shrinking Log Files

Occasionally, when you do a very large import, the transaction log file may temporarily need to expand. SQL Server does this for you automatically, but it does not automatically shrink the file back to its original sizes afterwards. You manually shrink the file by first backing up the transaction log to clear out its contents, and then using the shrink file option to reduce its size. The code in Listing 5-3 does just that.

Listing 5-3. Shrinking the Log File

```
USE [DWWeatherTracker]
GO
BACKUP LOG [DWWeatherTracker] TO DISK=N'C:\_BISolutions\DWWeatherTracker.bak' WITH INIT
GO
DBCC SHRINKFILE (N'DWWeatherTracker_log' , 0, TRUNCATEONLY)
GO
```

■ **Note** In previous versions of SQL, Microsoft included a `TRUNCATE_ONLY` option for clearing the log file entries without having to back them up. But, this caused problems that could introduce corruption into your database if not handled properly. Therefore, since SQL 2008, Microsoft removed the ability to clear log file entries with the

TRUNCATE_ONLY command, requiring log file entries to be backed up before truncation. Oddly, Management Studio offers a TRUNCATEONLY option for the DBCC SHRINKFILE command, but it is only applicable to data files. We are able to use it in Listing 5-3 because it does not cause corruption even when shrinking a transaction log, and it does not require data files to be backed up before truncation.

Keeping Data Warehouse Backups

As any database administrator (DBA) will attest, database backups are vital to every business. Still, you may not need to keep a series of database backups for your data warehouse to the same degree as expected of an OLTP database. In an OLTP database, a daily cycle of full and transactional backups is recommended. Data warehouse databases, however, may be updated only once a night or once a week and do not require repeated backups during the middle of the day or even daily. In point of fact, it is sometimes argued that because you can rebuild the entire data warehouse from the OLTP database—which is continually backed up—you do not need a backup of the data warehouse. If the data warehouse ever crashes, you can always rebuild it and reload.

This approach makes the DBAs very nervous, sometimes for good reason. Many OLTP databases are not designed to track changing dimensional values over time. When an update or deletion occurs, previous data in a column and table is lost. On the other hand, in a data warehouse this data is often tracked in some, if not all, dimensions. If your data warehouse tracks changes to dimensional values and the OLTP database does not, you need to retain your backups.

This is an important point, so let's take a look at an example. Suppose you have a table that is tracking changes to product prices where the standard retail price of a product can change as time goes on. In the data warehouse, you could add dimension columns to track these changes. (Tracking columns are referred to as *slow-changing* dimension columns, as discussed in Chapter 4.)

In the example in Figure 5-15, you see two different table designs: an OLTP table called Products without any slow-changing dimension columns and an OLAP table called DimProducts. The DimProducts table includes four slow-changing dimension columns (RecordStartDate, RecordEndDate, IsCurrent and Productkey).

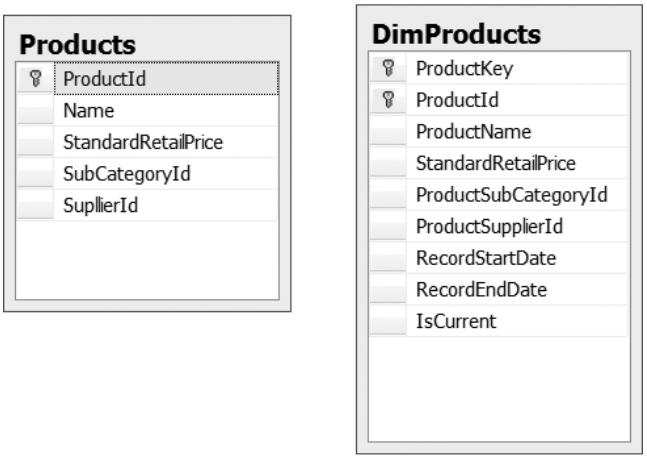

Figure 5-15. *Tables with and without slow-changing dimension columns*

If the standard retail price is overridden in the Products table, the original price will not have been lost, but rather it will be recorded in the DimProducts table during the ETL process. In a perfect world, the slow-changing dimension columns would have been included in the Products table and tracked in the OLTP environment. There are times, however, where you will have no control over the design of the OLTP environment but may

have control over the OLAP environment. And if that is the case, you will be left implementing slow-changing dimension tracking similar to this example (Figure 5-15).

Were the data warehouse database to crash, you would not be able to reload it from the current OLTP database, but rather, it must be reloaded from the backups. In this scenario, you need to maintain a set of data warehouse backups.

Using the Filegroups Option

The Filegroups page of the New Database window allows you to create and configure filegroups for your data warehouse (Figure 5-16). Using filegroups can increase performance, but doing so adds complexity. With filegroups, you can control the placement of your fact and dimension tables in a specific set of one or more files. These files, in turn, are located on separate hard drives allowing the database engine to read and write from several hard drives at once. With this design, not only will you benefit from ETL performance but also from data retrieval performance.

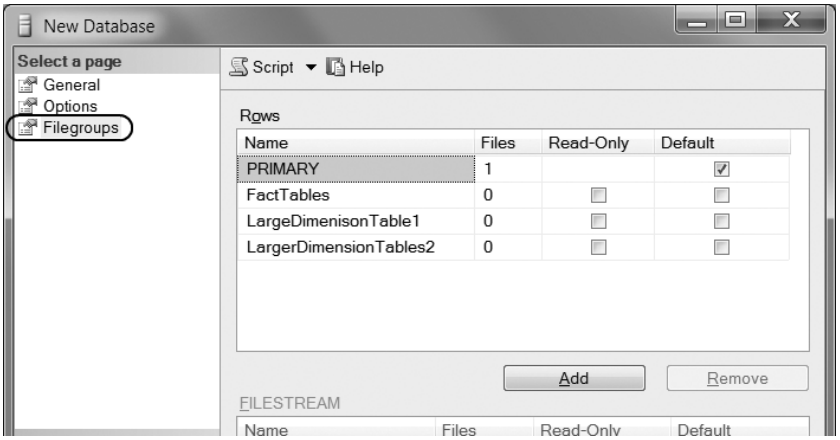

Figure 5-16. *Creating filegroups*

Every database contains a filegroup called *primary*, and a master data file (`.MDF`) within it. All the internal system tables are mapped to the primary filegroup and are therefore placed in the `.MDF` file.

To create an additional filegroup, just click the Add button for each filegroup you want to create. As an example, you might create one filegroup to store your fact tables, another filegroup to store one set of dimension tables, and yet a third to store other dimension tables.

At first filegroups have no files associated with them. To use the filegroup, you must go back to the General page (Figure 5-16), click the Add button to create a new file, type in a logical file name, select the file group you want to use, define a file path, and type in a physical file name (Figure 5-17). Other options are available as well, but these are the common ones.

■ **Note** The dropdown box showing the filegroups (Figure 5-17) "automagically" appears only when you click the file group cell; it can be confusing because it is set to Primary by default with no dropdown option in sight!

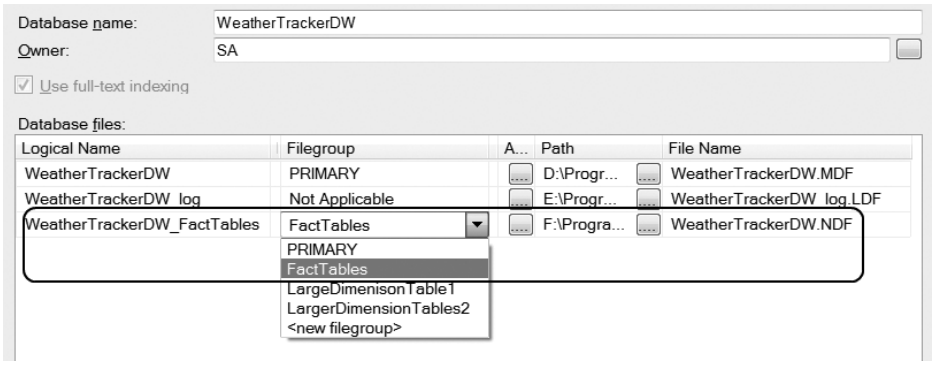

Figure 5-17. *Adding files to a filegroup on the General page*

When creating the new file, be sure to change the Path property of that file to point to a particular hard drive, as shown in Figure 5-17. The whole point of using filegroups is to be able to dictate which hard drive to use to store a particular table's data. If you go to the trouble of creating filegroups and files without distributing the files on multiple hard drives, you have just added administrative overhead with no benefit.

In addition, you should name your file with an `.NDF` extension rather than an `.MDF` or `.LDF` extension. All files beyond the original `.MDF` file are given the extension of `.NDF`. The explanation of the file names are as follows: the `.MDF` file is your main data file, the `.LDF` is your log file, and any `.NDF` files are every other additional data file. If you add six additional files, they will all have `.NDF` as their extension.

■ **Note** These extensions are not case sensitive; they are capitalized here for clarity.

Once you have the filegroups and files created, you can add a table in a particular filegroup by using the syntax shown in Listing 5-4. Although you cannot determine which file a table goes into, you can determine which file group a table belongs to. This has the advantage of allowing you to have multiple files in a single file group. If each file in the same filegroup is on a different hard drive, your table will span multiple hard drives.

Listing 5-4. Creating Tables in a Particular Filegroup

```
/* (Note the following code is expected to error out and is only a demo) */

CREATE TABLE [dbo].[FactWeather]
([Date] [datetime] NOT NULL,
 [EventKey] [int] NOT NULL,
 [MaxTempF] [int] NOT NULL,
 [MinTempF] [int] NOT NULL,
 CONSTRAINT [PK_FactWeathers] PRIMARY KEY CLUSTERED ( [Date] ASC,[EventKey] ASC )
) ON [FactTables] -- Name of the File Group not the file!
```

■ **Note** We cover filegroups in this chapter because the Filegroups page is part of the New Database creation window, but you most likely will not need to use them unless you are working with very large data warehouses. Be aware that filegroups can give increased performance when reading and writing to multiple hard drives but do not provide any fault tolerance to hard drive failure. If one drive fails, you have to replace the failed drive and restore your most recent backup or at least refill the Data Warehouse with all of the OLTP data. Although this does

not represent a total tragedy, you may still want to avoid this by using RAID technology, which provides similar performance, simpler administration, and automatic recovery from a hard drive failure. It is also easier from a BI designer perspective, because the network team will set it up for you most of the time!

EXERCISE 5-1. CREATING THE PUBLICATION INDUSTRIES DATABASE

In this exercise, you create a database for the Publication Industries data warehouse.

Important: You practice administrator-level tasks in this book, so you need administrator-level privileges. The easiest way to achieve this is to right-click the menu item, select Run as Administrator, and then answer Yes to access administrator-level privileges while running this program. In Windows 7 and Vista, just logging in with an administrator account is not enough. For more information, search the Web for "Windows 7 True Administrator and User Access Control."

1. Open SQL Server Management Studio 2012. (You can do so by clicking the Start button and navigating to All Programs ➤ Microsoft SQL Server 2012 ➤ SQL Server Management Studio. Right-click SQL Server Management Studio 2012 and click the Run as Administrator menu item. If the UAC message box appears asking, "Do you want the following program to make changes to this computer?" click Yes [or Continue depending upon your operating system] to accept this request.)
2. When Management Studio opens, choose to connect to the database engine by selecting this option in the Server Type dropdown box. Then click the Connect button to connect to the database engine (Figure 5-2).
3. Use the Query Window to create a database using the code in Listing 5-5.

Tip: The files for this exercise, as well as all of the exercises throughout this book, are available in the downloadable book content.

Listing 5-5. Creating the DWPubsSales Data Warehouse Database

```
USE [master]
GO
IF EXISTS (SELECT name FROM sys.databases WHERE name=N'DWPubsSales')
  BEGIN
    -- Close connections to the DWPubsSales database
    ALTER DATABASE [DWPubsSales] SET SINGLE_USER WITH ROLLBACK IMMEDIATE
    DROP DATABASE [DWPubsSales]
  END

GO
CREATE DATABASE [DWPubsSales] ON PRIMARY
( NAME=N'DWPubsSales'
, FILENAME=N'C:\_BISolutions\PublicationsIndustries\DWPubsSales.mdf'
, SIZE=10MB
, MAXSIZE=1GB
, FILEGROWTH=10MB )
LOG ON
```

```
( NAME = N'DWPubsSales_log'
, FILENAME = N'C:\_BISolutions\PublicationsIndustries\DWPubsSales_log.LDF'
, SIZE = 1MB
, MAXSIZE = 1GB
, FILEGROWTH = 10MB)
GO
EXEC [DWPubsSales].dbo.sp_changedbowner @loginame = N'SA', @map = false
GO
ALTER DATABASE [DWPubsSales] SET RECOVERY BULK_LOGGED
GO
```

If you are typing the SQL code, highlight and execute the code by clicking the "! Execute" button.

In this exercise, you created the database that is used as the Publication Industries sales data warehouse. Next you need to fill it with fact and dimension tables.

Creating Tables

As you saw from Listing 5-5, creating a database with SQL code is pretty straightforward. However, many developers prefer to use the New Database dialog windows instead.

Creating a table is just as easy, and as with creating a database, you can use either code or designer windows. SQL Server Management Studio has two tools for designing tables. The first one we look at is the table designer.

■ **Note** Different versions of SQL Server may not include all the designer tools mentioned here. In this book, we assume that you are using SQL Server's developer edition, which includes all SQL Server's designer tools. For more information, see SQL Books Online.

Using the Table Designer

SQL Server Management Studio's table designer provides a graphical way to create tables by using a designer window along with a property window. To create a new table with the table designer, start by using Object Explorer to select the database in which you want to create the table, and then expand the treeview until you see the Tables folder. From there, right-click the Tables folder to get a context menu that allows you to add a new table (Figure 5-18).

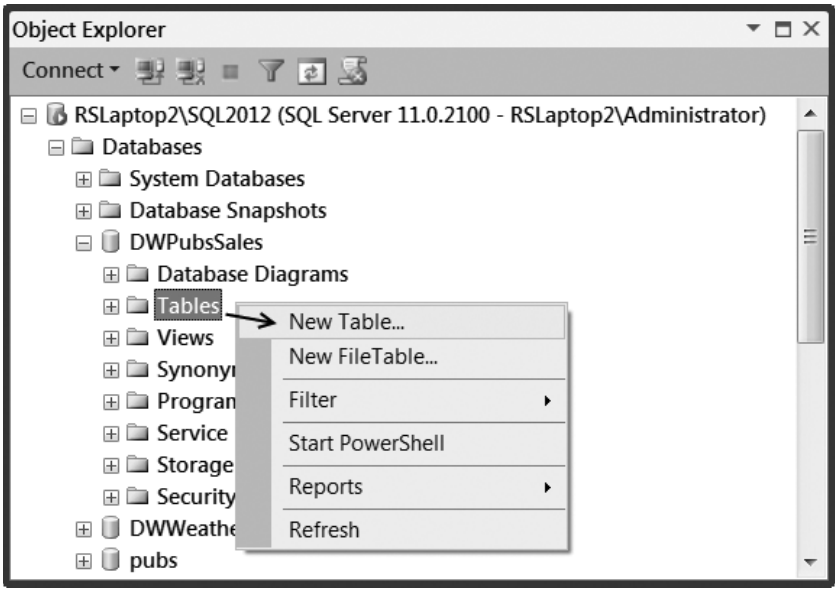

Figure 5-18. *Launching the table desiner*

By default, the table designer window opens in the center of SQL Server Management Studio, and the Properties window displays to the right of it (Figure 5-19). The Properties window does not always appear unless you press the F4 key or use Management Studio's View ➤ Properties Window menu item (Figure 5-19).

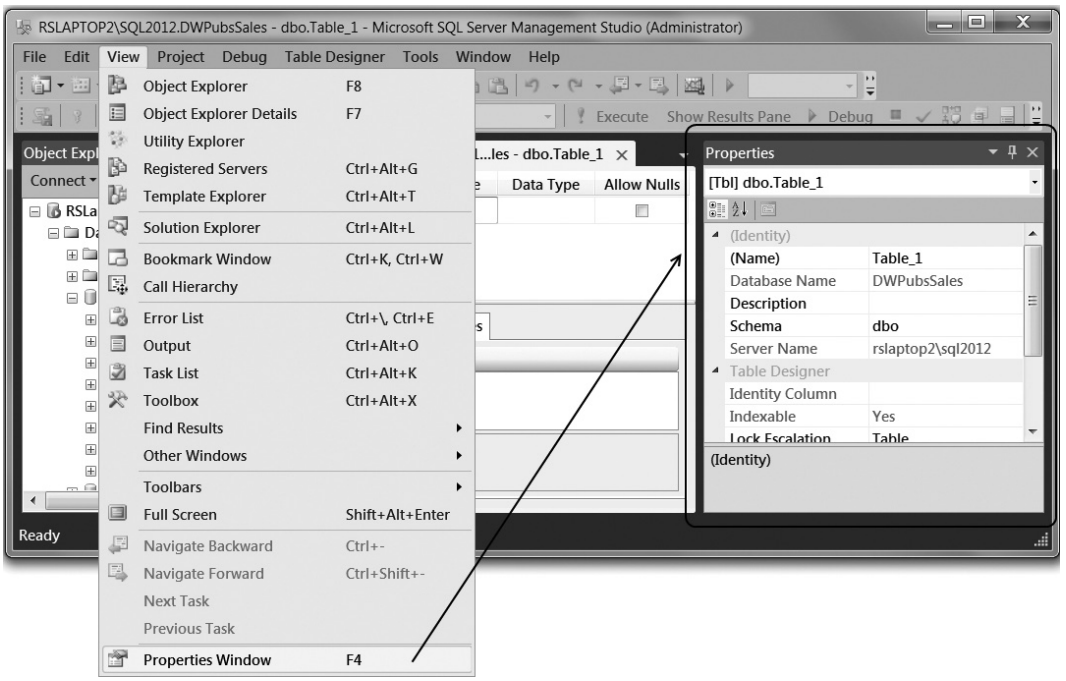

Figure 5-19. *Displaying the Properties window*

■ **Note** The window in SQL Management Studio can be repositioned, so your screen may not look the same as shown in the figure. You can reset the layout to its default settings at any time using the Window ➤ Reset Windows Layout menu item.

When you open the table designer initially, there will be no columns specified. Therefore, your first job is to type in the column name, choose a data type, and then choose whether to allow nulls. Additional settings can be added as well using the Properties window. For example, if the first column was a customer ID column and you decided to use the autonumbering feature in SQL Server known as the identity setting, you could use the Properties window to set the identity column on the table, as shown in Figure 5-20.

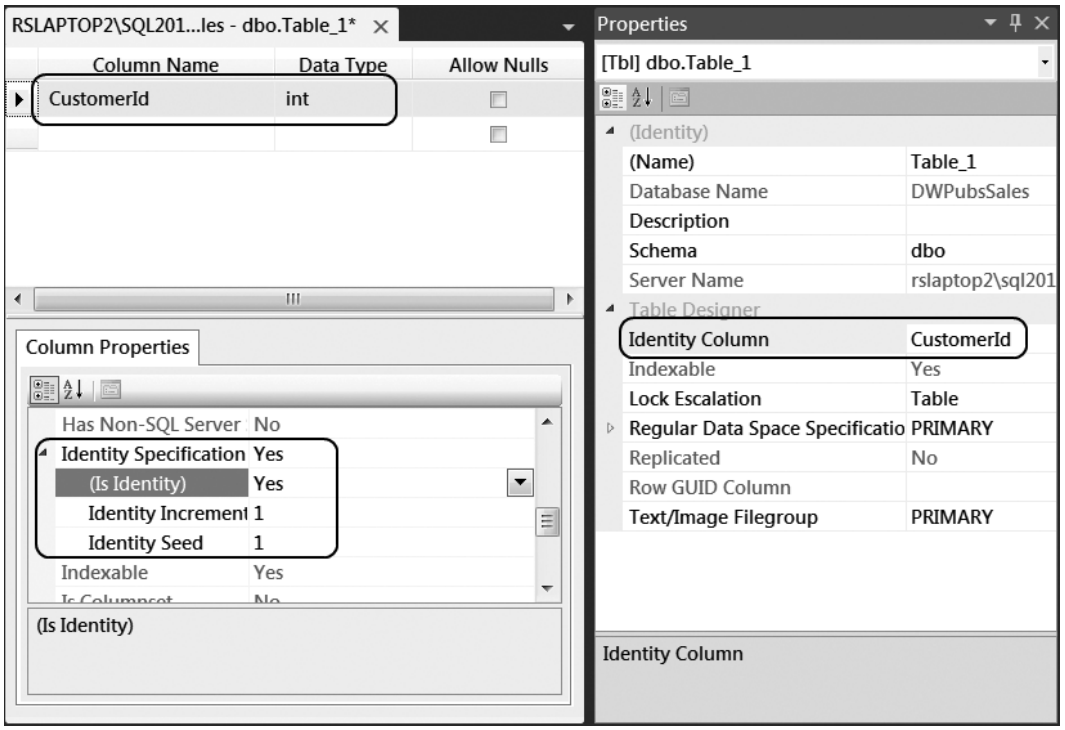

Figure 5-20. *Setting column properties*

You can easily add a primary key to the table by right-clicking a column and selecting the Set Primary Key option from the context menu. If you want to create a composite primary key constraint on a table, this is also easily accomplished. To do so, you must select both columns at once by holding down the Control button as you click each column. After that, you can right-click to get the context menu and select the Set Primary Key option to create a composite primary key constraint (Figure 5-21).

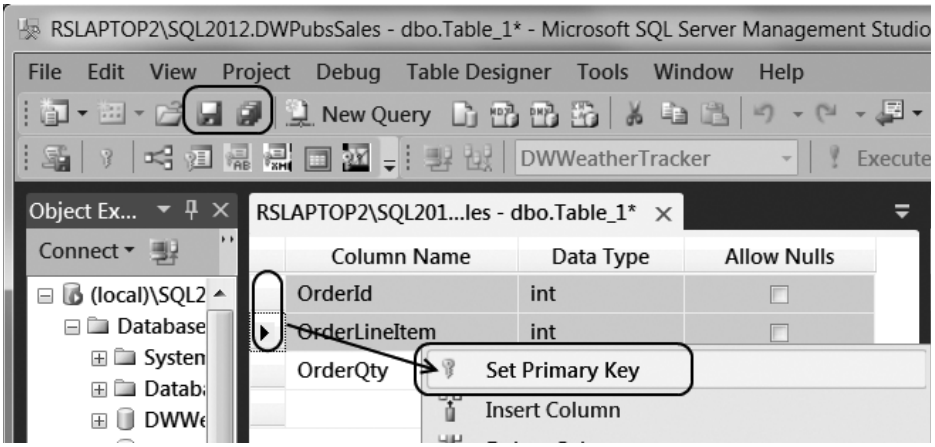

Figure 5-21. *Setting a composite primary key constraint*

Once you have designed the table to your satisfaction, save your work by using the File menu and choosing the Save Table option. You can also click the button on the toolbar that looks like a floppy disk (Figure 5-21). Upon saving, you will be prompted for a table name, unless if you have configured a new name in the Properties window already (Figure 5-20).

Generating SQL Scripts

The SQL Server Management Studio's designer tool invisibly creates SQL code for you each time you create or modify a table. You can access this code by using the Generate Change Script menu option, as shown in Figure 5-22.

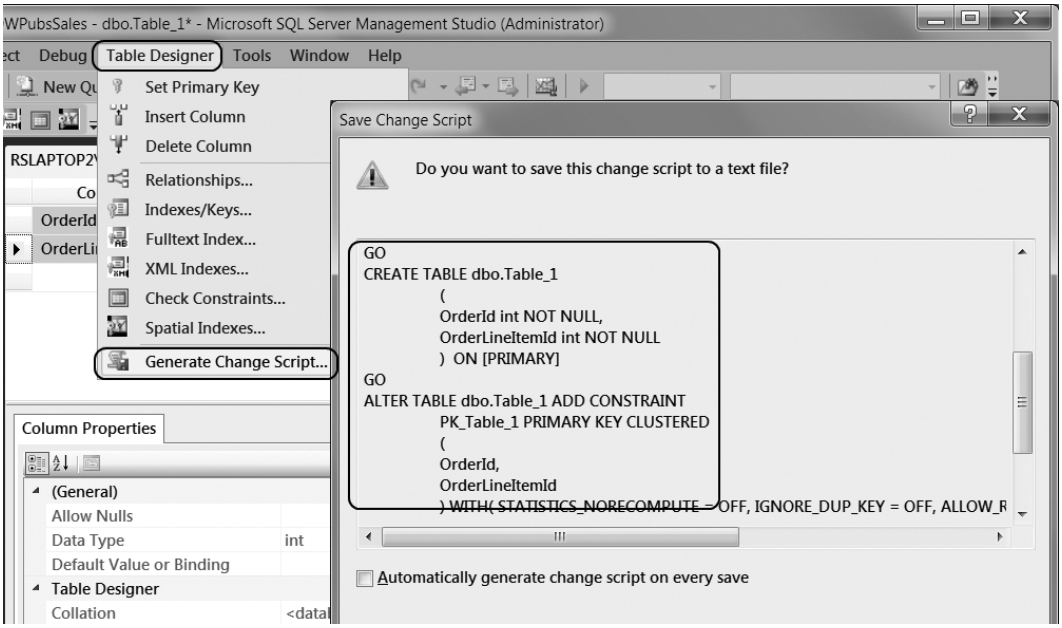

Figure 5-22. *Generating a change script*

Be aware that menus in Management Studio are context sensitive, meaning that the table designer window will not be available unless the table designer is open. It also means that if there are no changes since the change script was last run, this menu item is grayed out, because there are no changes recorded.

After you have saved your table, you can continue to work on it and save it again, or you can close the designer window by clicking the small *x* on the table designer tab (Figure 5-21). Each time you save and make a new change, a new script is created.

Changing an Existing Table

If you wish to work on an existing table you can reopen the table designer by right-clicking the table in Object Explorer and selecting Design from the context menu. When the table designer opens, you can modify the table's current columns, or add a column by clicking the blank row at the bottom of the column list. Alternately, you can add a column by right-clicking existing columns and choosing the Insert Column option from the context menu.

In the table designer, changes to a table sometimes force it to be re-created behind the scenes. When this happens, SQL Server Management Studio will do the following:

1. Create a temporary table.
2. Import data from an original table into the temporary table.
3. Delete the original table.
4. Rename the temporary table to the original table name.

This re-creation process could take a long time if the table had a lot of data. Because of this, Microsoft's default setting does not allow for operations that force a table re-creation. You can, however, change the SQL Server Management Studio options to allow this to occur. To do so, go to the Tools ➤ Options menu item. An Options dialog window will open, as shown in Figure 5-23. You need to navigate down to the Designers section in the treeview of this dialog window and then expand the section before being able to see the Tables and Database Designers page. Navigate to this page and look for the "Prevent saving changes that require table re-creation" checkbox. Uncheck this box, and then click OK.

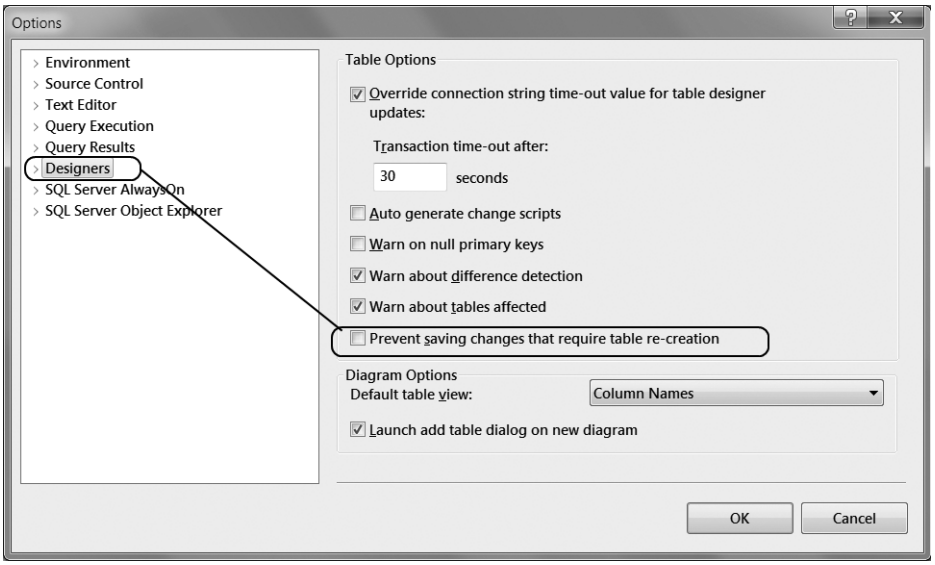

Figure 5-23. *Allowing table re-creation in the designers page*

Once you have unchecked this option, you will be able to make changes in the table designer that force a re-creation event to occur. You will not notice the difference overtly, except that certain items you may have wanted to change previously, such as the primary key settings, will now work; whereas they did not before.

Using the Diagramming Tool

Although creating a table with the table designer is easy, you may prefer another method even more: using the diagramming tool. To create a database diagram, right-click the diagram folder in Object Explorer and use the context menu to create a new diagram, as shown in Figure 5-24.

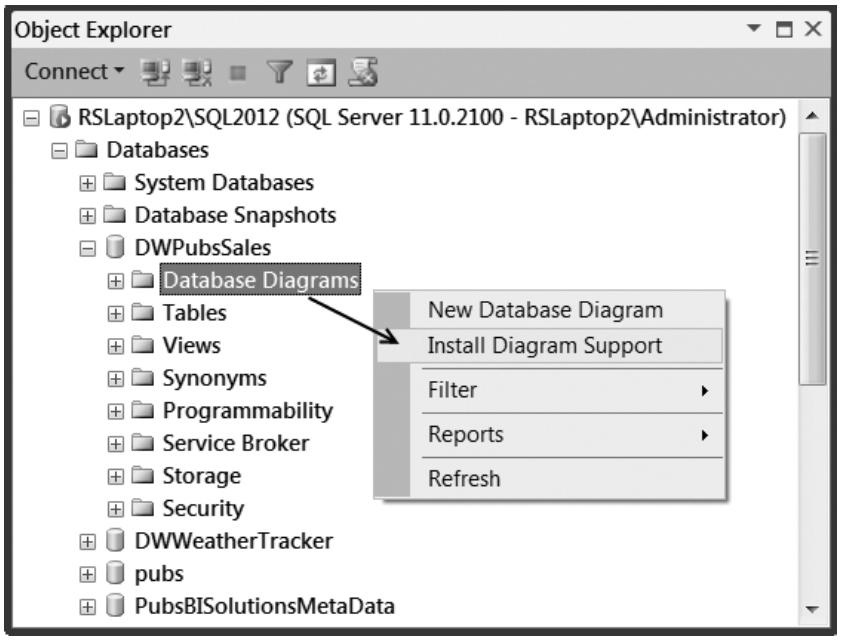

Figure 5-24. *Installing diagram support*

The SQL Server database diagram provides a schematic of a database's tables and their relationships, which can be quite helpful in understanding a database's design. In addition, the diagramming tool also allows you to create, edit, and delete tables. Each database can have one or more diagrams. Diagrams are stored within database system tables, which can be backed up along with other database objects and data.

SQL Server database diagrams have a couple of useful features you should know about. Firstly, you can show all or only some of the tables in the database. This is convenient because you can make diagrams specific to a particular subject matter, which is a big plus when you have a data warehouse with several data marts inside. Secondly, multiple database diagrams can reference the same table, which is advantageous when you have conformed dimensions that are shared between different data marts.

To create a database diagram, you must install some supporting objects within the database. If you have not installed the support objects before you attempt to make a database diagram, you will be prompted to install them.

■ **Tip** Although you can select the New Database Diagram option in the context menu before the Install Diagram Support option, no matter what selection you choose, you have to install the supporting objects (which makes you wonder why they even bothered including a menu option). As such, a message-box may appear informing you that

it needs to create the supporting object when you click the New Database Diagram option first. Simply close the message box when it appears by clicking the Yes button.

Each time you create a new database diagram, an Add Table dialog box appears, as shown in Figure 5-25. This allows you to select which table to add to the new diagram. You can select one or all of the tables in the dialog box. And, you can only select tables that exist in the same database as the diagram.

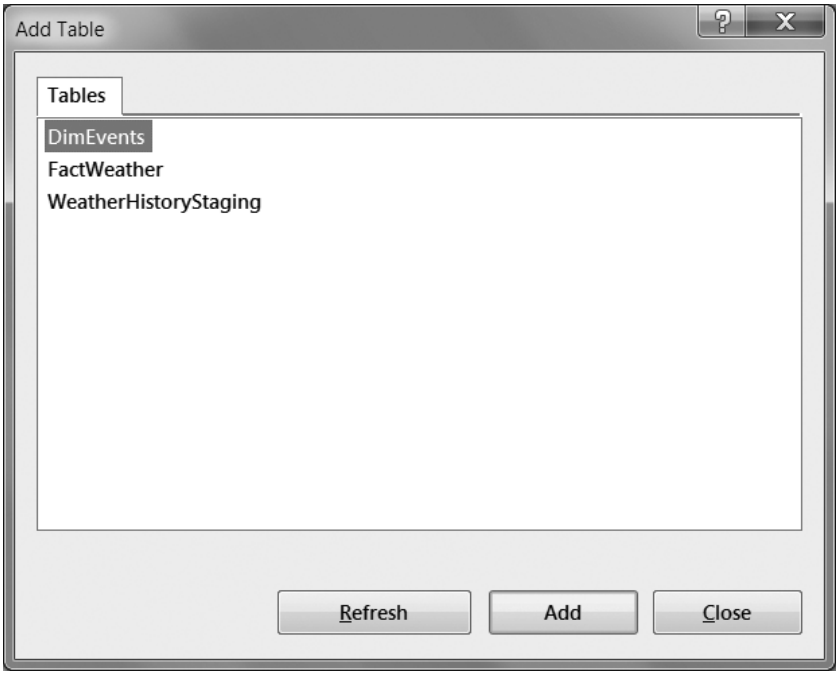

Figure 5-25. *Adding existing tables to a diagram*

Creating New Tables with the Diagram Tool

If there are no tables in the database, the Add Table dialog box appears empty. This is not a problem per se, but it is often disconcerting. If you have no tables in the database, yet want to use a diagramming tool to create them, just close the empty Add Table dialog box and then create a new table (Figure 5-26). Remember that if the diagram does not include any existing tables, you have the equivalent of a blank sheet of paper. Once again, this may initially be confusing, but it is perfectly normal.

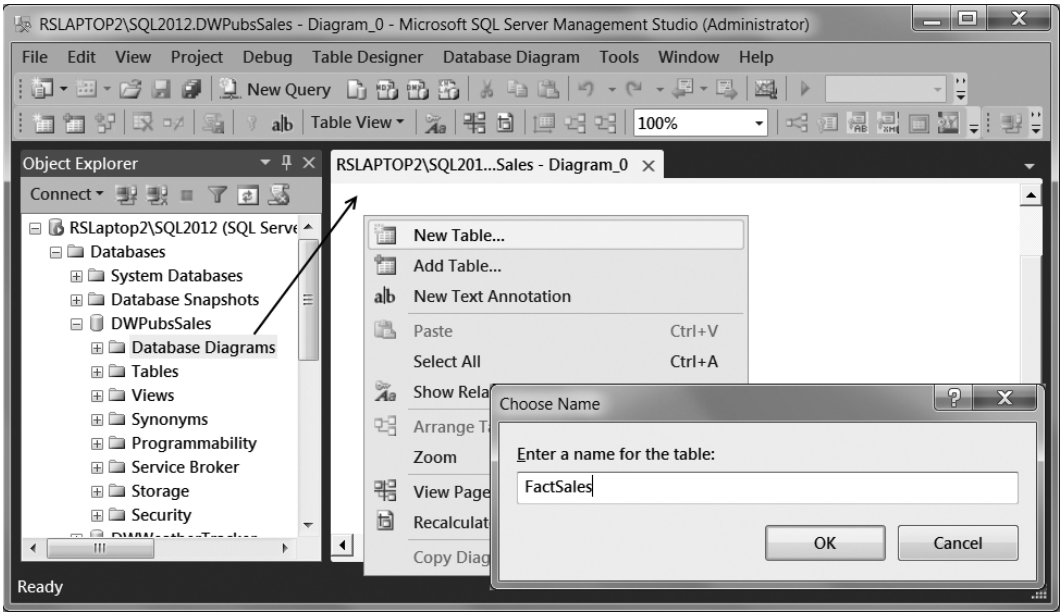

Figure 5-26. *Adding a table to the diagram*

Whether you have tables in the diagram already or the diagram is completely blank makes no difference to your ability to add a new table. You can easily add a new table to the diagram by right-clicking a blank spot and selecting the New Table option from the context menu, as shown in Figure 5-26.

Unlike the table designer, which asks you for the name of the table after you save it, the diagramming tool asks you for the name of the table straightaway, as shown in Figure 5-26. In addition, you can change the name later by modifying the properties of the property sheet as you did with the table designer.

A square representing your table is presented on the diagram immediately after you click OK in the Choose Name dialog box. At that point, you can add columns just as you did in the table designer. You can set the data type and the null special specification as well. You can also set primary keys by clicking to select the column or columns if you are using a composite primary key and select the Set Primary Key from the context menu, as shown in Figure 5-27.

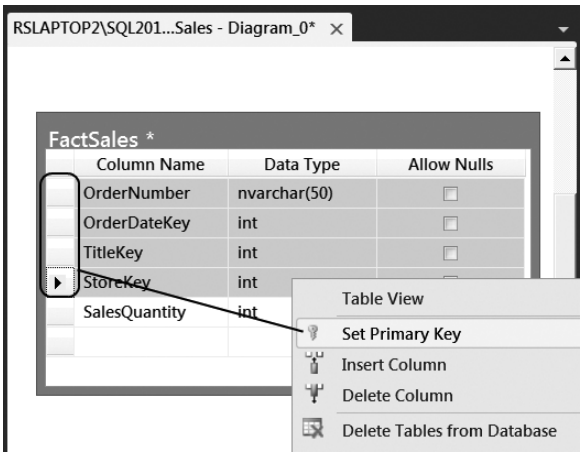

Figure 5-27. *Adding a primary key constraint in the diagram tool*

A classic way to use a diagramming tool is to document your table designs using the Excel spreadsheet and then use the worksheet to create the new tables, much as a carpenter uses a blueprint to build a house. As you can see in Figure 5-28, we created a new fact table while having both the diagramming tool and an Excel spreadsheet open.

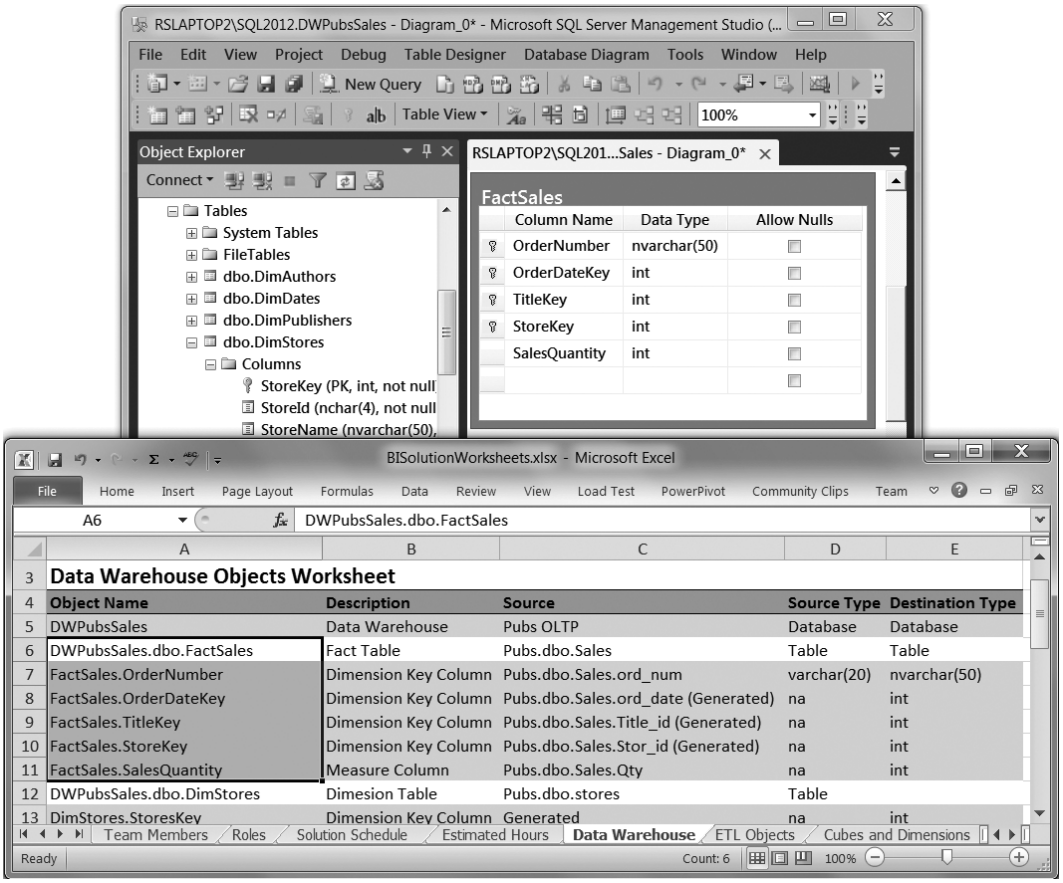

Figure 5-28. *Using a BI solution plan as a blueprint*

Creating Foreign Keys with the Diagramming Tool

Once you have more than one table in the diagram, you can easily create foreign key constraints between them. You can accomplish this by dragging and dropping between the columns where you want to place a foreign key constraint, as shown in Figure 5-29.

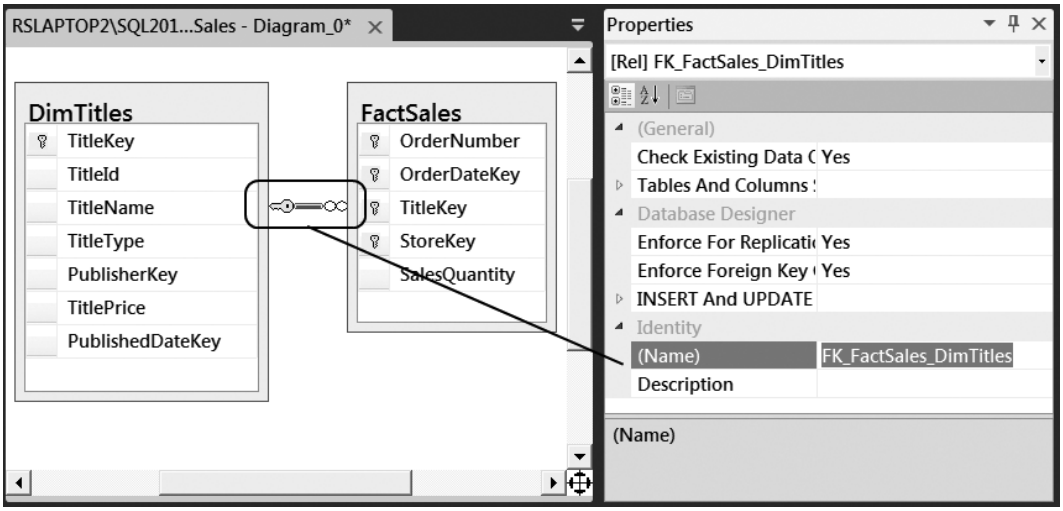

Figure 5-29. *Tables with a foreign key constraint*

Here are the steps:

1. Click the foreign key or child column.
2. With the mouse button still depressed, drag the mouse cursor to the primary key or parent column.
3. Release the mouse button.

We have seen many struggle with this method; therefore, it may take some practice before you become comfortable with it. Perhaps one reason for the difficulty is that the diagram tool does not place the foreign key constraint line directly next to the columns being used but instead shows the connection between the tables themselves. And although you can move the connecting line toward the columns manually to make it look more presentable, it is not done automatically. Once you get used to it, you will find this technique is not as confusing as it may seem at first. And as long as the columns are named similarly, you will easily be able to figure out which columns are interconnected.

Using the Query Window

You can also use code to create your tables. For some developers, this is the most straightforward way to do so. Listing 5-6 shows the syntax for creating a table with code.

Listing 5-6. Creating the Fact Table with SQL Code

```
CREATE TABLE [dbo].[FactSales](
  [OrderNumber] [nvarchar](50) NOT NULL,
  [OrderDateKey] [int] NOT NULL,
```

```
  [TitleKey] [int] NOT NULL,
  [StoreKey] [int] NOT NULL,
  [SalesQuantity] [int] NOT NULL,
  CONSTRAINT [PK_FactSales] PRIMARY KEY CLUSTERED
    ( [OrderNumber] ASC,[OrderDateKey] ASC, [TitleKey] ASC, [StoreKey] ASC )
)
GO
```

By typing this code into a query window and clicking the "! Execute" button, the table is created with the composite primary key on the first four columns. After you create all the tables in a database, you can create foreign key constraints between the tables using code similar to that shown in Listing 5-7.

Listing 5-7. Adding Foreign Key Constraints

```
ALTER TABLE [dbo].[FactSales] WITH CHECK ADD CONSTRAINT [FK_FactSales_DimStores]
FOREIGN KEY([StoreKey]) REFERENCES [dbo].[DimStores] ([Storekey])
GO

ALTER TABLE [dbo].[FactSales] WITH CHECK ADD CONSTRAINT [FK_FactSales_DimTitles]
FOREIGN KEY([TitleKey]) REFERENCES [dbo].[DimTitles] ([TitleKey])
GO
```

Note that in Listing 5-7 there are two foreign key constraint commands. If a table has multiple foreign key relationships, as most fact tables do, you need to execute a separate command for each of them.

At this point, you have seen three ways to create tables: using SQL code, using the table designer and using the diagramming tool. All three ways accomplish the same thing.

Now, after so many pages of theory, it is time to do another exercise!

EXERCISE 5-2. CREATING TABLES AND FOREIGN KEY CONSTRAINTS

In this exercise, you create the tables and foreign key constraints in the Publication Industries data warehouse. You can choose to use the SQL code presented here, the table designer or the diagramming tools to accomplish your goal.

The files for this exercise, as well as all the exercises throughout this book, are available in the downloadable book content.

Creating the Tables

The first thing we need to do is create the tables. Let's do that now.

1. If it is not open already, open SQL Server Management Studio (see Exercise 5-1 for more details).
2. Decide on a method for creating the tables and begin creating them utilizing the table names, column names, data types, nullablity, and primary keys represented in Listing 5-8. You can find a copy of this SQL script in the downloadable book files.

   Listing 5-8. Creating the DWPubsSales Tables

   ```
   USE [DWPubsSales]
   GO

   /****** Create the Dimension Tables ******/
   ```

```
CREATE TABLE [dbo].[DimStores](
  [StoreKey] [int] NOT NULL PRIMARY KEY Identity,
  [StoreId] [nchar](4) NOT NULL,
  [StoreName] [nvarchar](50) NOT NULL
)
GO

CREATE TABLE [dbo].[DimPublishers](
  [PublisherKey] [int] NOT NULL PRIMARY KEY Identity,
  [PublisherId] [nchar](4) NOT NULL,
  [PublisherName] [nvarchar](50) NOT NULL
)
GO

CREATE TABLE [dbo].[DimAuthors](
  [AuthorKey] [int] NOT NULL PRIMARY KEY Identity,
  [AuthorId] [nchar](11) NOT NULL,
  [AuthorName] [nvarchar](100) NOT NULL,
  [AuthorState] [nchar](2) NOT NULL
)
GO

CREATE TABLE [dbo].[DimTitles](
  [TitleKey] [int] NOT NULL PRIMARY KEY Identity,
  [TitleId] [nvarchar](6) NOT NULL,
  [TitleName] [nvarchar](100) NOT NULL,
  [TitleType] [nvarchar](50) NOT NULL,
  [PublisherKey] [int] NOT NULL,
  [TitlePrice] [decimal](18, 4) NOT NULL,
  [PublishedDateKey] [int] NOT NULL
)
GO

/****** Create the Fact Tables ******/

CREATE TABLE [dbo].[FactTitlesAuthors](
  [TitleKey] [int] NOT NULL,
  [AuthorKey] [int] NOT NULL,
  [AuthorOrder] [int] NOT NULL,
CONSTRAINT [PK_FactTitlesAuthors] PRIMARY KEY CLUSTERED
    ( [TitleKey] ASC, [AuthorKey] ASC )
)
GO

CREATE TABLE [dbo].[FactSales](
  [OrderNumber] [nvarchar](50) NOT NULL,
  [OrderDateKey] [int] NOT NULL,
  [TitleKey] [int] NOT NULL,
  [StoreKey] [int] NOT NULL,
  [SalesQuantity] [int] NOT NULL,
CONSTRAINT [PK_FactSales] PRIMARY KEY CLUSTERED
    ( [OrderNumber] ASC,[OrderDateKey] ASC, [TitleKey] ASC, [StoreKey] ASC )
)
GO
```

3. If you are typing the SQL code, highlight and execute the code by clicking the "! Execute" button. If you choose to use the table designer or the diagramming tool, make sure that you use the Save button to create the tables in the database.

Adding Foreign Key Constraints

Now that the tables are created, we want to add foreign key constraints between the tables.

1. Using either SQL code or the diagramming tool, create foreign key constraints as described by the code in Listing 5-9.

Listing 5-9. Creating the DWPubsSales Foreign Key Constraints

```
/****** Add Foreign Keys ******/
ALTER TABLE [dbo].[DimTitles] WITH CHECK ADD CONSTRAINT [FK_DimTitles_DimPublishers]
FOREIGN KEY([PublisherKey]) REFERENCES [dbo].[DimPublishers] ([PublisherKey])
GO

ALTER TABLE [dbo].[FactTitlesAuthors] WITH CHECK ADD CONSTRAINT
[FK_FactTitlesAuthors_DimAuthors]
FOREIGN KEY([AuthorKey]) REFERENCES [dbo].[DimAuthors] ([AuthorKey])
GO

ALTER TABLE [dbo].[FactTitlesAuthors] WITH CHECK ADD CONSTRAINT
[FK_FactTitlesAuthors_DimTitles]
FOREIGN KEY([TitleKey]) REFERENCES [dbo].[DimTitles] ([TitleKey])
GO

ALTER TABLE [dbo].[FactSales] WITH CHECK ADD CONSTRAINT [FK_FactSales_DimStores]
FOREIGN KEY([StoreKey]) REFERENCES [dbo].[DimStores] ([Storekey])
GO

ALTER TABLE [dbo].[FactSales] WITH CHECK ADD CONSTRAINT [FK_FactSales_DimTitles]
FOREIGN KEY([TitleKey]) REFERENCES [dbo].[DimTitles] ([TitleKey])
GO
```

2. If you are typing the SQL code, highlight and execute the code by clicking the "! Execute" button. If you choose to use the table designer or the diagramming tool, make sure you use the Save button to create the tables in the database.

Verifying Your Tables

Once the tables are made and the foreign keys created, it is best to check your work by comparing it to the design document, the Publication Industries BI solution worksheet you reviewed in Chapter 3.

1. Open the file `C:\_BISolutions\PublicationsIndustries\BISolutionWorksheets.xlsx` in Microsoft Excel, and verify that you have created your data warehouse accurately (Figure 5-28).

In this exercise, you created the data warehouse tables and foreign key constraints. Currently these tables are empty. In Chapters 6 through 8 we create an ETL process to fill them.

Creating a Date Dimension Table

One common practice in data warehousing is the creation of a date dimension table. We discussed the use of these tables in Chapter 4 but have not discussed how to create them. There are a number of ways to accomplish this, but using SQL code is probably the most common way to do so.

As a standard practice, the creation of a date dimension table includes a surrogate key in the form of an integer value, a natural date key in the form of a datetime value and other descriptive attributes such as month, quarter, and year. Listing 5-10 shows an example of a date dimension table being created with this design.

Listing 5-10. Creating the DimDates Table

```
-- We should create a date dimension table in the database
CREATE TABLE dbo.DimDates (
[DateKey] int NOT NULL PRIMARY KEY IDENTITY
, [Date] datetime NOT NULL
, [DateName] nVarchar(50)
, [Month] int NOT NULL
, [MonthName] nVarchar(50) NOT NULL
, [Quarter] int NOT NULL
, [QuarterName] nVarchar(50) NOT NULL
, [Year] int NOT NULL
, [YearName] nVarchar(50) NOT NULL
)
```

Once the table is created, fill it with dimensional values. You can do so during the ETL process or immediately after creating the table. This is different from the other dimensional tables, because the data for a data dimension table is not imported from the OTLP database and instead is programmatically generated.

The simplest way to accomplish this is to create a SQL `WHILE` loop and specify the range of dates that are to be placed inside the table. Listing 5-11 shows an example of a transact SQL statement that accomplishes this goal.

Once the data dimension table is filled with data, it can be referenced from both fact and dimensional tables. Therefore, you create foreign key constraints to all the tables that reference this new table.

In the DWPubsSales example, we must specifically use dates that were appropriate to Microsoft's Pubs database. These dates included sales records from the 1990s. If the date table was going to be utilized by multiple data marts or data warehouses, you would include a much broader range of dates.

Listing 5-11. Filling the DimDates Table

```
-- Because the date table has no associated source table we can fill the data
-- using a SQL script.

-- Create variables to hold the start and end date
DECLARE @StartDate datetime = '01/01/1990'
DECLARE @EndDate datetime = '01/01/1995'

-- Use a while loop to add dates to the table
DECLARE @DateInProcess datetime
SET @DateInProcess = @StartDate

WHILE @DateInProcess <= @EndDate
 BEGIN
 -- Add a row into the date dimension table for this date
 INSERT INTO DimDates (
   [Date]
 , [DateName]
 , [Month]
```

```
 , [MonthName]
 , [Quarter]
 , [QuarterName]
 , [Year]
, [YearName]
)
VALUES (
 -- [Date]
   @DateInProcess
 -- [DateName]
, Convert(varchar(50), @DateInProcess, 110)+', '
  + DateName( weekday, @DateInProcess )
 -- [Month]
, Month( @DateInProcess )
-- [MonthName]
, Cast( Year(@DateInProcess) as nVarchar(4) )+' - '
  + DateName( month, @DateInProcess )
 -- [Quarter]
, DateName( quarter, @DateInProcess )
  -- [QuarterName]
, Cast( Year(@DateInProcess) as nVarchar(4) )+' - '
  + 'Q'+DateName( quarter, @DateInProcess )
 -- [Year]
 , Year(@DateInProcess)
 -- [YearName]
 , Cast( Year(@DateInProcess) as nVarchar(4) )
)

 -- Add a day and loop again
 SET @DateInProcess=DateAdd(d, 1, @DateInProcess)
 END

-- Check the table SELECT Top 10 * FROM DimDates
```

In the next exercise, you create this date table using the code in Listing 5-10. Then, you create the foreign key constraints from the DimTitles and FactSales tables.

EXERCISE 5-3. CREATE A DATE DIMENSION

In this exercise, you create a date dimension table in the Publication Industries data warehouse. You can choose to use either the SQL code presented here, the table designer or the diagramming tools to accomplish your goal. (In Chapter 7 you fill the table with data using the code in Listing 5-11.)

Tip: The code files for this exercise, as well as all of the exercises throughout this book, are available in the downloadable book content.

Create the DimDates Table

You first task is to create a table to hold date dimension data.

1. If it is not open already, open SQL Server Management Studio; see Exercise 5-1 for more details.

2. Decide on a method for creating the DimDates table, and begin creating them utilizing the design represented in both Listing 5-10 and Figure 5-30.

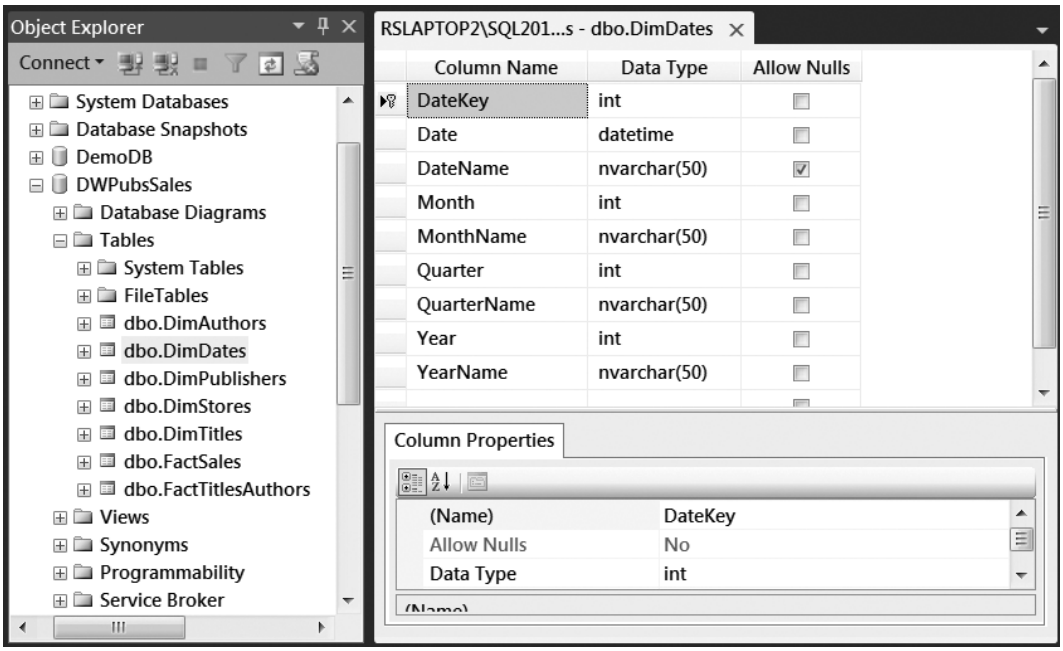

Figure 5-30. *The DimDates table*

3. If you are typing the SQL code, highlight and execute the code by clicking the "! Execute" button. If you choose to use the table designer or the diagramming tool, make sure you use the Save button to create the tables in the database.

Adding Foreign Key Constraints

We have added a new table to the data warehouse database, but now we want to connect this new table to the existing ones using foreign key constraints.

1. Add the foreign key constraints to the DimDates table using the code in Listing 5-12.

Listing 5-12. Creating the DWPubsSales Foreign Key Constraints

```
USE [DWPubsSales]
GO

ALTER TABLE [dbo].[FactSales] WITH CHECK ADD CONSTRAINT [FK_FactSales_DimDates]
FOREIGN KEY([OrderDateKey])
REFERENCES [dbo].[DimDates] ([DateKey])
GO

ALTER TABLE [dbo].[DimTitles] WITH CHECK ADD CONSTRAINT [FK_DimTitles_DimDates]
```

```
FOREIGN KEY([PublishedDateKey])
REFERENCES [dbo].[DimDates] ([DateKey])
GO
```

In this exercise, you created a date dimension table in the data warehouse and added foreign key constraints to the tables that reference it. We will fill the table with data during the ETL process, so for now we leave it empty.

Getting Organized

Excellent, we now have the database and all the tables, and we can get started with the ETL process. Before we do, however, we need to organize our hard work by performing a few simple tasks.

Backing Up the Data Warehouse

We have already talked about how it is sometimes necessary to back up a data warehouse database. Well, this is one of those times! The reason to do so now is to have a copy of the data warehouse in its empty state so that you can hand it over to your testers and other developers for review. These team members can easily restore the backup and do their work simultaneously with yours. Also, a database backup is a simple way to save your progress thus far.

The backup and restore process is easy. Listing 5-13 exhibits code that does both.

Listing 5-13. Backing Up and Restoring a Database

```
BACKUP DATABASE [DWPubsSales]
TO DISK =
N'C:\_BISolutions\PublicationsIndustries\DWPubsSales\DWPubsSales_BeforeETL.bak'
GO

RESTORE DATABASE [DWPubsSales]
FROM DISK =
N'C:\_BISolutions\PublicationsIndustries\DWPubsSales\DWPubsSales_BeforeETL.bak'
WITH REPLACE
Go
```

■ **Important** The SQL backup statement will not create folders if they do not exist. Consequently, before you execute the backup command, it is important to create any folders indicated in the backup path. If you try running the code in Listing 5-13 without the folder, you can expect an error.

Scripting the Database

Another way that you can preserve your work is by using the database scripting tool. SQL Server has long provided a tool for scripting the objects in the database, and SQL 2012 is no exception. You can launch the scripting tool by right-clicking the database in Object Explorer and selecting Tasks ➤ Generate Scripts from the context menu. The selection launches the Generate and Publish Scripts Wizard you see in Figure 5-31.

■ **Note** You may notice that there is also a Script Database menu item, but this option only generates code to create the database and not all the tables within it.

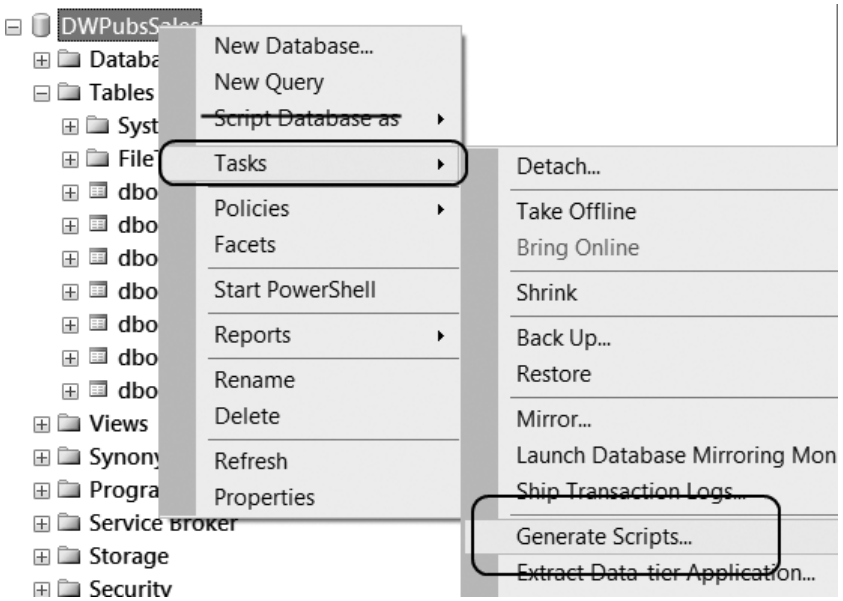

Figure 5-31. *Generating SQL scripts for your database and objects*

When the wizard appears, walking through the steps enables the generation of SQL code that can be used to quickly re-create your data warehouse. The first page of the wizard displays a list of the wizard pages on the left of its window (Figure 5-32).

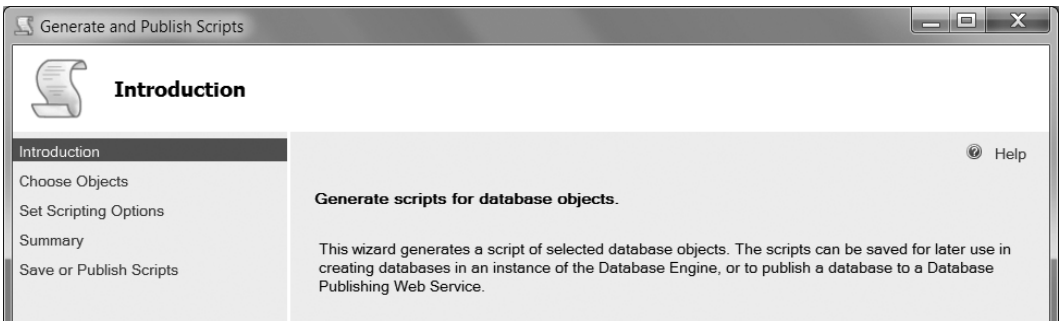

Figure 5-32. *The first page of the Generate and Publish Scripts Wizard*

The second page of the wizard allows you to choose the items you script (Figure 5-33). For our purposes, we want all of the database objects, which is the default choice.

Figure 5-33. *Selecting which database objects to script*

The third page allows you to choose where your script is saved. The two most common choices are to save to a file, which is the default, and to save to a New Query window (Figure 5-34). This second option is convenient because you will most likely want to review the code and make changes before you save it as one of your BI solution files.

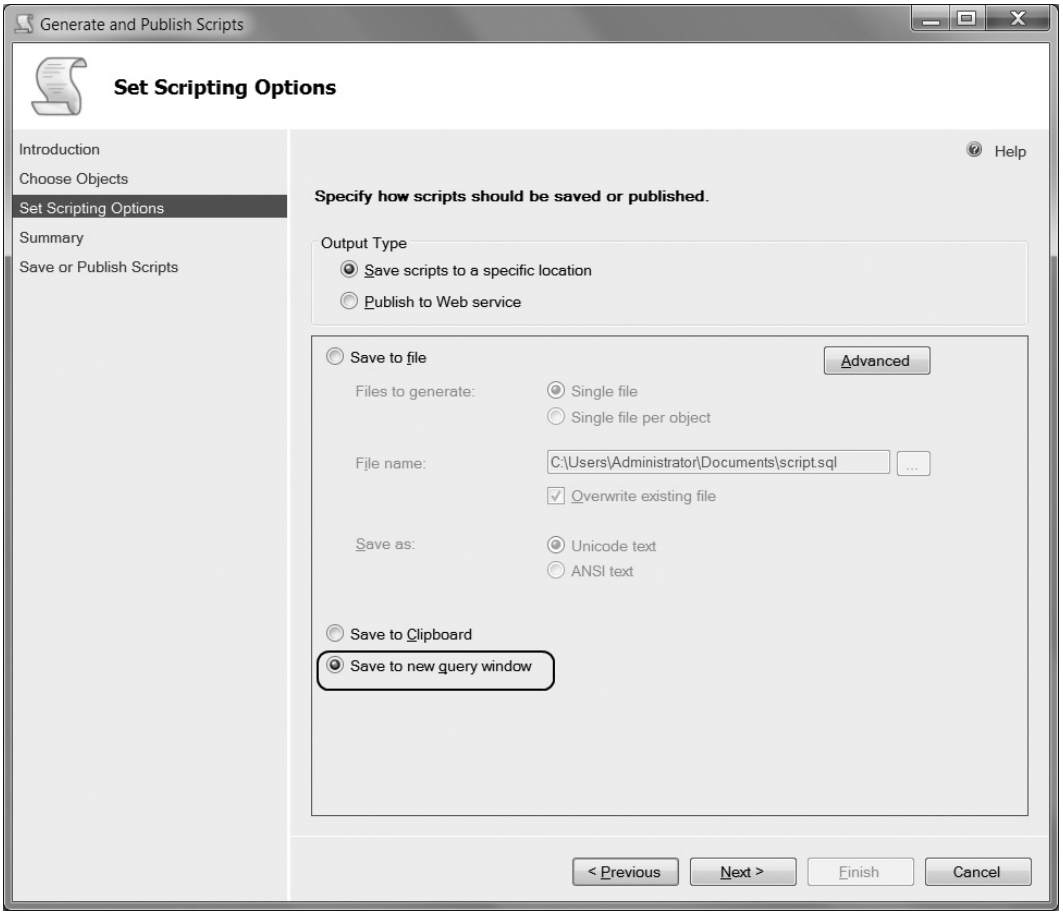

Figure 5-34. *Selecting the scripting options*

The next page provides you with a summary of your choices (Figure 5-35).

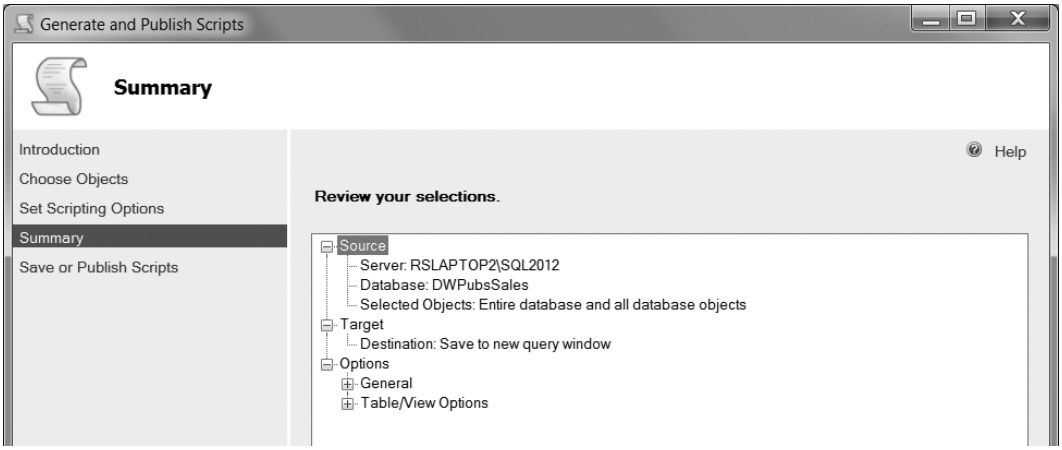

Figure 5-35. *Reviewing the Generate and Publish Scripts summary page*

When you navigate to the last page, it starts the scripting process and indicates the successful completion or failure of each item scripted (Figure 5-36).

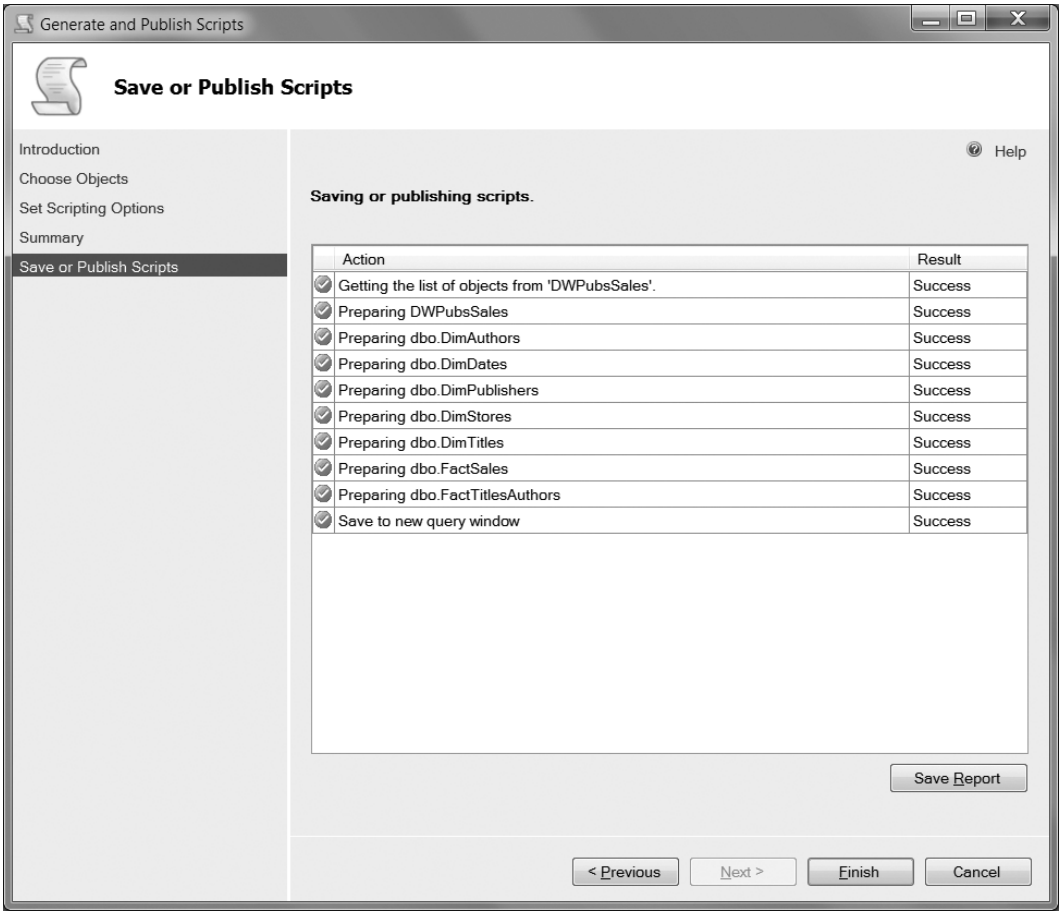

Figure 5-36. *Saving and publishing the SQL Scripts*

Having a script file in addition to a database backup is practical for a couple of reasons. It is useful for training new team members, familiar with SQL code, about the database design. Another advantage is that you can include the script along with your business intelligence projects in a Visual Studio solution. Although this is also true of a backup file, you cannot open a backup directly in a Visual Studio solution. With a script file, you can simply double-click the file in Solution Explorer, and its contents conveniently display within Visual Studio.

■ **Tip** The scripting tool generates working code but often adds more settings and features than you need in your script. We use this tool to create the basic outline of the script and then modify it to remove any superfluous code and add our own comments.

Organizing Your Files with Visual Studio

As you saw in Chapter 2, using Visual Studio to manage your BI solution files is practical because most of the BI servers already store their files in Visual Studio projects. The exception to this rule is the SQL Server database engine. This can be done in Visual Studio, but the means of accomplishing this can be quite obscure, especially

because SQL Server Management Studio has its own type of solutions for code files. Sadly, SQL Management Studio and Visual Studio solutions are not aligned. This means you cannot take a Visual Studio project and open it in SQL Server Management Studio, and vice versa. To work around this dilemma, you can use a solution folder in Visual Studio and add your SQL files to it.

■ **Note** Visual Studio 2010 finally has a true SQL Server Database project type when you install the SQL Server Data Tools (SSDT) plug-in. We use SQL Server Management Studio instead, because it is simpler to use and applicable to all versions of SQL Server. We have included information about SSDT at `http://NorthwestTech.org/SSDTDemos`.

One nice feature of Visual Studio is the ability to add logical folders to a solution. This helps you organize files that are not part of a standard Visual Studio project, such as SSIS or SSAS, but are still part of your overall BI solution.

In Chapter 3, Exercise 3-4, you created a blank Visual Studio Solution and then added a solution folder to it called `SolutionDocuments`. You then placed the planning documents you created in Chapter 3 into that folder. This organized your planning documents within Visual Studio. Your SQL scripts and backup files can also be added in a Visual Studio solution in a similar manner (Figure 5-37). Let's see how this is done in the following exercise.

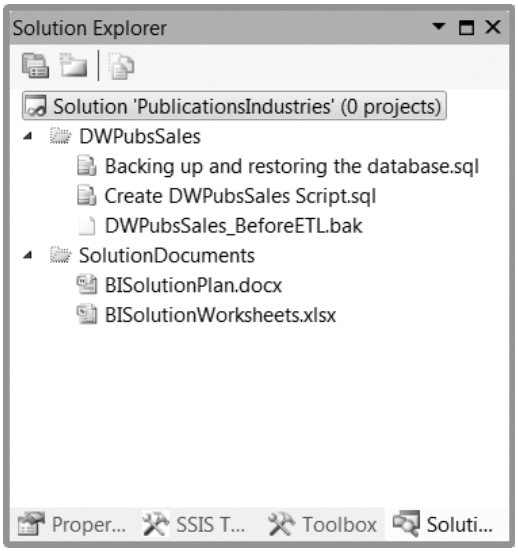

Figure 5-37. *Organizing files with solution folders*

EXERCISE 5-4. ADDING DATABASE FILES TO VISUAL STUDIO

In this exercise, you add scripts and backup files to the Visual Studio Solution you created in Chapter 3. You start by creating the files and a subfolder to hold the files. Then add these files to a logical Visual Studio folder, as shown in Figure 5-37.

Important: You are practicing administrator-level tasks in this book, so you need administrator-level privileges. The easiest way to achieve this is to remember to always right-click a menu item, select Run as Administrator, and then answer Yes to access administrator-level privileges while running this program. In Windows 7 and Vista, logging in with an administrator account is not enough. For more information, search the Web on the keywords "Windows 7 True Administrator and User Access Control."

Create a Database Backup and Restore Script

Your first task is to create an operating system folder on your hard drive to hold your scripts and backup files.

1. Create a subfolder to hold the script. To do this, use Windows Explorer and navigate to `C:\_BISolutions\PublicationsIndustries`; then right-click the PublicationsIndustries folder and select the New ➤ Folder option from the context menu. The new folder is created as a subfolder of the PublicationsIndustries folder.
2. Rename the new folder **DWPubsSales**. To do this, right-click the new folder and select Rename from the context menu. Type in the name **DWPubsSales** when prompted (Figure 5-38).

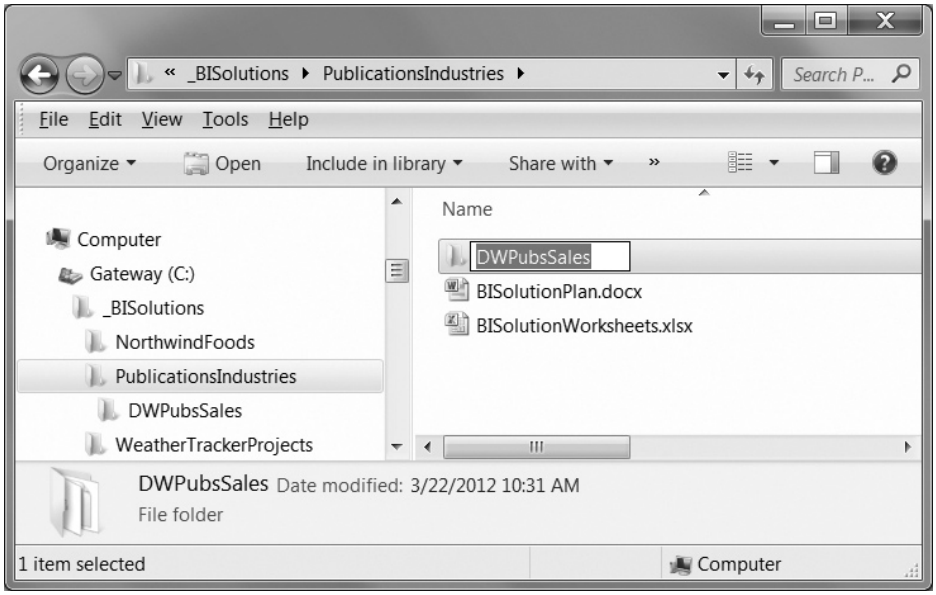

Figure 5-38. *The current files and subfolders in the* `_BISolutions` *folder*

Create a Database Backup and Restore Script

Now we need to place files within the subfolder. Let's start by making a backup and restore script and then use it to create a SQL backup file.

1. If it is not open already, open SQL Server Management Studio; see Exercise 5-1 for more details.
2. Using the code in Listing 5-14 to create the backup and restore script in a new query window.

Tip: If you want to use our script file, use the File ➤ Open ➤ File menu in SQL Server Management Studio to open the file `C:\_BookFiles\Chapter05Files\Listing 5-14. Backup and Restore DWPubsSales.sql`"

Listing 5-14. Backing Up and Restoring the DWPubsSales Database

```
/***********************************************
1) Make a copy of the empty database
before starting the ETL process
***********************************************/
BACKUP DATABASE [DWPubsSales]
TO DISK =
N'C:\_BISolutions\PublicationsIndustries\DWPubsSales\DWPubsSales_BeforeETL.bak'
GO

/***********************************************
2) Send the file to other team members
and tell them they can restore the database
with this code...
***********************************************/

-- Check to see if they already have a database with that name…
IF EXISTS (SELECT name FROM sys.databases WHERE name=N'DWPubsSales')
  BEGIN
  -- If they do, they need to close connections to the DWPubsSales database, with this code!
    ALTER DATABASE [DWPubsSales] SET SINGLE_USER WITH ROLLBACK IMMEDIATE
  END

-- Now they can restore the empty database...
USE Master
RESTORE DATABASE [DWPubsSales]
FROM DISK =
N'C:\_BISolutions\PublicationsIndustries\DWPubsSales\DWPubsSales_BeforeETL.bak'
WITH REPLACE
GO
```

3. Save the script file into the new `DWPubsSales` folder as the `Backing up and restoring database.sql` file (Figure 5-39). To do this, use SQL Server Management Studio and click the File ➤ Save ➤ current file name ➤ As . . . menu item.

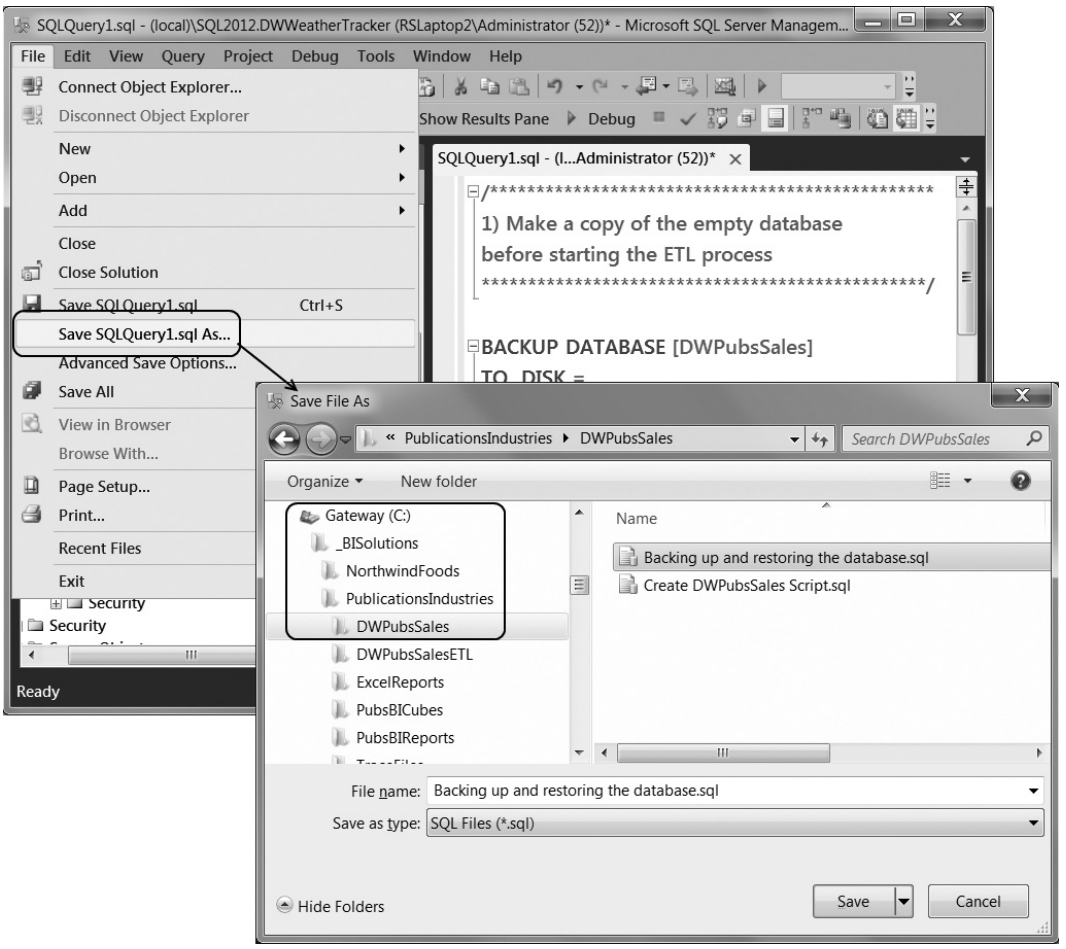

Figure 5-39. *Saving your SQL script files to the new* `DWPubsSales` *folder*

Create a Database Backup

Now that you have the backup and restore script, you need to use it to back up your database.

1. Execute the `BACKUP DATABASE` statement to create a database backup file in the `DWPubsSales` folder. To do this, highlight the `BACKUP DATABASE` statement in the Query Window, as shown in Figure 5-40, and click the "! Execute" button.

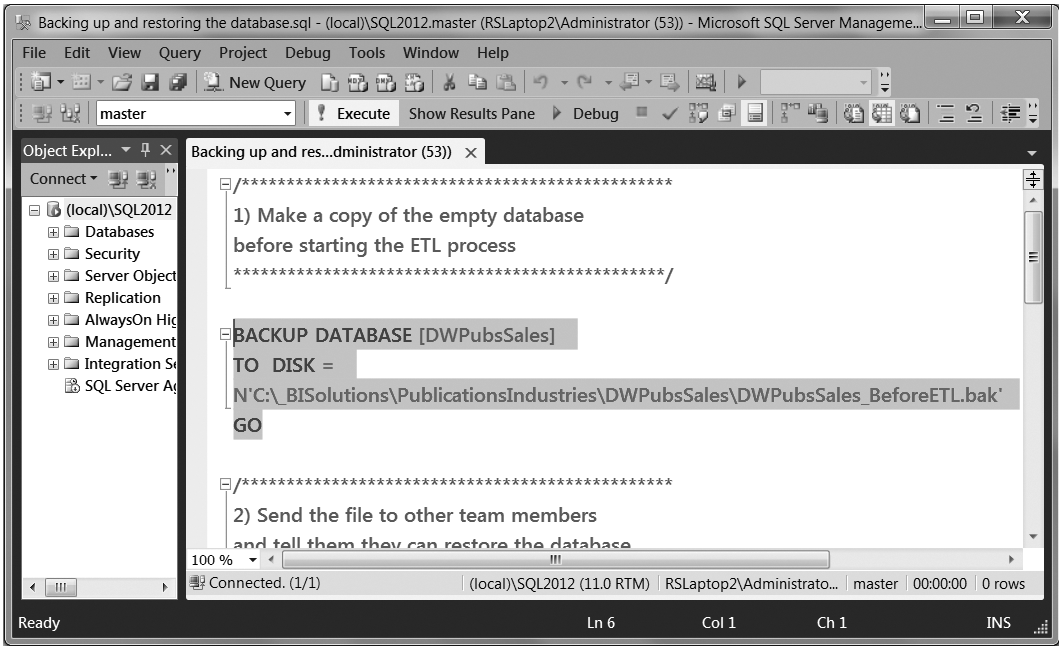

Figure 5-40. *Executing the BACKUP DATABASE statement*

2. Test the restore process by restoring the database. You can do this by highlighting all the code under the `IF` statement, the `USE` statement and the `RESTORE DATABASE` statement, as shown in Figure 5-41.

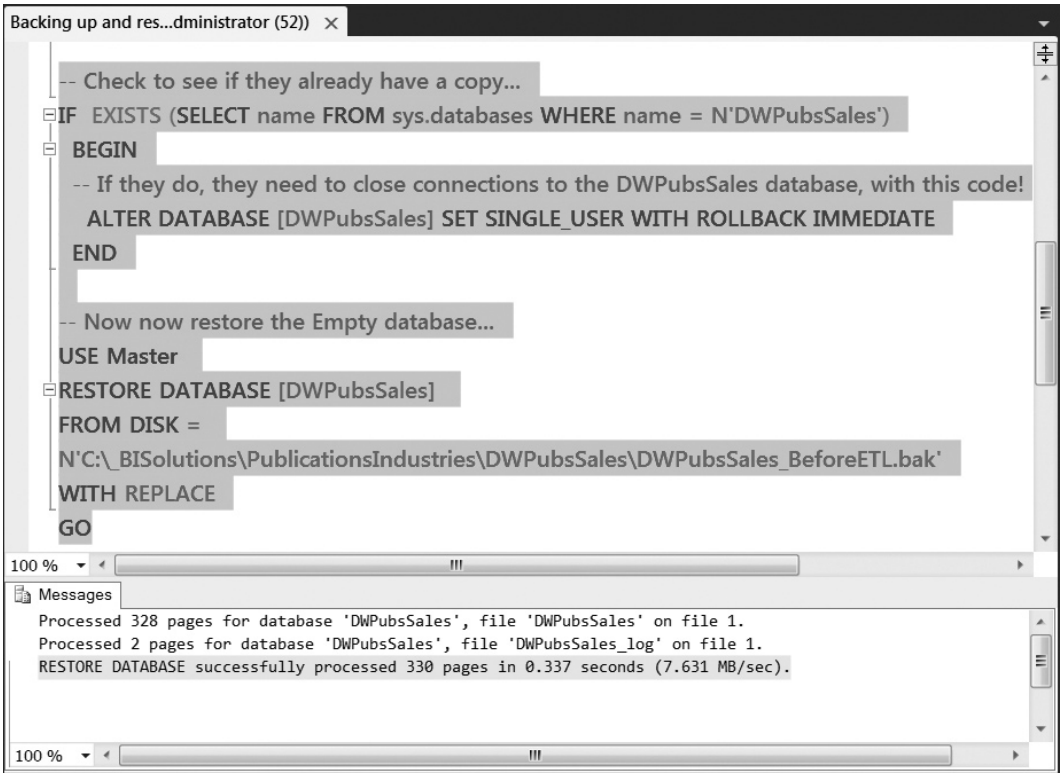

Figure 5-41. *Executing the RESTORE DATABASE statement*

Create a Database Script File

You now have a tested backup and restore script as well as a database backup file in the `DWPubsSales` folder. Let's add a complete script for creating the database. We have created a custom script for this process, so you just need to review the code and add it to the folder.

1. Open the author's version of the script file and review its contents. To do this, use the File ➤ Open ➤ File menu in SQL Server Management Studio and open the file `C:\_BookFiles\Chapter05Files\Create DWPubsSales Script.sql`.
2. Review the contents of the file and then save it to the `DWPubsSales` folder. To do so use SQL Server Management Studio and click the File ➤ Save ➤ current file name ➤ As . . . menu item.

Add the Files to Visual Studio

We now have three files in the operating system folder, the backup and restore script, the backup file and the database creation script. Let's add them to the Publications Industries Visual Studio solution you created in Chapter 3.

1. Open Visual Studio 2010. (You can do so by clicking the Start button, navigating to All Programs ➤ Microsoft Visual Studio 2010, and right-click Microsoft Visual Studio 2010 to see an additional context menu [Figure 2-7]. In this new menu, click the Run as Administrator menu item. If the UAC message box appears asking "Do you want the following program to make changes to this computer?" click Yes [or Continue depending upon your operating system] to accept this request.)

To add these files to the Visual Studio solution you made for Publication Industries in Chapter 3, follow these steps:

2. When Visual Studio opens, open the solution you created in Chapter 3. You can do so by using the File ➤ Open-Project/Solution menu item. A dialog window will open allowing you to select your solution file.
3. Navigate to the SLN file `C:\_BISolutions\PublicationsIndustries\PublicationsIndustries.sln`, select it, and then click the Open button at the bottom of the dialog window.
4. Right-click the solution icon in the Solution Explorer window and select Add ➤ New Solution Folder from the context menu (Figure 5-42).

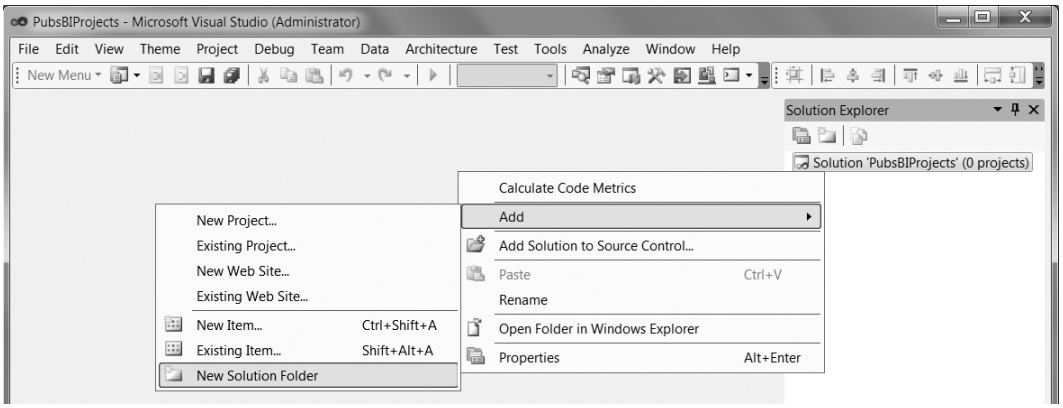

Figure 5-42. *Adding a new solution folder using Solution Explorer*

5. Rename the new solution folder (we called ours DWPubsSales in Figure 5-43).

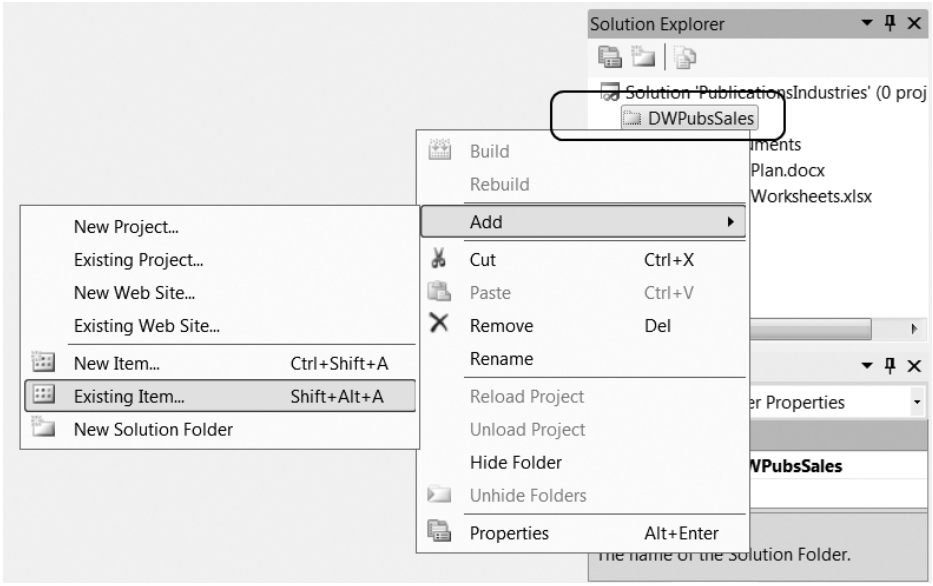

Figure 5-43. *Adding existing files to the new solution folder*

6. Right-click the new folder and select Add ➤ Existing Item from the context menu.
7. Select the files you want to add (in our example those are the backup and script files).

When this is completed, the files are displayed in Solution Explorer alongside the solution documents (Figure 5-44). Visual Studio tries to open the files added in this manner. If it cannot interpret the type of file it is, such as the backup file, it shows a hex representation of the file. You can close any windows, tabs, or applications that appear.

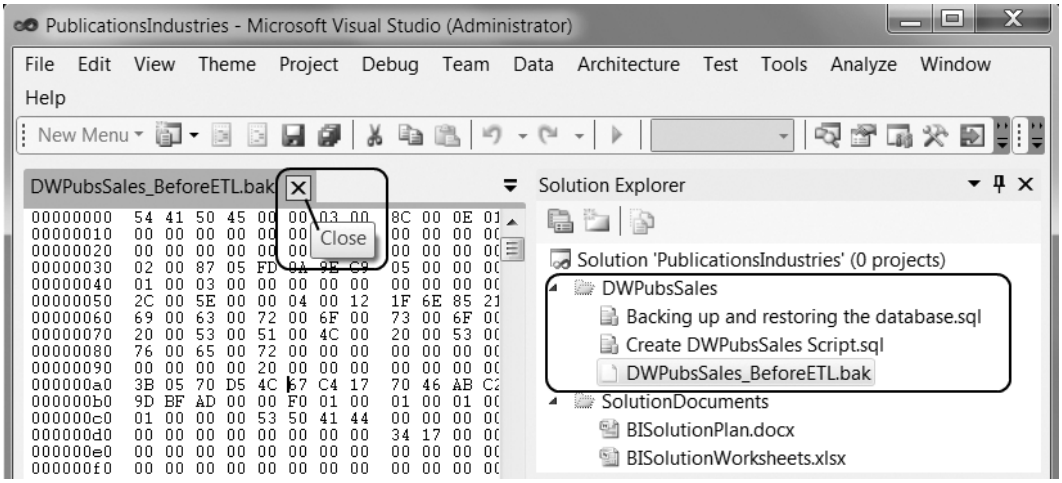

Figure 5-44. *Adding folders and files to the solution folders*

In this exercise, you opened a preexisting Visual Studio solution and added various solution files. Currently there are no projects within the solution, but we change that shortly by adding an SSIS project to it in Chapter 6.

Moving On

In this chapter, you saw how to implement the creation of a data warehouse by creating a database and tables using SQL Server 2012. We examined various options that you can use to create your database, as well as three possible ways to create the tables: using SQL code, the table designer, and the database diagramming tool.

Thus far, we have covered three steps of our eight-step outline. In Chapter 3, we walked through the interview process. In Chapter 4, we looked at data warehouse designs and planned the solution. And in this chapter, we have completed the process of creating the data warehouse. As you can see in Figure 5-45, we are now ready to move on to step 4, the ETL process.

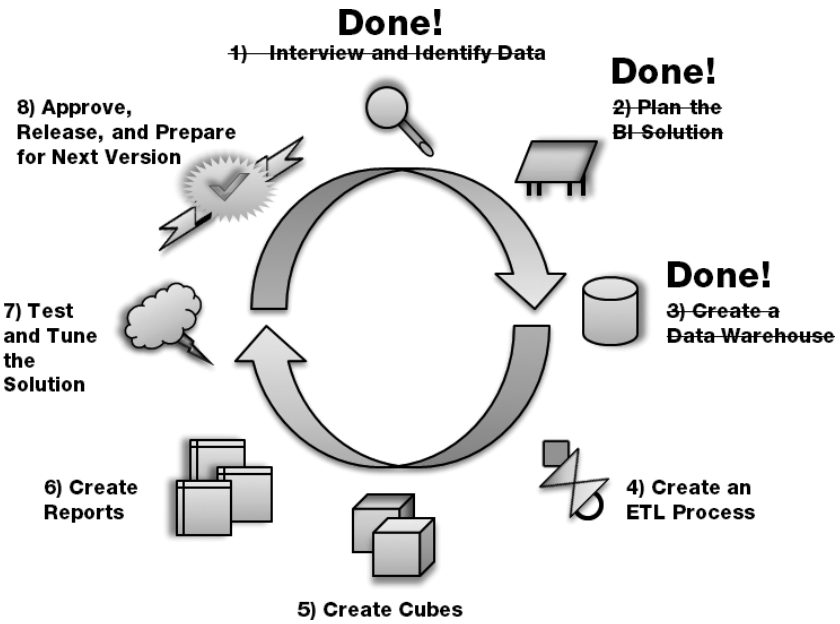

Figure 5-45. *Progressing through the BI solution steps*

In Chapter 6, we examine commonly used SQL code for the ETL process. Then, we follow this up in Chapters 7 and 8 by showing how this code is used in SQL Server integration Services to create your ETL process. Until then, we recommend you practice creating a data warehouse based on the Northwind database in this "Learn by Doing" exercise.

LEARN BY DOING

In this "Learn by Doing" exercise, you perform the process defined in this chapter using the Northwind database. We have included an outline of the steps you performed in this chapter and an example of how the authors handled them in two Word documents. These documents are found in the folder `C:\_BISolutionsBookFiles\_LearnByDoing\Chapter05Files`. Please see the `ReadMe.doc` file for detailed instructions.

What's Next?

Using the SQL Server database engine effectively is a complex task. We have included only a minimum of what you need to be effective when creating a BI solution. If you want a deeper understanding, you will find that many books have been written about the subject; some of them are excellent for database administrators, whereas others are more for general knowledge.

Most BI developers are not the actual database administrators and do not necessarily need to have a great degree of knowledge about the database engine itself. Still, most BI developers can benefit from having a more complete understanding of this subject. For this reason, we recommend the following beginning book on SQL administration: *Beginning SQL Server 2008 Administration* by Grant Fritchey and Robert Walters (Apress, 2009).