data access. Instances of denormalization can be found in production systems where the join to a table is slowing down queries, or perhaps where normalization is not required (for example, when working with a system in which the data are not regularly updated).

Just because others say your data should be totally normalized, it is not necessarily true, so don't feel forced down that route. The drawback of denormalizing your data too far, though, is that you'll be holding duplicate and unnecessary information that could be normalized out to another table and then just joined during a query. This will, therefore, create performance issues as well as use a larger amount of data storage space. However, the costs of denormalization can be justified if queries run faster. That said, data integrity is paramount in a system. It's no use having denormalized data in which there are duplications of data where one area is updated when there's a change, and the other area isn't updated.

Denormalization is not the route you want to take in the database example; now that you have all the data to produce the system, it's time to look at how these tables will link together.

Creating the Sample Database

Let's now begin to create your example database. In this section, you will examine two different ways to create a database in SQL Server:

- Using the SQL Server Management Studio graphical interface
- Using T-SQL code

Both methods have their own merits and pitfalls for creating databases, as you'll discover, but these two methods are used whenever possible throughout the book, and where you might find one method is good for one task, it may not be ideal for another. Neither method is right or wrong for every task, and your decision of which to use basically comes down to personal preference and what you're trying to achieve at the time. You may find that using T-SQL code for building objects provides the best results, because you will see instantly the different possible selections. Using T-SQL also provides you with a greater base from which to work with other programming languages and databases, although some databases do differ considerably in the language used to work with them. However, if the syntax for the commands is not familiar to you, you may well choose to use a wizard or SQL Server Management Studio. Once you become more comfortable with the syntax, then a Query Editor pane should become your favored method.

You'll also examine how to drop a database in SQL Server Management Studio.

Creating a Database in SQL Server Management Studio

The first method of creating a database you will look at is using SQL Server Management Studio, which was introduced in Chapter 2.

TRY IT OUT: CREATING A DATABASE IN SQL SERVER MANAGEMENT STUDIO

1. Before creating the database, you'll need to start up SQL Server Management Studio. To do this, select Start ➤ All Programs ➤ Microsoft SQL Server 2012 ➤ SQL Server Management Studio.

■ **Tip** I run all this book's examples from a server called `FAT-BELLY-Sony` using the named instance `APRESS_DEV1`. You will have your own name for your host computer and possibly even your own name for the installed instance. Therefore, throughout the book, when you see `FAT-BELLY-Sony`, you should substitute your own host computer name.

2. Ensure that you have registered and connected to your server. If the SQL Server service was not previously started, it will automatically start as you connect, which may take a few moments. However, if you have not shut down your computer since the install of SQL Server, then everything should be up and running. SQL Server will stop only if you have shut down your computer and indicated not to start the SQL Server service automatically or the Windows account used to start the service has a problem, such as being locked out. To start SQL Server, or conversely, if you want to set up SQL Server not to start automatically when Windows starts, set this either from Control Panel or from the SQL Server Configuration Manager found under Start ➤ All Programs ➤ Microsoft SQL Server 2012 ➤ Configuration Tools.
3. In Object Explorer, expand the Databases node until you see either just the system database and database snapshot nodes that always exist. Ensure that the `Databases` folder is highlighted and ready for the next action, as shown in Figure 3-7.

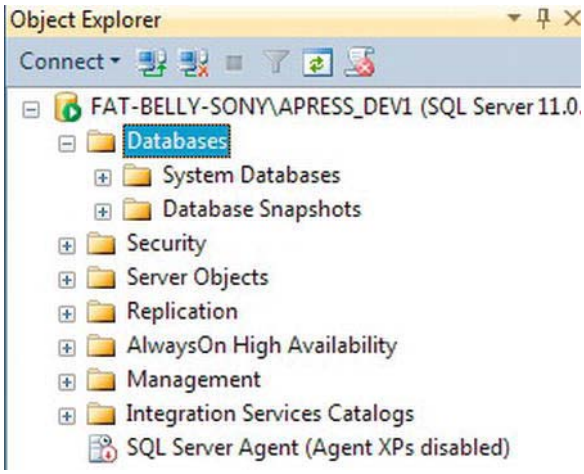

Figure 3-7. *The `Databases` node in Object Explorer*

A minimum amount of information is required to create a database:

- The name the database will be given
- How the data will be sorted
- The size of the database
- Where the database will be located
- The name of the files used to store the information contained within the database

SQL Server Management Studio gathers this information using the New Database menu option.

4. Right-click the `Databases` node to bring up a pop-up menu with a number of different options. Select New Database, as shown in Figure 3-8.

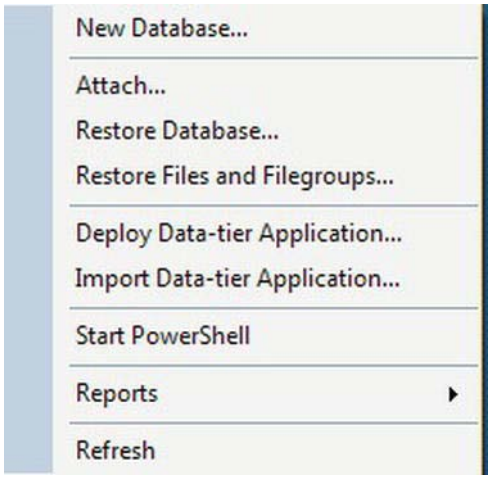

Figure 3-8. *Selecting to create a new database*

5. You are now presented with the New Database dialog box set to the General tab. First enter the name of the database you want to create—in this case, `ApressFinancial`. Notice as you type that the two file names in the Database Files list box also populate. This is simply an aid, and you can change the names (see Figure 3-9). However, you should have a very good reason to not accept the names that the screen is creating, as this is enforcing a standard.

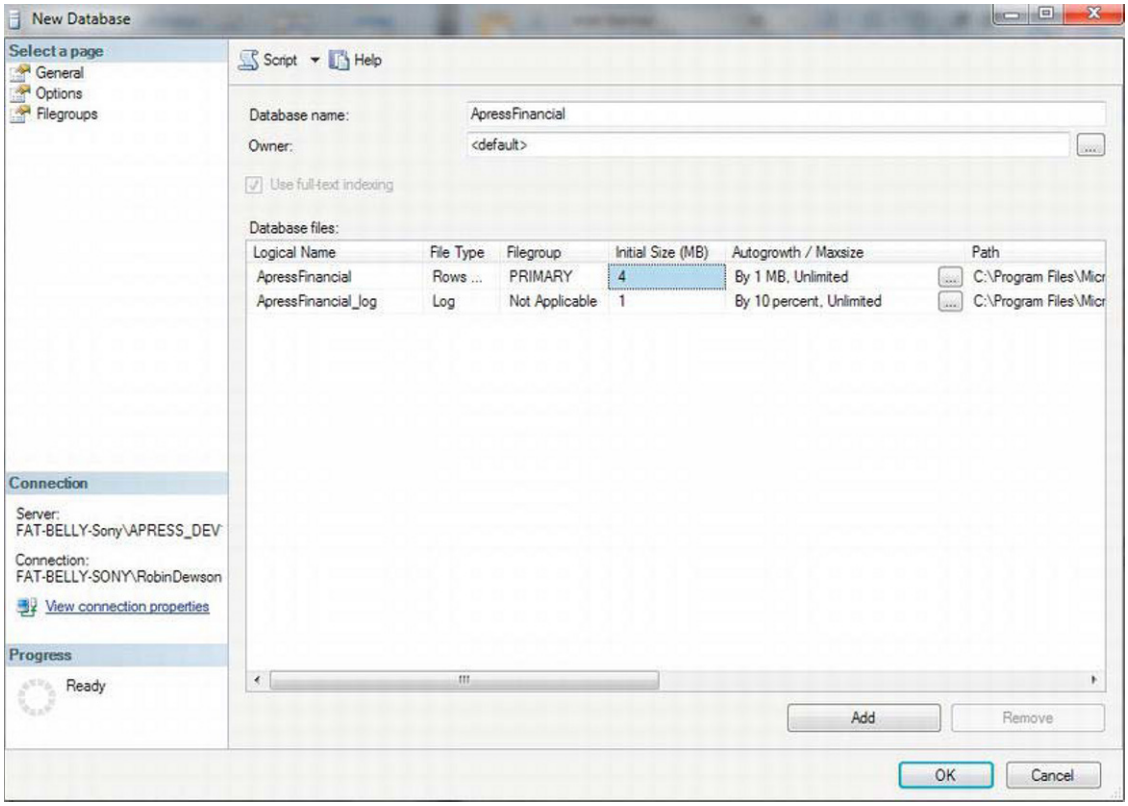

Figure 3-9. *General settings in the New Database dialog*

The General screen within this page collects the first two pieces of information. The first piece of information required is the database name. No checks are done at this point as to whether the database exists (this comes when you click OK); however, there is some validation in the field so that certain illegal characters will not be allowed.

■ **Note** Illegal characters for a database name are as follows:
" ' */?:\<>

Keep to alphabetic, numeric, underscore, or dash characters. Also, you may want to keep the database name short, as the database name has to be entered manually in many parts of SQL Server. Below the database name is the owner of the database—in other words, the login defined as having the ability to do anything within the database as well as determine what other logins can do with the database. This can be any login that has the authority to create databases. A server in many—but not all—installations can hold

databases that belong to different development groups. Each group would have a login that was the database owner, and at this point, you would assign the specific owner. For the moment, let it default to the `<default>` account, which will be the account currently logged in to SQL Server; you'll learn how to change this later. If you're using Windows authentication, then your Windows account will be your user ID, and if you're using SQL Server authentication, it will be the login ID you used at connection time.

The database owner, which is often shortened to dbo, initially has full administration rights on the database, from creating the database, to modifying it or its contents, to even deleting the database. Because a dbo account has such power it is best to create either a user or a group specifically for administration rights. You will see groups discussed in Chapter 4.

Ignore the check box for Full-Text Indexing. You would select this option if you wanted your database to have columns that you could search for a particular word or phrase and Full-Text Indexing had been installed. For example, search engines could have a column that holds a set of phrases from web pages, and full-text searching could be used to find which web pages contain the words being searched for.

The File Name entry (off-screen to the right in Figure 3-9) is the name of the physical file that will hold the data within the database you're working with. By default, SQL Server takes the name of the database and adds a suffix of `_Data` to create this name.

Just to the left of the File Name option is the physical path where the files will reside. The files will typically reside in a directory on a local drive. Some larger companies attach network storage to the server, and the data will reside there, off the main computer, and in this case you would enter the network path. For an installation such as you are completing on a local machine, the path will normally be the path specified by default. That path is to a subfolder under the SQL Server installation directory. If, however, you are working on a server, although you will be placing the files on a local hard drive, the path may be different if the server has disk partitions or even a second hard drive specifically for the data, so that different teams' installations will be in different physical locations or even on different local hard drives.

The database files are stored on your hard drive with an extension of `.MDF`—for example, `ApressFinancial_Data.MDF`. In this case, `.MDF` is not something used by DIY enthusiasts, but it actually stands for Master Data File and is the name of the *primary data file*. Every database *must* have at least one primary data file. This file may hold not only the data for the database, but also the location of all the other files that make up the database, as well as startup information for the database catalog. An MDF is an allocation of space on the hard drive where SQL Server places the tables as well as the rows of information that are within the tables.

It is also possible to have *secondary data files*. These would have the suffix `.NDF`. Again, you could use whatever name you wished, and, in fact, you could have an entirely different name from the primary data file. However, if you did so, the confusion that would abound is not worth thinking about. So do use the same name, and if you need a third, fourth, and so on, then add on a numerical suffix.

Secondary data files allow you to spread your tables and indexes over two or more disks. The upside is that by spreading the data over several disks, you will get better performance. In a production environment, you may have several secondary data files to spread out your heavily used tables.

■ **Note** As the primary data file holds the database catalog information that will be accessed constantly during the operation of the server, it would be best, in a production environment at least, to place all your tables on a secondary data file.

Finally, there will be another file type, which is `.LDF`. This is the transaction log file that holds a record of certain actions you perform. This is covered in detail in Chapter 8 when we take a look at backup and restores. You would place the file name for a secondary data file in the row below the `ApressFinancial_Data` entry in the Data Files list box, after clicking the Add button. The `File Type` column shows whether the file is a data file or a log file, as in a file for holding the data or a file for holding a record of the actions done to the data.

The next column in the grid is titled `Filegroup`. This allows you to specify the `PRIMARY` filegroup and any `SECONDARY` data filegroups for your database. Filegroups are for grouping logical data files together to manage them as a logical unit. You could place some of your tables in one filegroup, more tables in another filegroup, indexes in another, and so on. Dividing your tables and indexes into filegroups allows SQL Server to perform parallel disk operations and tasks if the filegroups are on different drives. You could also place tables that are not allowed to be modified together in one filegroup and set the filegroup to Read-Only. If you want to have another database file, you can click Add at the bottom of the General page. Figure 3-10 shows the dialog for creating a new filegroup, which is displayed if you click the combobox in the Filegroup column on any newly created database file. The top option allows you to create read-only filegroups. Finally, when creating a database object, the default filegroup is the `PRIMARY` filegroup. In a production environment—and therefore in a development environment as well, so that it is simpler to move from development through to production—you would create a secondary filegroup and set it as the default. In this book, you will just keep to the `PRIMARY` filegroup for simplicity. At this point, a third database file is not required.

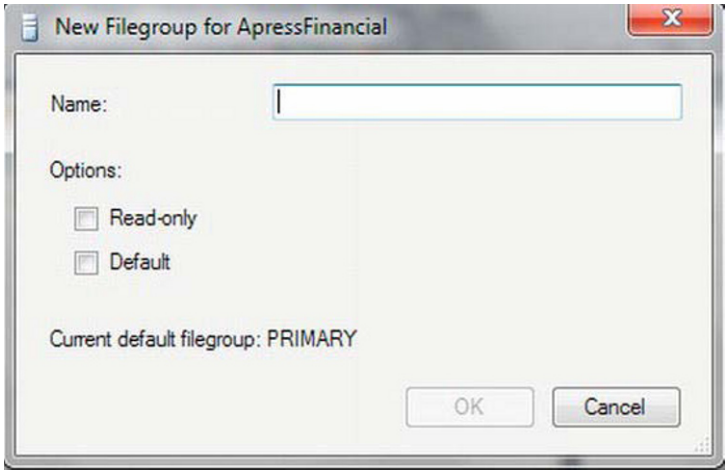

Figure 3-10. *New filegroup*

The logic behind secondary data files is relatively straightforward. A `PRIMARY` filegroup will always—and must always—contain the system tables that hold the information about the database, the tables, the columns, and so on. If you have the Autogrowth file option (covered shortly) switched off, the `PRIMARY` filegroup is likely to run out of space at some point. If this happens and no secondary data files are specified, the database will grind to a halt until some space is added. However, in most instances, especially when you're first starting out, you can leave the database with only a `PRIMARY` filegroup.

Don't misunderstand filegroups and space, though: filegroups are there to help you organize your files within your database storage, and the files that make up the filegroup may span several disks for a performance issue. You will move files around filegroups for speed, efficiency, security, backups, and a number of other reasons. However, you can still hold all the files in one filegroup—the `PRIMARY` filegroup— which is what you'll do throughout this book.

6. Click Cancel in the New Filegroup dialog box, and then click Remove in the New Database dialog box to remove the third line within the database files grid if you added a line while checking out the filegroup discussion for Figure 3-10.

■ **Note** Remember that the `PRIMARY` filegroup may hold not only data, but also the system tables, so the `PRIMARY` filegroup could fill up purely with information about tables, columns, and so forth used to run SQL Server.

The next item in the New Database dialog box is the Initial Size (MB) column. The initial size of the database is its size when empty. Don't forget that the database won't be totally empty and is created from the `model` system database, and some of the space will be initially taken up with the system tables. It is impossible to say, "I'm creating a database, and the initial size must be *nn*MB"—the database size depends on many factors, such as the number of tables, how much static information is stored, to what size you expect the database to grow, and so on. It would be during the investigation phase that you would try to determine the size that you expect the database to reach over a given period of time. If anything, estimate larger rather than smaller to avoid fragmentation.

Moving on to the next, and possibly most important, area: Autogrowth. This option indicates whether SQL Server will automatically handle the situation that arises if your database reaches the initial size limit. If you don't set this option, you will have to monitor your database and expand its size manually, if and when required. Think of the overhead in having to monitor the size, let alone having to increase the size! It is much easier and less hassle, and much less of a risk, to let SQL Server handle this when starting out. However, do set a maximum size so that you have some emergency disk space available in case it is required.

■ **Note** In a production environment, or even when you're developing in the future, it will be more common to switch Autogrowth on and fix the maximum size. This prevents your hard drive from filling up and your server from being unable to continue. At least when you fix the maximum size, you can keep some hard drive space in reserve to enable your SQL Server to continue running while the development team tries to clear out unwanted data, but also create an initial smaller size and allow growth if required.

While SQL Server handles increasing the size of the database for you, it has to know by how much. This is where the Autogrowth option comes in. You can let SQL Server increase the database either by a set amount each time in megabytes or by a percentage. The default is By Percent, and at this stage, it doesn't really matter. In the example, the first increase will be 0.4MB; the second increase will be 0.44MB. For the example, this is sufficient, as there won't be a great deal of data being entered. However, the percentage option does give uneven increases, and it could mean that a larger 10% increase could be tens or hundreds of megabytes. This could be a large increase that could be unnecessary for the database. If you want to change these options by selecting the autogrowth options button (the ellipsis) to the right of the current setting, you can disable autogrowth of your database in the dialog that appears. You can also, as discussed, alter it to increase by By MB rather than By Percent, and, to repeat, this can be the better option when you come to developing your own database.

In the Autogrowth dialog, the Maximum File Size option sets a limit on how large the database is allowed to grow. The default is "unrestricted growth"—in other words, the only limit is the spare space on the hard drive. This is good, as you don't have to worry about maintaining the database too much. But what if you have a rogue piece of code entering an infinite loop of data? This scenario is rare, but not unheard of. It might take a long time to fill up the hard drive, but fill up the hard drive it will, and with a full hard drive, purging the data will prove troublesome. When it is time to start moving the database to a production environment, ensure the Restrict File Growth (MB) option is set to guard against such problems.

The final column that you will find in the New Database dialog box by scrolling to the right is Path. In this column, you define where the database files will reside on your hard drive. If SQL Server is installed on your `C` drive, no paths for the data were changed, and you are working on a default instance, then you will find that the default is `C:\Program Files\Microsoft SQL Server\MSSQL.11.MSSQLSERVER\MSSQL\Data`. Figure 3-9 shows I'm working on a mapped drive that has been given the drive letter `C`. The command button with the ellipsis (. . .) to the right of the path brings up an explorer-style dialog that allows you to change the location of the database files. For example, if you move to a larger SQL Server installation, moving the location of these files to a server relevant to the needs of your database will probably be a necessity.

The line that has a File Type setting of Log includes the same information as a Data File Type setting, with one or two minor exceptions. The File Name places a suffix of `_Log` onto the database name, and there is no ability to change the `Filegroup` column, since the Transaction Log doesn't actually hold system tables, and so would fill up only through the recording of actions. It is possible, however, to define multiple log file locations. Filling the transaction log and not being able to process any more information because the log is

full will cause your SQL Server to stop processing. Specifying more than one log location means that you can avoid this problem. The use of a failover log file in larger production systems is advisable.

Let's now move on to discuss the Options page of the New Database dialog box (see Figure 3-11).

The first field in the Options page is labeled Collation—I discussed this option in Chapter 1 when installing SQL Server. If you need to alter a collation setting on a database, you can do so, but care is required. Note that altering the collation sequence on a server should be undertaken only by the system administrator, who will be aware of the issues and have the authority to perform this task.

The next setting is Recovery Model. You'll learn about backing up and restoring your database in Chapter 7, and this option forms part of that decision-making process. In development, the best option is to choose the Simple backup mode, as you should have your most up-to-date source being developed and saved to your local hard drive. The three modes are as follows:

- *Full*: Allows the database to be restored to where the failure took place; every transaction is logged; therefore, you can restore a database backup and then move forward to the individual point in time required using the transaction log.
- *Bulk-Logged*: Minimally logs bulk operations, so if you're performing a bulk operation such as bulk copying into SQL Server, or if you're inserting a bulk of rows of data, then only the action is recorded and not every row is inserted; this will increase performance during these sorts of operations, but if a problem occurs, then recovery can occur only to the end of the last log backup.
- *Simple*: Truncates the transaction log after each database backup; this allows restores to be created to the last successful data backup only, as no transaction log backups are taken. You should not use this mode in a production environment except in low-volume, low-risk applications.

■ **Note** It is possible to switch modes, and you may want to do this if you have a number of bulk-logged operations—for example, if you're completing a refresh of static data.

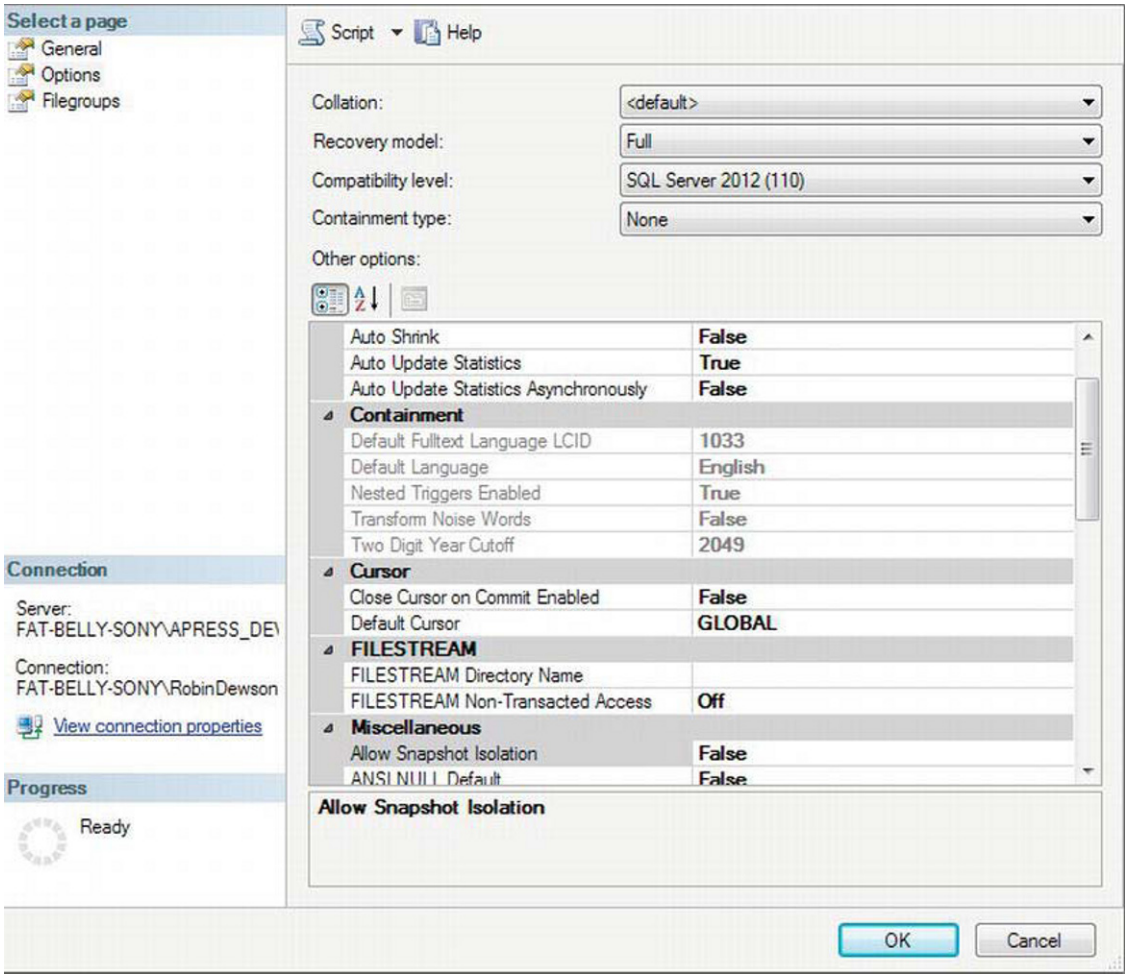

Figure 3-11. *Options page of the New Database dialog*

The third item in the Options page is Compatibility Level. It is possible to build a database for previous versions of SQL Server, provided you are willing to sacrifice the new functionality. This will allow you to connect to a SQL Server 2008 or earlier–defined database.

Among the next set of options, the ones of interest to you at the moment are the first five. You will examine the remaining options when you build the database using T-SQL.

- *Auto Close*: If you want the database to shut down when the last user exits, then set this option to True. The standard practice is a setting of False, and you should have a good reason to set this option to True, especially on a remote server.
- *Auto Create Statistics*: This option relates to the creation of statistics used when querying data. The standard practice is a setting of True; however, in a production environment, especially if you have a nightly or weekly process that generates statistics on your data, you would switch this to False. Creating and updating statistics while your system is being used does increase processing required on your server, and if your server is heavily used for inserting data, then you will find a performance degradation with this option set to True. To clarify, though, it is necessary to balance your choice with how much your system will have to query data.
- *Auto Shrink*: Database and transaction logs grow in size not only with increased data input, but also through other actions, which I'll discuss in more detail in Chapter 7. You can shrink the logical size of the log file through certain actions, some of which can be instigated by T-SQL and some as a by-product of actions being performed. Shrinking a database, whether automatically or manually, takes processing power and time. You should be in control of when shrinking takes place rather than leaving this to SQL Server, and therefore Auto Shrink should be set to False in a production environment. Even if the database is remote, alerts should be set up to warn you about database size reaching its limit rather than allowing SQL Server to perform this task unattended.
- *Auto Update Statistics and Auto Update Statistics Asynchronously*: These are more common options to have set to True, even on production servers, although there is still a performance degradation. These options will update statistics as data are inserted, modified, or deleted for tables for use in indexes, and they will also update statistics for columns within a table. I will discuss indexes further in Chapter 6. Statistics are used by SQL Server to find the best method to work with your data, and having good statistics will ensure the best method is normally chosen.

■ **Tip** You need not create the database at this point if you don't want to. There are several other options available to you to save the underlying T-SQL to a file, to the clipboard, or to the Query window. The first two options are very useful as methods of storing actions you're creating to keep in your source code repository, such as Visual SourceSafe. The first two options are also useful if you wish to keep T-SQL code to execute when deploying on several different computers at client sites. Clients may have SQL Server installed on a server but no SSMS installed locally from which to build a database graphically. Finally, by using T-SQL rather than dialog screens, you will achieve consistency and clarity when performing an action multiple times. The third option is ideal if you wish to add more options to your database than you have defined within the wizard setup. All of the

options enable you to see the underlying code and understand what is required to create a database. You will look at the code in a moment.

7. Click the OK button at the bottom of the screen to create the database.

SQL Server will now perform several actions. First, it checks whether the database already exists; if so, you will have to choose another name. Second, once the database name is validated, SQL Server does a security check to make sure the user has permission to create the database. This is not a concern here, since by following this book, you will always be logged on to SQL Server with the proper permissions. Now that you have security clearance, the data files are created and placed on the hard drive. Provided there is enough space, these files will be successfully created, and it is not until this point that SQL Server is updated with the new database details in the internal system tables.

Once this is done, the database is ready for use. As you can see, this whole process is relatively straightforward.

When you return to Object Explorer in SQL Server Management Studio, refresh the contents manually if the explorer hasn't autorefreshed. You will see the new database listed, as shown in Figure 3-12.

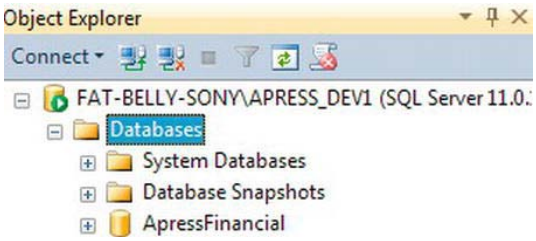

Figure 3-12. *The new database within Object Explorer*

SQL Server Management Studio is simply a GUI front end to running T-SQL scripts in the background. As you progress through the book, you'll see the T-SQL generated for each object type you're using, and you'll create the objects graphically, as you've just seen.

Once the database has been created, you can right-click and, as shown in Figure 3-13, have the details sent to one of four locations.

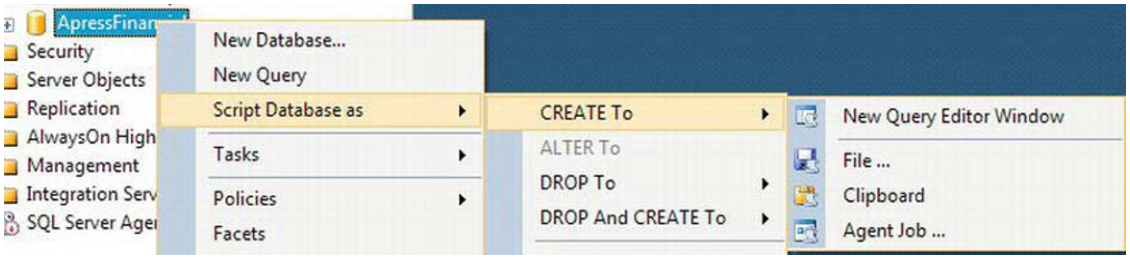

Figure 3-13. *Scripting the database from SSMS*

Whichever method you choose to use, the script will be the same, with the exception of a comment line when you create the script in the second option. The script for generating the database from this option is listed here, so you can go through what is happening. The fourth option allows you to schedule a re-create of the database at a certain point in time. This is ideal to use when building a database from scratch on a regular basis. For example, you might create a daily job for setting up a test area.

First of all, SQL Server points itself to a known database, as shown in the following snippet. `master` has to exist; otherwise, SQL Server will not work. The `USE` statement, which instructs SQL Server to alter its connection to default to the database after the `USE` statement, points further statements to the `master` database:

```
USE [master]

GO
```

Next, the script builds up the `CREATE DATABASE` T-SQL statement built on the options selected. (I'll walk you through the `CREATE DATABASE` syntax that could be used in the "Creating a Database in a Query Pane" section, as this statement doesn't cover all the possibilities.) Notice in the code that follows the next note that the name of the database is surrounded by square brackets: `[]`. SQL Server does this as a way of defining that the information between the square brackets is to be used similarly to a literal and not as a variable. Also it defines that the information is to be treated as one unit. To clarify, if you want to name the database `Apress Financial` (i.e., with a space between "Apress" and "Financial"), then you and SQL Server need to have a method of knowing where the name of the database starts and ends. This is where the identifier brackets come in to play.

■ **Note** Recall the Set Quoted Identifier option you encountered in Chapter 2, with the T-SQL command `SET QUOTED_IDENTIFIER ON/OFF`. Instead of using the square brackets around `master` as you see in the preceding text, you can define identifiers by surrounding them with double quotation marks using this command. Therefore, anything that has double quotation marks around it is seen as an identifier rather than a literal, if this option is set to `ON`. To get around this requirement, you can use single quotation marks, as shown in the example, but then if you do have to enter a single quotation mark—as in the word "don't"—you would have to use another single quotation mark. So as you can see, this situation can get a bit messy. I prefer to have `QUOTED_IDENTIFIER` set to `OFF` to reduce confusion.

The following code is generated by the script for creating the `ApressFinancial` database:

```
CREATE DATABASE ApressFinancial
CONTAINMENT = NONE
ON  PRIMARY
( NAME = N'ApressFinancial',
FILENAME = N'C:\Program Files\Microsoft SQL Server\MSSQL11.APRESS_DEV1
\MSSQL\DATA\ApressFinancial.mdf' , SIZE = 4096KB ,
MAXSIZE = UNLIMITED, FILEGROWTH = 1024KB )
 LOG ON
( NAME = N'ApressFinancial_log',
FILENAME = N'C:\Program Files\Microsoft SQL Server\MSSQL11.APRESS_DEV1
\MSSQL\DATA\ApressFinancial_log.ldf' ,
```

```
SIZE = 1024KB , MAXSIZE = 2048GB , FILEGROWTH = 10%)
GO
```

■ **Note** Most of the preceding code should be easy to understand and follow as it is taking the details entered in the database creation dialog screen and transforming those details into T-SQL. However, there is one item I would like to bring to your attention and that is the `CONTAINMENT = NONE` argument. With SQL Server 2012, databases can be partially contained or non-contained. Future releases will expand this functionality to include fully contained databases as found on the SQL Azure (SQL Server in the cloud) platform. A fully contained database is one in which all of the objects, data, accounts, and metadata are contained within that database. You can then move databases between SQL Server instances and installations much more easily than is the case with partial or non-contained databases, as all of the information required to move a database is held by the database itself. Previous versions of SQL Server are seen as non-contained, and therefore porting databases have problems with database ownership and some of the metadata required to be rebuilt when placed on the new server. Partially contained databases have some of the information held within the database itself, although code can have dependence outside of its own database. One example would be when trying to retrieve customer information from the `Customer` database from within the `Invoicing` database. Finally, non-contained databases, as created for `ApressFinancial`, are built similarly to previous versions of SQL Server, where to move a database from one server to another requires some work. At this point, I am building the database as a non-contained database for simplicity of discussion throughout the book and to ensure you have the basics of databases clear in your mind before introducing these complexities.

The `CREATE DATABASE` command is followed by a `GO` command. This signals to SQL Server—or any other SQL Server utility—that this is the end of a batch of T-SQL statements, and the utility should send the batch of statements to SQL Server. You saw this in Chapter 2 when you were looking at Query Editor's options. Certain statements need to be in their own batch and cannot be combined with other statements in the same batch. To clarify, a `GO` statement determines that you have come to the end of a batch of statements and that SQL Server should process these statements before moving on to the next batch of statements.

■ **Note** `GO` statements are used only in ad hoc T-SQL, which is what I'm demonstrating here. Later in the book, you'll build T-SQL into programs called stored procedures. `GO` statements are not used in stored procedures.

Next, you define the new database's compatibility level. The following `ALTER DATABASE` statement sets the database's base level to SQL Server 2012. It is possible to define SQL Server to an earlier level, as far back as SQL Server 2000, by changing the version number. A base level of 100 actually means 10.0, as in version 10 (100), which equates to SQL Server 2008.

```
ALTER DATABASE [ApressFinancial] SET COMPATIBILITY_LEVEL = 110
GO
```

You then can define the remaining database options. The statements to set those options have GO statements separating them. But in this scenario, the GO statement is superfluous. So why are they included? When SQL Server is preparing the wizard, it is safer for it to place GO statements after each statement, because it then doesn't have to predict what the next statement is, and therefore whether the end of the batch of transactions has to be defined.

It is possible to initialize full-text indexing or remove all full-text catalogs from the current database. By default, all user-created databases in SQL Server 2012 are enabled for full-text indexing.

There is an IF statement around the following code that enables or disables full-text searching. This code is testing whether full-text indexing has been installed as part of the current instance. If it has not been installed, then by default the option is not enabled.

```
IF (1 = FULLTEXTSERVICEPROPERTY('IsFullTextInstalled'))
begin
EXEC [ApressFinancial].[dbo].[sp_fulltext_database] @action = 'enable'
end
```

There will be times when columns have no data in them. When a column is empty, it is said to contain the special value of NULL. Setting ANSI_NULL_DEFAULT to OFF means that a column's default value is NOT NULL. You'll learn about NULL values in Chapter 5 during the table creation discussion. The following is the statement to define the default setting for a new column definition when defining a new column within a table in SQL Server. If you define a new column for a table without defining whether it can hold NULL values, using the T-SQL ALTER TABLE command, then the column by default will not allow NULL values.

```
ALTER DATABASE [ApressFinancial] SET ANSI_NULL_DEFAULT OFF
GO
```

Still on the topic of NULL values, the ANSI standard states that if you are comparing two columns of data that have this special NULL value in them, then the comparison should fail and the two columns will not be considered equal. They also won't be considered not equal to each other. Setting ANSI NULLS to OFF alters that behavior, so that when you do compare two NULL values, they will be considered equal to each other. The following is the statement to use:

```
ALTER DATABASE [ApressFinancial] SET ANSI_NULLS OFF
GO
```

There are columns of characters that can store variable-length data. You'll come across these when you build your table in Chapter 5. If set to ON, this option makes every column of data contain the maximum number of characters, whether you sent through just one character or many more. It is common to have this set to OFF.

```
ALTER DATABASE [ApressFinancial] SET ANSI_PADDING OFF
GO
```

If an ANSI standard warning or error occurs, such as divide by zero, switching the ANSI_WARNINGS setting to OFF will suppress these. A value of NULL will be returned in any columns that have the error.

```
ALTER DATABASE [ApressFinancial] SET ANSI_WARNINGS OFF
GO
```

If the ANSI_WARNINGS setting were ON, and you performed a divide by zero, the query would terminate. To change this in combination with ANSI_WARNINGS set to ON, you tell SQL Server not to abort when there's an arithmetic error.

```
ALTER DATABASE [ApressFinancial] SET ARITHABORT OFF
GO
```

If you have a database that is "active" only when users are logged in, then switching the AUTO_CLOSE setting to ON would close down the database when the last user logged out. This is unusual, as databases tend to stay active 24/7, but closing unwanted databases frees up resources for other databases on the server to use if required. One example of when to switch this setting ON is for a database used for analyzing data by users through the day (for example, one in an actuarial department, where death rates would be analyzed).

```
ALTER DATABASE [ApressFinancial] SET AUTO_CLOSE OFF
GO
```

SQL Server uses statistics when returning data. If it finds that statistics are missing when running a query, having the following option ON will create these statistics.

```
ALTER DATABASE [ApressFinancial] SET AUTO_CREATE_STATISTICS ON
GO
```

If the volume of data within your database grows smaller (for example, if you have a daily or weekly archive process), you can reduce the size of the database automatically by setting the following option ON. It is standard to have the option OFF because the database size will simply increase as data are re-added. It would be switched ON only if a reduction in the database is required—due to disk space requirements, for example—but it is never a good idea for this option to kick in when the database is in use, so really it is best to keep it off.

```
ALTER DATABASE [ApressFinancial] SET AUTO_SHRINK OFF
GO
```

■ **Note** It would be better to shrink the database manually by using the DBCC SHRINKDATABASE command. You will see this in Chapter 7.

When data are added or modified to SQL Server, statistics are created that are then used when querying the data. These statistics can be updated with every modification, or they can be completed via a T-SQL set of code at set times. There is a performance reduction as data are inserted, modified, or deleted, but this performance is gained back when you want to return data. Your application being a pure insertion, pure query, or a mix determines whether you'll want this option on. If you have a pure insertion application, you probably want this option switched off, for example, but this is an optimization decision.

```
ALTER DATABASE [ApressFinancial] SET AUTO_UPDATE_STATISTICS ON
GO
```

A *cursor* is an in-memory table built with T-SQL used for row-at-a-time processing. It's a temporary memory resident table, in essence. A cursor can exist for the lifetime of a program, but if you switch the

following setting to ON, when a batch of data is committed or rolled back during a transaction, the cursor will be closed.

```
ALTER DATABASE [ApressFinancial] SET CURSOR_CLOSE_ON_COMMIT OFF
GO
```

A cursor can exist either locally or globally. This means that if GLOBAL is selected for this option, then any cursor created in a program is available to any subprogram that is called. LOCAL, the other option, indicates that the cursor exists only within that program that created it.

```
ALTER DATABASE [ApressFinancial] SET CURSOR_DEFAULT  GLOBAL
GO
```

If you're concatenating character fields and if the following option is ON, then if any of the columns has a NULL value, the result is a NULL.

```
ALTER DATABASE [ApressFinancial] SET CONCAT_NULL_YIELDS_NULL OFF
GO
```

When you're working with some numeric data types, it is possible to lose precision of the numerics. This can occur when you move a floating-point value to a specific numeric decimal point location, and the value you're passing has too many significant digits. If the following option is set to ON, then an error is generated. OFF means the value is truncated.

```
ALTER DATABASE [ApressFinancial] SET NUMERIC_ROUNDABORT OFF
GO
```

As mentioned earlier, when you're defining database names, if there is a space in the name or the name is a reserved word, it is possible to tell SQL Server to ignore that fact and treat the contents of the squared brackets as a literal. You are using *quoted identifiers* when you use the double quotation mark instead of square brackets. I'll delve into this topic further when showing how to insert data in Chapter 9, as there are a number of details to discuss with this option.

```
ALTER DATABASE [ApressFinancial] SET QUOTED_IDENTIFIER OFF
GO
```

The following option relates to a special type of program called a *trigger*. A trigger can run when data are modified, and one trigger can call another trigger. A setting of OFF means that an AFTER trigger is not allowed to recursively cause itself to be invoked. This is known as a direct recursion.

```
ALTER DATABASE [ApressFinancial] SET RECURSIVE_TRIGGERS OFF
GO
```

Service Broker provides developers with a raft of functionality, such as asynchronous processing or the ability to distribute processing over more than one computer. Such a scenario might be heavy overnight batch processing that needs to be completed within a certain time window. By distributing the processing, it could mean that a process that wouldn't have been distributed could finish within that time frame.

```
ALTER DATABASE [ApressFinancial] SET DISABLE_BROKER
GO
```

I mentioned statistics earlier with another option and how they can be updated as data are modified. The following option is similar to AUTO_UPDATE_STATISTICS. If this option is set to ON, the query that triggers an update of the statistics will not wait for the statistics to be created. The statistics update will start, but it will do so in the background asynchronously.

```
ALTER DATABASE [ApressFinancial] SET AUTO_UPDATE_STATISTICS_ASYNC OFF
GO
```

If there is a relationship between two tables via a foreign key, by setting this option to `ON`, SQL Server will correlate statistics for the two tables to try to improve query optimization. Unless you have this scenario, the setting should be switched to `OFF`, which is the default:

```
ALTER DATABASE [ApressFinancial] SET DATE_CORRELATION_OPTIMIZATION OFF
GO
```

This option defines whether this database is seen as trustworthy with what resources it can access. For example, the option defines whether SQL Server can trust the database not to crash the server. By setting the option to `OFF`, you ensure (among other things) that SQL Server will not allow any code developed to have access to external resources.

```
ALTER DATABASE [ApressFinancial] SET TRUSTWORTHY OFF
GO
```

If you build a database that is set for replication—in other words, where data changes are replicated to another server, which you sometimes see for distributed solutions—then this option details how SQL Server deals with data you are trying to process when it is involved in a transaction. This is an advanced topic outside the scope of this book.

```
ALTER DATABASE [ApressFinancial] SET ALLOW_SNAPSHOT_ISOLATION OFF
GO
```

The basis of the following option is to inform SQL Server how best to work with code that has parameters within it and to decide the best and fastest way to work with that query. To clarify, when you try to retrieve data from a table, you are likely to have some sort of filtering such as a specific value, a range of values, and so on. This is a query parameter. SQL Server can discard the value when trying to figure out the best way to get the data from the database based on previous queries. SQL Server uses previous query executions to try to calculate the fastest method of getting that data, known as a query plan. By keeping the parameterization simple, SQL Server will look at the query, and if the query is a simple query with very few parameters, it will try to find a query plan that it has in memory to use. If the query is complex or SQL Server feels the number of parameters is too high to be deemed simple, then SQL Server will build a new query plan.

```
ALTER DATABASE [ApressFinancial] SET PARAMETERIZATION SIMPLE
GO
```

SQL Server has many features and techniques for ensuring that your data are correct, safe, and valid. In Chapter 9, you will see one way to keep your data correct when you look at transactions. There are two ways SQL Server can enforce data safety: either by locking data that are being updated in a transaction, or by keeping a version of the row prior to the modification in `tempdb` until it is committed via a transaction. This latter technique is known as row versioning. By setting `READ_COMMITTED_SNAPSHOT` to `OFF` as in the following example, you are indicating that you will be taking the approach that the data will be locked. You guarantee that the information being read contains the value currently committed to the database. If you set the option to `ON`, it is possible that you could be reading the data after it has been altered within T-SQL but before it has been committed to the database. There are pros and cons of both methods, and I will cover more about transactions in Chapter 9.

```
ALTER DATABASE [ApressFinancial] SET READ_COMMITTED_SNAPSHOT OFF
GO
```

The option HONOR_BROKER_PRIORITY, shown in the example to follow, is used with SQL Server Service Broker, which is used by SQL Server when XML messages are being passed from one SQL Server to another. One way of thinking about how Service Broker works would be to compare its use to having a chat conversation over the Internet in Facebook. The software accepts the message you type and sends it to the recipients, and the recipients send back an acknowledgement indicating that the message has been successfully received. A setting of ON indicates that if messages are given a priority, then SQL Server will process these in a priority order. A setting of OFF indicates that messages will be processed in order of generation.

```
ALTER DATABASE [ApressFinancial] SET HONOR_BROKER_PRIORITY OFF
GO
```

The following option defines how the filegroups are set: READ_WRITE or READ_ONLY. The use of READ_ONLY is ideal when you have a backup database that users can use to inspect data. The database is an exact mirror of a production database, for example, so it has the security on it set to allow updates to it, but by setting this option to READ_ONLY, you can be sure that no updates can occur.

```
ALTER DATABASE [ApressFinancial] SET READ_WRITE
GO
```

The next option determines how your data can be recovered when a failure such as a power outage happens. In other words, the following option defines the recovery model, as discussed earlier. You'll look at this in more detail when I discuss database maintenance in Chapter 7.

```
ALTER DATABASE [ApressFinancial] SET RECOVERY FULL
GO
```

The following option defines the user access to the database. MULTI_USER is the norm and allows more than one user into the database. The other settings are SINGLE_USER and RESTRICTED_USER, where only people who have powerful privileges can connect. You would set your database to RESTRICTED_USER after a media or power failure, for example, when a database administrator needs to connect to the database to ensure everything is okay. (The setting RESTRICTED_USER causes the database to allow only administrator logins.) Specify SINGLE_USER when you want to lock down the database so that no other person can connect—for example, when you are detaching the database from this instance and you are moving it to another instance. Specify MULTI_USE when you want to open the database up for general use, as in the following example:

```
ALTER DATABASE [ApressFinancial] SET  MULTI_USER
GO
```

When you have an I/O error (e.g., a hard drive might be on its way to breaking down), then this option will report an error if checksums don't match:

```
ALTER DATABASE [ApressFinancial] SET PAGE_VERIFY CHECKSUM
GO
```

Finally, the following line is used for controlling whether permissions checks are required when referring to objects in another database:

```
ALTER DATABASE [ApressFinancial] SET DB_CHAINING OFF
```

Dropping the Database in SQL Server Management Studio

To follow the next section properly and build the database using code, it is necessary to remove the database just created. It is also handy to know how to do this anyway, for those times when you have

made an error or when you want to remove a database that is no longer in use. Deleting a database is also known as *dropping* a database.

TRY IT OUT: DROPPING A DATABASE IN SQL SERVER MANAGEMENT STUDIO

1. If SQL Server Management Studio should still be running from the previous example, expand the nodes until you see the database `ApressFinancial`.
2. Right-click `ApressFinancial` to bring up the context menu.
3. Click the Delete option, as shown in Figure 3-14.

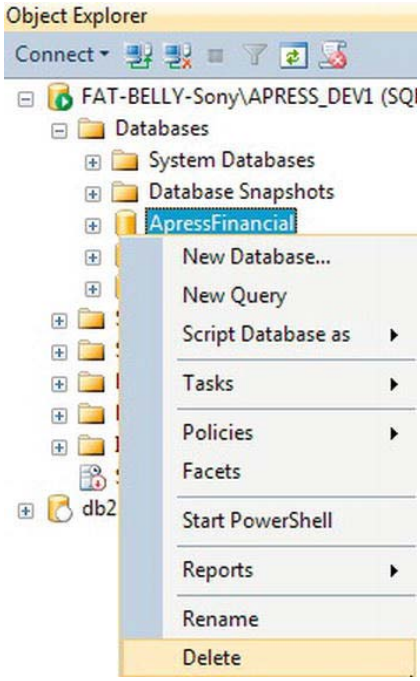

Figure 3-14. *Deleting a database within SSMS*

4. The dialog shown in Figure 3-15 will display. Select Close Existing Connections, and then click OK.

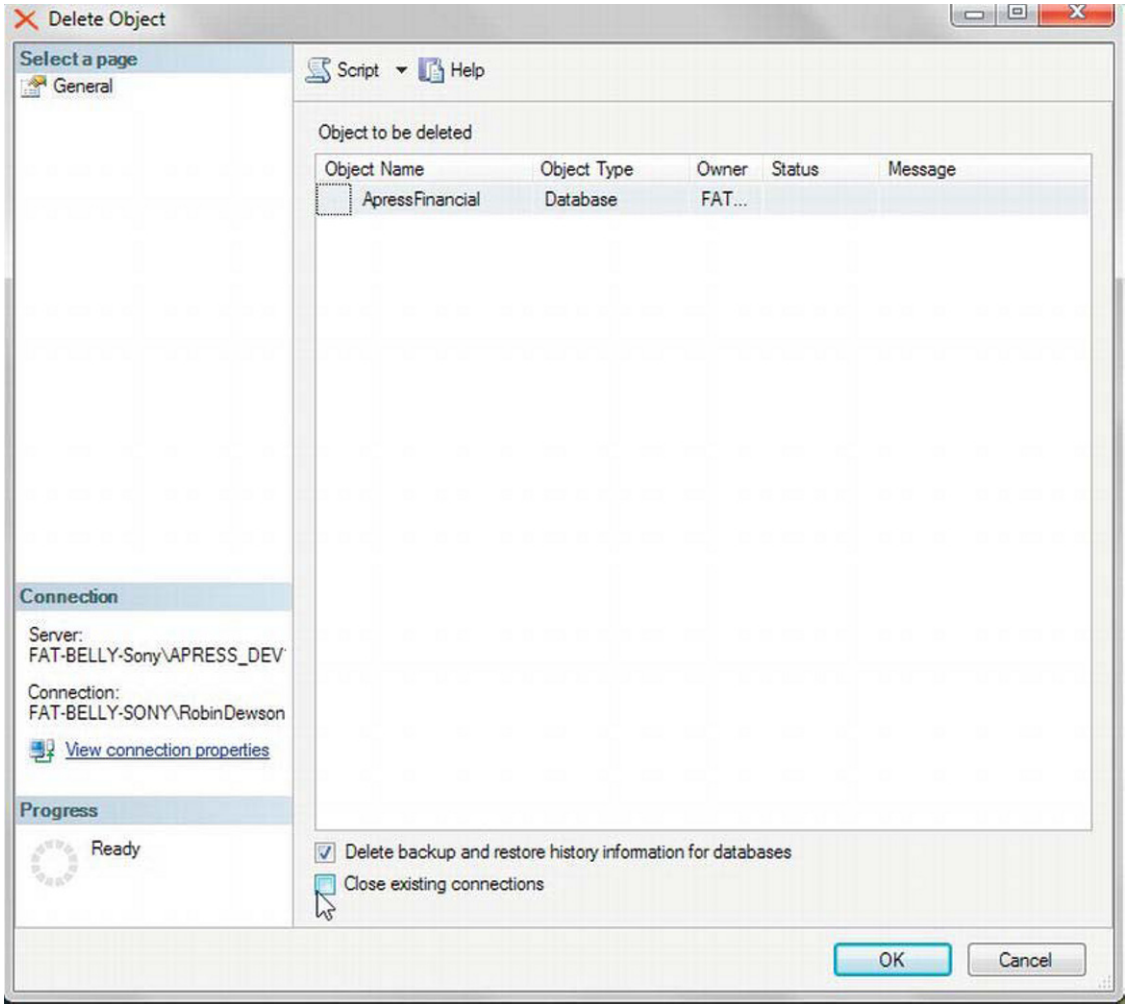

Figure 3-15. *Selecting to delete a database in the Delete Object dialog*

The first check box, Delete Backup and Restore History Information for Databases, gives you the option of keeping or removing the history information that was generated when completing backups or restores. If you want to keep this information for audit purposes, then uncheck the box.

The second check box is very important. If there is a program running against a database, or if you have any design windows or query panes open and pointing to the database you want to delete, then this option will close those connections. If you are deleting a database, then there really should be no connections there. This is a good check and will prevent accidents from happening, and it also allows any rogue databases to be removed without having to track down who is connected to them.

5. Click OK. The database is now permanently removed.You will be re-creating the database in a moment, so it is okay for you to delete.

When you click the OK button, SQL Server actually performs several actions. First, a command is sent to SQL Server informing it of the name of the database to remove. SQL Server then checks that nobody is currently connected to that database. If someone is connected, through either SQL Server Query Editor or a data access method like ADO.NET, then SQL Server will refuse the deletion. Only if you select Close Existing Connections will this process be overridden.

For SQL Server to refuse the deletion, it does not matter whether anyone connected to the database is actually doing anything; all that is important is the existence of the connection. For example, if you selected `ApressFinancial` in Query Editor and then returned to SQL Server Management Studio and tried to drop the database, you would see the error shown in Figure 3-16.

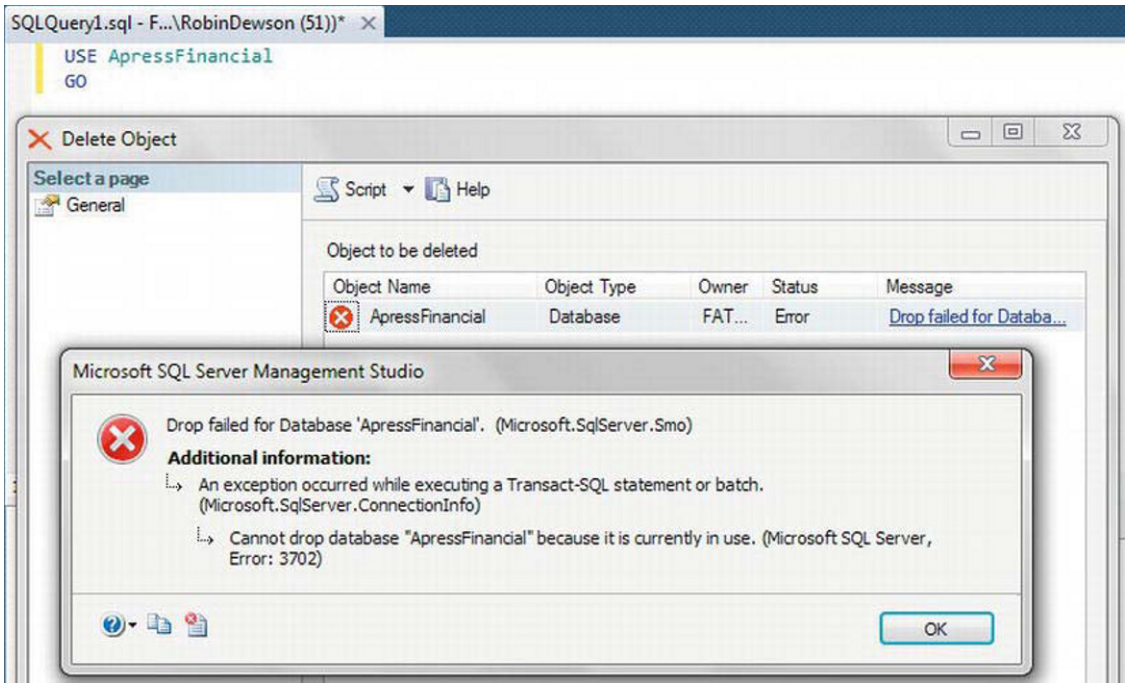

Figure 3-16. *Failed database deletion*

■ **Tip** Errors like the one shown in Figure 3-16 on the screen in the background provide hyperlinks to documentation that can give you further help, as shown in the dialog box in the foreground.

Once SQL Server has checked that nobody is connected to the database, it then checks that you have *permission* to remove the database. SQL Server will allow you to delete the database if it was your user ID that created it, in which case you own this database and SQL Server allows you to do what you want with it. However, you are not alone in owning the database.

If you recall from Chapter 1, there was mention of the `sa` account when installing SQL Server. Since it is the most powerful ID and has control over everything within SQL Server, there were warnings about leaving the `sa` account without any password and also about using the `sa` account as any sort of login ID in general. This section also mentioned that the `sa` account was in fact a member of the `sysadmin` server role. A *role* is a way of grouping together similar users who need similar access to sets of data. Anyone in the `sysadmin` role has full administrative privileges—and this includes rights to remove any database on the server.

So regardless of whether you are logged in as yourself or as `sysadmin`, take care when using SQL Server Management Studio to drop a database.

Creating a Database in a Query Pane

To use the second method of creating databases, you first need to drop the `ApressFinancial` database as described in the previous section. Then you can continue with the following steps.

TRY IT OUT: CREATING A DATABASE IN A QUERY PANE

1. From the standard toolbar of SQL Server Management Studio, select New Query.
2. In the query pane, enter the following T-SQL script:

```
CREATE DATABASE ApressFinancial ON  PRIMARY
( NAME = N'ApressFinancial',
FILENAME = N'C:\Program Files\Microsoft SQL
Server\MSSQL11.APRESS_DEV1\MSSQL\DATA\ApressFinancial.mdf' ,
 SIZE = 4096KB ,
MAXSIZE = UNLIMITED, FILEGROWTH = 1024KB )
 LOG ON
( NAME = N'ApressFinancial_log',
FILENAME = N'C:\Program Files\Microsoft SQL
Server\MSSQL11.APRESS_DEV1\MSSQL\DATA\ApressFinancial_log.ldf' ,
SIZE = 1024KB , MAXSIZE = 2048GB , FILEGROWTH = 10%)
 COLLATE SQL_Latin1_General_CP1_CI_AS
GO
```

3. Execute this code by pressing F5 or by clicking the Execute Query toolbar button.
4. Once the code is executed, you should see the following result:

```
Command(s) completed successfully.
```

HOW IT WORKS: CREATING A DATABASE IN A QUERY

The main focus of this sidebar is the code listed in the previous exercise: the `CREATE DATABASE` command.

When placing code in the Query Editor, you're building up a set of instructions for SQL Server to act on. As you progress through the book, you will encounter many commands that you can place in Query Editor, all of which build up to provide powerful and useful utilities or methods for working with data. An in-depth discussion of Query Editor took place in Chapter 2, so if you need to refresh your memory, take a quick look back at the material covered in that chapter.

Before you actually look at the code itself, you need to inspect the syntax of the `CREATE DATABASE` command:

```
CREATE DATABASE <database name>
[CONTAINMENT = NONE | PARTIAL | FULL]
[ON
   ( [ NAME = logical_name, ]
       FILENAME = physical_file_name
     [, FILESIZE = size ]
     [, MAXSIZE = maxsize ]
     [, FILEGROWTH = growth_increment] ) ]
[LOG ON
   ( [ NAME = logical_name, ]
       FILENAME = physical_file_name
     [, FILESIZE = size ]
     [, MAXSIZE = maxsize ]
     [, FILEGROWTH = growth_increment] ) ]
[COLLATE collation_name ]
```

The parameters are as follows:

- `database name`: The name of the database that the `CREATE DATABASE` command will create within SQL Server
- `Containment`: Defines whether the database is fully contained or whether there is usage of metadata, logins, and so on, outside of the database
- `ON`: The `ON` keyword informs SQL Server that the command will specifically mention where the data files are to be placed, as well as their name, size, and file growth. With the `ON` keyword comes a further list of comma-separated options:
- `NAME`: The logical name of the data file that will be used as the reference within SQL Server
- `FILENAME`: The physical file name and full path where the data file will reside

- SIZE: The initial size, in megabytes by default, of the data file specified; this parameter is optional, and if omitted, it will take the size defined in the model database. You can suffix the size with KB, MB, GB, or TB (terabytes). The minimum size for a database will be the same size as the model system database, defined by default as 4MB. MAXSIZE indicates the maximum size the database can grow to.
- FILEGROWTH: The amount that the data file will grow each time it fills up; you can specify either a value that indicates by how many megabytes the data file will grow or a percentage, as discussed earlier when you created a database with SQL Server Management Studio.
- LOG ON: The use of the LOG ON keyword informs SQL Server that the command will specifically mention where the log files will be placed, and their name, size, and file growth:
- NAME: The name of the log file that will be used as the reference within SQL Server
- FILENAME: The physical file name and full path to where the log file will reside; you must include the suffix .LDF. This could be a different name from the FILENAME specified earlier.
- SIZE: The initial size, in megabytes by default, of the log file specified; this parameter is optional, and if omitted, it will take the size defined in the model database. You can suffix the size with KB, MB, GB, or TB.
- FILEGROWTH: The amount by which the log file will grow each time the data file fills up, which has the same values as for the data file's FILEGROWTH
- COLLATE: The collation used for the database; collation was discussed earlier in the chapter when you created a database with SQL Server Management Studio.

It's now time to inspect the code entered into Query Analyzer that will create the ApressFinancial database.

Commencing with CREATE DATABASE, you are informing SQL Server that the following statements are all parameters to be considered for building a new database within SQL Server. Some of the parameters are optional, and SQL Server will include default values when these parameters are not entered. But how does SQL Server know what values to supply? Recall that at the start of this chapter, I discussed the built-in SQL Server databases—specifically, the model database. SQL Server takes the default options for parameters from this database unless they are otherwise specified. Thus, it is important to consider carefully any modifications to the model database.

The database name is obviously essential, and in this case, ApressFinancial is the chosen name.

The ON parameter provides SQL Server with specifics about the data files to be created, rather than taking the defaults. Admittedly, in this instance, there is no need to specify these details, as by taking the defaults, SQL Server would supply the parameters as listed anyway.

This can also be said for the next set of parameters, which deal with the Transaction Log found with `LOG ON`. In this instance, there is no need to supply these parameters, as again the listed amounts are the SQL Server defaults.

Finally, the collation sequence you specify is actually the default for the server.

Taking all this on board, the command could actually be entered as follows, which would then take all the default settings from SQL Server to build the database:

```
CREATE DATABASE ApressFinancial
```

You can then set the database options as outlined during the discussion of the script earlier in the chapter.

Similarly, if you want to delete the database using T-SQL code, it's a simple case of ensuring that you are not connected within that particular query pane to `ApressFinancial` via the `USE` command. Then you use the command `DROP` followed by the object you want to drop, or delete, and then the name of the object.

```
USE Master
GO
DROP DATABASE ApressFinancial
```

You may be thinking that it is a lot of work to type in the `CREATE DATABASE` T-SQL statement when there is a quick and simple screen that can do the job for you. This is true; however, there are limitations to using the screen that sometimes make T-SQL the better option. I have mentioned before that by using T-SQL you can place code in programs. You will see how to do that in Chapter 16. Using T-SQL also enables you to store the code in a source repository such as Visual SourceSafe that then allows you to track any amendments to the database code as you release upgrades to your application. T-SQL can also be read by other developers to ensure the code is delivering what is expected. Finally, in some institutions, auditors will visit your premises to check code releases to ensure that the release code matches what was sought.

Summary

In this chapter, you looked at designing and creating the example database. Though just a sample database, it still required careful thought.

In the next chapter, you will take a look at security and start investigating how to ensure your database is safe and secure from within and outside your organization.