12

Introducing Databases

WHAT YOU WILL LEARN IN THIS CHAPTER:

- What a database is and which databases are typically used with ASP.NET pages
- What SQL is, how it looks, and how you use it to manipulate data
- What database relationships are and why they are important
- Which tools you have available in VWD to manage objects (such as tables) and how to use them

Being able to use a database in your ASP.NET 4 web sites is just as critical as understanding HTML and CSS: it's almost impossible to build a modern, full-featured web site without it. Databases are useful because they enable you to store and retrieve data in a structured way. The biggest benefit of databases is that they can be accessed at runtime, which means you are no longer limited to just the relatively static files you create at design time in Visual Web Developer. You can use a database to store reviews, musical genres, pictures, information about users (usernames, e-mail addresses, passwords, and so on), log information about who reads your reviews, news articles, and much more, and then access that data from your ASPX pages.

This gives you great flexibility in the data you present, and the way you present it, enabling you to create highly dynamic web sites that can adapt to your visitors' preferences, to the content your site has to offer, or even to the roles or access rights that your users have.

To successfully work with a database in an ASPX page, this chapter teaches you how to access databases using a query language called *SQL* — or *Structured Query Language*. This language enables you to retrieve and manipulate data stored in a database. You also see how to use the database tools in VWD to create tables and queries.

Although ASP.NET and the .NET Framework offer you many tools and technologies that enable you to work with databases without requiring a firm knowledge of the underlying concepts like SQL, it's still important to understand them. Once you know how to access a database, you'll find it easier to understand and appreciate other technologies, like the ADO.NET Entity Framework (discussed in Chapter 14), which provides easier access to database operations directly from code.

In the chapters that follow, you apply the things you learn in this chapter. In Chapter 13, you see how to use built-in controls to work with data in your database. In Chapter 14, you learn how to use the ADO.NET Entity Framework as an additional layer on top of your database to access data in an object-oriented way with minimal code. Chapter 15, the last of the data-focused chapters, shows you advanced techniques for working with data.

In the following sections, you see what a database is, and what different kinds of databases are available to you.

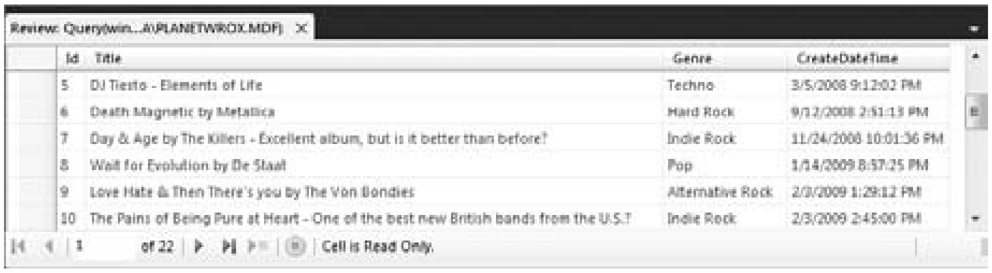
WHAT IS A DATABASE?

By its simplest definition, a database is a collection of data that is arranged so it can easily be accessed, managed, and updated. For the purposes of this book, and the web sites you will build, it's also safe to assume that the data in the database is stored in an electronic format.

The most popular type of database is the *relational database*. It's the type of database that is frequently used in web sites and is also the type of database that is used in the remainder of this book. However, the relational database is not the only one. Other types exist, including flat-file, object-relational, and object-oriented databases, but these are less common in Internet applications.

A relational database has the notion of *tables* where data is stored in rows and columns, much like a spreadsheet. Each row in a table contains the complete information about an item that is stored in the table. Each column, on the other hand, contains information about a specific property of the records in the table.

The term "relational" refers to the way the different tables in the database can be related to each other. Instead of duplicating the same data over and over again, you store repeating data in its own table and then create a relationship to that data from other tables. Consider the table called `Review` in Figure 12-1. This table could store the CD or concert reviews that are presented on the Planet Wrox web site.



Id	Title	Genre	CreateDateTime
5	DJ Tiesto - Elements of Life	Techno	3/5/2008 9:12:02 PM
6	Death Magnetic by Metallica	Hard Rock	9/12/2008 2:51:13 PM
7	Day & Age by The Killers - Excellent album, but is it better than before?	Indie Rock	11/24/2008 10:01:36 PM
8	Wait for Evolution by De Staat	Pop	1/14/2009 8:57:25 PM
9	Love Hate & Then There's you by The Von Bondies	Alternative Rock	2/3/2009 1:29:12 PM
10	The Pains of Being Pure at Heart - One of the best new British bands from the U.S.?	Indie Rock	2/3/2009 2:45:00 PM

FIGURE 12-1

As you can see in Figure 12-1, each review is assigned to a musical genre such as Pop, Indie Rock, or Techno. But what if you wanted to rename the genre Techno to something like Hardcore Techno?

You would need to update all the records that have this genre assigned. If you had other tables that stored a genre, you would need to visit those tables as well and manually make the changes.

A much better solution would be to use a separate table and call it `Genre`, for example. This table could store the name of a genre and an ID (a sequential number for example) that uniquely identifies each genre. The `Review` table then has a relationship to the `Genre` table and stores only its ID instead of the entire name. Figure 12-2 shows the conceptual model for this change.

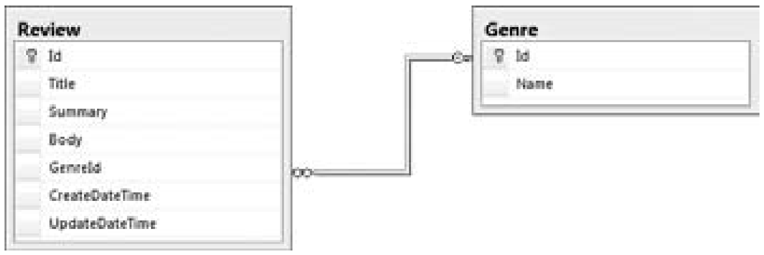


FIGURE 12-2

With just the ID of the genre now stored in the `Review` table, it's easy to rename a genre. All you need to do is change the name of the genre in the `Genre` table, and all tables with a relationship to that genre pick up the change automatically. Later in this chapter, you see how to create and make use of relationships in your relational database.

DIFFERENT KINDS OF RELATIONAL DATABASES

You can use many different kinds of databases in your ASP.NET projects, including Microsoft Access, SQL Server, Oracle, and MySQL. However, the most commonly used database in ASP.NET 4 web sites is probably Microsoft SQL Server. This book focuses on using Microsoft SQL Server 2008 Express edition, because it comes free with VWD and has a lot to offer out of the box. Also, because the database engine is identical to that of the commercial versions of SQL Server 2008, it's easy to upgrade to those versions at a later stage in the development cycle. This upgrade path is described in more detail in Appendix B. Even if you already have a different version of SQL Server 2008 installed, it's still recommended to install a copy of the 2008 Express edition as well. This makes it easy to follow along with the examples in this book, and with many examples you find on the Internet, which often assume you're using the Express edition.

To work with the data in the database, SQL Server (and all other relational database systems) supports a query language called SQL.

USING SQL TO WORK WITH DATABASE DATA

To get data in and out of a database, you need to use *Structured Query Language* (SQL). This is the de facto language for querying relational databases that almost all relational database systems understand. A number of clear standards exist, with the most popular one being the ANSI 92 SQL standard. Besides the grammar that this standard supports, many database vendors have added their own extensions to the language, giving it a lot more flexibility and power on their own system, at the cost of decreased interoperability with other systems.

Microsoft SQL Server 2008 is no exception, and supports most of the grammar that has been defined in the ANSI 92 SQL Standard. On top of this standard, Microsoft has added some proprietary extensions. Collectively, the two are referred to as T-SQL, or Transact SQL. I'll stick to the term SQL for the remainder of this book.

In the following sections, you see how to use SQL targeting a SQL Server 2008 database to retrieve and manipulate data in your database. However, before you can write your first SQL statement, you need to know how to connect to your database first. The following exercise shows you how to connect to the sample database that comes with the downloadable code for this book.

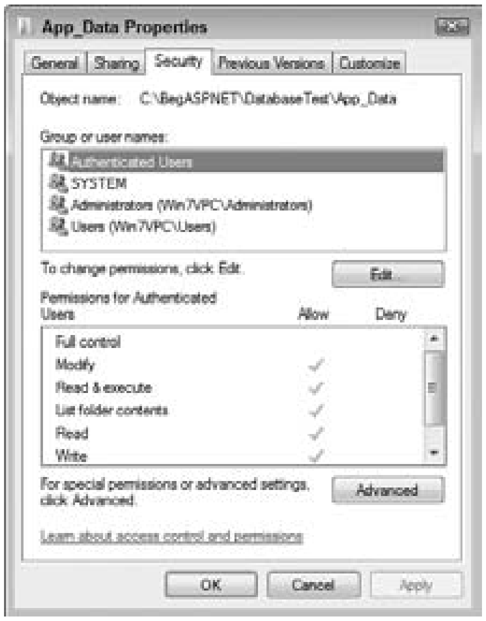
TRY IT OUT Connecting to the SQL Server Sample Database

In this exercise you learn how to connect to and work with a database from within VWD. To give you something to work with, the code download for this chapter comes with a sample database that contains a few tables. To be able to access the database from within VWD, the account that you use to log on to your Windows machine needs at least read and write permissions to the folder where the database resides. If you are logged on as an Administrator, there's a fair chance this is already the case. When you get errors while accessing the database in the next exercise, refer to Chapter 19, in the section "Understanding Security in IIS" for detailed instructions about setting up the proper permissions.

1. For this Try It Out exercise you need a brand new web site, which you can create by choosing File ➪ New Web Site (or File ➪ New ➪ Web Site) in VWD and then choose the ASP.NET Web Site template. Don't follow this exercise with the Planet Wrox web site you've been working on so far or you'll get in trouble later when a database with the same name is used. You can save the web site in a folder like `C:\BegASPNET\DatabaseTest`. If you use a different location, make note of it because you need it in the next step.

2. After you have created the new web site, ensure that your account has sufficient permissions to write to its `App_Data` folder. Because VWD, and thus the built-in web server, runs under the account that you use to log in to Windows, you need to make sure that your account has the correct permissions. To this end, open Windows Explorer (not VWD's Solution Explorer), and locate the folder `C:\BegASPNET\DatabaseTest`. Right-click the `App_Data` folder and choose Properties. Switch to the Security tab, and ensure that your account (or a group you have been assigned to) has at least the Modify permission, as shown in Figure 12-3.

   If you don't have a Security tab, or your account or the Users group is not listed, refer to the Try It Out entitled "NTFS Settings for Planet Wrox" in Chapter 19 for detailed instructions on adding your own account to this list.



Object name: C:\BegASPNET\DatabaseTest\App_Data

Group or user names:
- Authenticated Users
- SYSTEM
- Administrators (Win7VPC\Administrators)
- Users (Win7VPC\Users)

To change permissions, click Edit.

Permissions for Authenticated Users	Allow	Deny
Full control		
Modify	✓	
Read & execute	✓	
List folder contents	✓	
Read	✓	
Write	✓	

For special permissions or advanced settings, click Advanced.

FIGURE 12-3

3. Open the `Resources` folder at `C:\BegASPNET\Resources` using Windows Explorer. If you don't have this folder, refer to the Introduction of this book to learn how to acquire the code that comes with this book. Then open the `Chapter 12` folder and select the file `PlanetWrox.mdf`. Arrange VWD and the Windows Explorer side by side and then drag the file from the Windows Explorer into the `App_Data` folder of your web site in VWD. Remember, if drag and drop doesn't work, you can accomplish the same thing using copy and paste. This `.mdf` file is the actual database and contains tables, records, and so on. When you start working with the database, you may also see an `.ldf` file appear in the `App_Data` folder on disk. This file is used by SQL Server to keep track of changes made to the database. If you are using SQL Server 2005 Express edition, you should get the database from the `Sql2005` folder instead.
4. Double-click the database file in the Solution Explorer in VWD. Doing so opens the database in the Database Explorer (called the Server Explorer in the commercial versions of Visual Studio).
5. You can now expand the connected database to access its objects, such as the tables and columns it contains, shown in Figure 12-4.

How It Works

To be able to access a database from within VWD, it needs to be registered in the Database Explorer window under Data Connections. In most cases, adding a database is as simple as adding the database to your `App_Data` folder and double-clicking it. However, when security on your system is tight, or you are not connecting to a physical SQL Server data file located in your web site, connecting to a database may be a bit trickier. In those cases, refer to the section "Understanding Security in IIS" in Chapter 19 and to Appendix B for more details about configuring your system. The `App_Data` folder is used only for data that is used at the server, such as databases and text files. Because the web server blocks access to this folder for remote browsers, you can't use it to store files such as images that must be downloaded by the client.

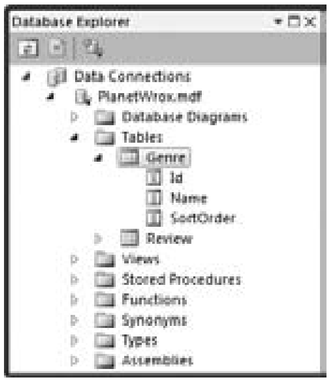


FIGURE 12-4

When you have a connection to your database in the Database Explorer, you can work with the objects in the database. In a later exercise you see how to store the information about the connection to the database (called the *connection string*) in the `web.config` file of your web site so you can use the database from code and with ASP.NET controls.

However, first you need to understand how you can access and change the data in your database.

RETRIEVING AND MANIPULATING DATA WITH SQL

When interacting with databases, you'll spend a good deal of time retrieving and manipulating data. Most of it comes down to four distinct types of operations, grouped under the *CRUD* acronym: *Create*, *Read*, *Update*, and *Delete*.

Because these data operations are so crucial, the next couple of sections show you how to use them in detail.

Reading Data

To read data from a database, you typically use a few different concepts. First, you need the ability to indicate the columns that you want to retrieve from the table you are querying. You do that with the `SELECT` statement. You need to indicate the table(s) you want to select the data from using the `FROM` keyword. Then you need a way to filter the data, making sure only the records you're interested in are returned. You can filter the data using the `WHERE` clause in the SQL statement. Finally, you can order your results using the `ORDER BY` clause.

Selecting Data

To read data from one or more database tables, you use the `SELECT` statement. In its most basic form, the `SELECT` statement looks like this:

```
SELECT ColumnName [, AnotherColumnName] FROM TableName
```

Here, the parts between the square brackets are considered optional. For example, to retrieve all rows from the `Genre` table and only select their `Id` and `Name` columns you use this SQL statement:

```
SELECT Id, Name FROM Genre
```

Right after the `SELECT` statement comes a comma-separated list of column names. You can have only one or as many columns as you like here. Instead of explicitly specifying the column names, you can also use the asterisk (`*`) character to indicate you want all columns to be returned. However, using `SELECT *` is usually considered a poor programming practice so it's better to explicitly define each column you want to retrieve.

Right after the `FROM` keyword, you specify the name of the table from which you want to retrieve data. The previous example showed only one table (the `Genre` table), but you see later that you can also specify multiple tables using joins.

NOTE *Although the SQL language is not case sensitive, it's common practice to write all keywords such as `SELECT` and `FROM` in all caps. Additionally, this book uses Pascal casing — where each new word is capitalized — for names of tables, columns, and so on. For example, the date and time a certain review is created are stored in a column called `CreateDateTime` in the `Review` table.*

Filtering Data

To filter data, you use the `WHERE` clause, with which you indicate the criteria that you want your data to match. For example, to retrieve the ID of the Grunge genre you use the following SQL statement:

```
SELECT Id FROM Genre WHERE Name = 'Grunge'
```

Note that the word Grunge is wrapped in single quotes. This is required for text data types and dates when you filter data or want to send values to an `INSERT` or `UPDATE` statement that enables you to create new or change existing records, as explained later. You can't use them for numeric or Boolean types, though, so to get the name of the genre with an ID of 8 you would use the following statement:

```
SELECT Name FROM Genre WHERE Id = 8
```

The preceding two examples show a `WHERE` clause that uses the equals operator for an exact match. However, you can also use other operators for different criteria. The following table lists a few popular comparison operators you can use in your `WHERE` clauses.

OPERATOR	DESCRIPTION
`=`	The *equals* operator matches only when the left side and the right side of the comparison are identical.
`>`	The *greater than* operator matches when the left side of the comparison represents a larger value than the right side.
`>=`	The *greater than or equal* operator matches when the left side of the comparison is equal to or larger than the right side.
`<`	The *less than* operator matches when the left side of the comparison represents a value smaller than the right side.
`<=`	The *less than or equal* operator matches when the left side of the comparison is equal to or smaller than the right side.
`<>`	The *not equals* operator does the reverse of the equals operator and matches when the left side and the right side of the comparison are different.

To combine multiple `WHERE` criteria, the SQL language supports a number of logical operators such as `AND` and `OR`. In addition, it supports other operators to search for text and to specify ranges. The following table lists a few of the operators and describes what they are used for.

OPERATOR	DESCRIPTION
`AND`	Enables you to join two expressions. For example, the `WHERE` clause ... `WHERE Id > 20 AND Id < 30` gives you all records with IDs that fall between 20 and 30 (with 20 and 30 themselves not included).
`OR`	Enables you to define multiple criteria of which only one has to match (although more matches are allowed). For example, this `WHERE` clause ... `WHERE Id = 12 OR Id = 27` gives you all the records with an ID of 12 or 27.

continues

(continued)

OPERATOR	DESCRIPTION
`BETWEEN`	Enables you to specify a range of values that you want to match with a lower and upper bound. For example, … `WHERE Id BETWEEN 10 AND 35` gives you all records whose IDs are between 10 and 35 (including 10 and 35 themselves if they exist in the database).
`LIKE`	Used to determine if a value matches a specific pattern. You can use wildcards like % to match any string of zero or more characters, and the underscore (_) to match a single character. For example, the following `WHERE` clause … `WHERE Name LIKE '%rock%'` returns all genres that have `rock` in their name, including Indie Rock, Hard Rock, and so on.

If no records match the `WHERE` clause, you don't get an error, but you simply get zero results back.

After you have defined your filtering requirements with the `WHERE` clause, you may want to change the order in which the results are returned from the database. You do this with the `ORDER BY` clause.

Ordering Data

The `ORDER BY` clause comes at the end of the SQL statement and can contain one or more column names or expressions, which can optionally include `ASC` or `DESC` to determine if items are sorted in ascending order (with `ASC`, which is the default if you leave out the keyword) or in descending order (using `DESC`).

For example, to retrieve all genres from the `Genre` table and sort them alphabetically by their name in ascending order, you can use this SQL statement:

```
SELECT Id, Name FROM Genre ORDER BY Name
```

Because ascending is the default order, you don't need to specify the `ASC` keyword explicitly, although you could if you wanted to. The next example is functionally equivalent to the preceding example:

```
SELECT Id, Name FROM Genre ORDER BY Name ASC
```

If you wanted to return the same records but sort them in reverse order on their `SortOrder` column, you use this syntax:

```
SELECT Id, Name FROM Genre ORDER BY SortOrder DESC
```

Notice how you can order by columns in the `ORDER BY` statement that are not part of the `SELECT` statement as is the case with the `SortOrder` column. So, even though a specific column is not part of the final result set, you can still use it to order on.

In the next exercise, you see how to perform a number of queries against the sample database, giving you a good idea of how different queries affect the results returned from the database.

TRY IT OUT Selecting Data from the Sample Database

In this exercise you use the database that you connected to in an earlier exercise. This database is used only for the samples in this chapter, so don't worry if you mess things up.

1. Open the Database Explorer (or the Server Explorer in the paid versions of Visual Studio) by choosing View ➪ Database Explorer. Locate the Data Connection that you added earlier, expand it, and then expand the Tables node. You should see two tables, `Genre` and `Review`, as shown in Figure 12-5.

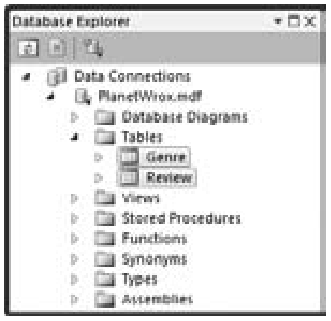
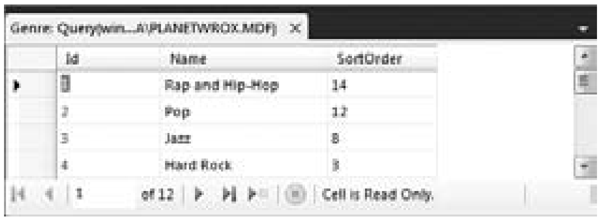


FIGURE 12-5

2. Right-click the `Genre` table and choose Show Table Data. In the Document Window you should now see a list with all the available genres in the `Genre` table, shown in Figure 12-6.



Genre: Query(win...A\PLANETWROX.MDF)

Id	Name	SortOrder
1	Rap and Hip-Hop	14
2	Pop	12
3	Jazz	8
4	Hard Rock	3

1 of 12 Cell is Read Only.

FIGURE 12-6

Note that this is not just a list with all the records in the `Genre` table. It's actually the result of a SQL `SELECT` query that is executed when you open the window. To see the query behind this list, ensure that the Query Designer toolbar, shown in Figure 12-7, is displayed onscreen. If the toolbar isn't visible, right-click an existing toolbar and choose Query Designer.

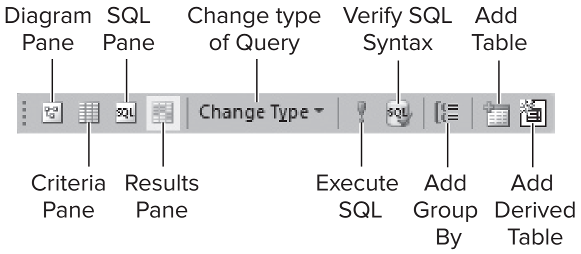
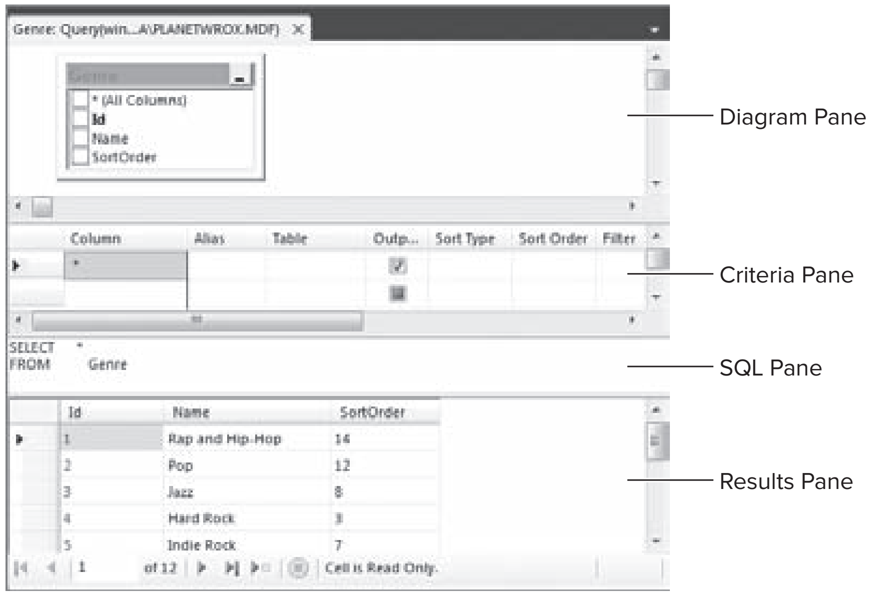


FIGURE 12-7

On this toolbar, click the Diagram pane, the Criteria pane, and the SQL pane buttons to open their respective windows. The first four buttons on the toolbar should now be in a pressed state and the Document Window is split in four regions, with each region corresponding to one of the buttons on the toolbar. Figure 12-8 shows the entire Document Window with the four panes.



Column	Alias	Table	Outp...	Sort Type	Sort Order	Filter
*			✓			

```
SELECT    *
FROM      Genre
```

Id	Name	SortOrder
1	Rap and Hip-Hop	14
2	Pop	12
3	Jazz	8
4	Hard Rock	3
5	Indie Rock	7

FIGURE 12-8

The SQL pane displays the SQL statement that is used to retrieve the genres that are displayed in the Results pane. In this case, the SQL statement reads `SELECT * FROM Genre` to retrieve all columns and records from the table, but you can easily change that.

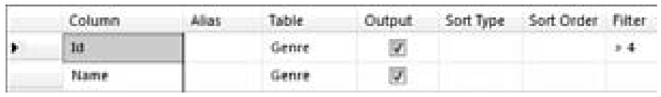
3. In the SQL pane, position your cursor right after the word `Genre`, press Enter once, and then type **`WHERE Id > 4`**. Your complete SQL statement should end up like this:

```
SELECT * FROM Genre WHERE Id > 4
```

In your SQL pane, the query is split over multiple lines. That's fine, because SQL enables you to spread your statements over multiple lines without the need for a line continuation character.

4. To make sure the SQL statement is valid, click the Verify SQL Syntax button on the toolbar and fix any errors your SQL statement may contain. Next, click the Execute SQL button (the one with the red exclamation mark on it) or press Ctrl+R. In both cases, the SQL statement is executed and the Results pane is updated to show all genres with an ID larger than 4.

5. In addition to showing the results, VWD also changed your query. Instead of `SELECT *`, it has listed each column in your table explicitly. Now take a look at the Diagram pane — the top part of the dialog box in Figure 12-8 that shows your entire table. In the Diagram pane you can check and uncheck column names to determine whether they end up in the query. Deselect the `SortOrder` column (don't accidentally change the check mark of the Output column in the Criteria pane instead). Note that it also gets removed from the Criteria pane (visible in Figure 12-9) and the SQL statement in the SQL pane.



Column	Alias	Table	Output	Sort Type	Sort Order	Filter
Id		Genre	✓			> 4
Name		Genre	✓			

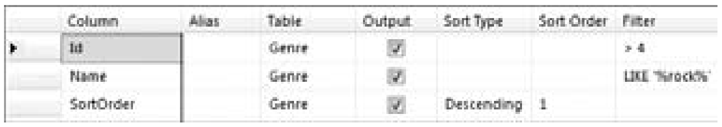
FIGURE 12-9

6. Take a look at the Criteria pane in Figure 12-9. It shows the two columns you are selecting. In the Filter column it shows the expression that filters all genres with an ID larger than 4.

   In this pane you can modify the query without manually writing a lot of code. To see how you can apply an additional filter, type **`LIKE '%rock%'`** in the Filter cell for the `Name` row. This limits the results to all genres that contain the word `rock` and that have an ID that is larger than 4. If you press Ctrl+R again the Results pane is updated to reflect the change in the query.

7. To determine the sort order, you can use the Sort Type column. You can do this for visible columns (for example, those that have their Output check box checked and end up in the final result set) but also for other columns. To order by the `SortOrder` column, click the cell under `Name` once. It changes and now shows a drop-down list instead. Choose `SortOrder` from the drop-down list. When you click or tab away from the field, VWD places a check mark in the Output column. You can click that check mark to remove the column again from the output so it remains available for ordering and filtering, but won't show up in the query results. However, for this exercise it's okay to leave the column selected.

8. In the Sort Type column choose Descending from the drop-down list for the `SortOrder`. Your final Criteria pane now looks like Figure 12-10.



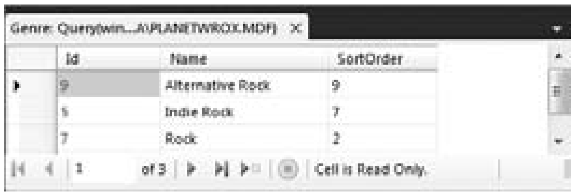
Column	Alias	Table	Output	Sort Type	Sort Order	Filter
Id		Genre	✓			> 4
Name		Genre	✓			LIKE '%rock%'
SortOrder		Genre	✓	Descending	1	

FIGURE 12-10

While you make your changes using the Diagram and Criteria panes, VWD continuously updates the SQL pane. Your final SQL statement should now include the extra `WHERE` clause and the `ORDER BY` statement:

```
SELECT Id, Name, SortOrder
FROM Genre
WHERE (Id > 4) AND (Name LIKE '%rock%')
ORDER BY SortOrder DESC
```

9. Press Ctrl+R again (or click the Execute SQL button on the toolbar) and the Results pane shows the records from the `Genre` table that match your criteria, visible in Figure 12-11.



Id	Name	SortOrder
9	Alternative Rock	9
5	Indie Rock	7
7	Rock	2

FIGURE 12-11

Note that the records are now sorted in descending order based on the `SortOrder` column.

How It Works

The Query Designer in VWD is a very helpful tool for creating new queries against your database. Instead of hand coding the entire SQL statement in the SQL pane, you use the Diagram and Criteria panes to create your queries visually. Of course, you can still use the SQL pane to make manual tweaks to the SQL code that VWD generates for you.

The final query you executed returned all the records that contain the word *rock* and that had an ID larger than 4. The query shown in step 8 has a `WHERE` clause that consists of two parts: the first part limits the records returned to those with an ID larger than 4. The second part filtered the records to those that contain the text *rock*. The two criteria are both applied at the same time using the `AND` keyword, so only records with an ID larger than 4 *and* the word *rock* in their name are returned. Effectively, this returns the Alternative Rock, Indie Rock, and Rock genres, while leaving out the Hard Rock genre because it has an ID of 4.

At the end, the result set is sorted in descending order on the `SortOrder` column using the syntax `ORDER BY SortOrder DESC`. Notice that `SortOrder` is an arbitrarily chosen name. You can easily give this column a different name, or order on a different column like the `Name` column to retrieve the genres in alphabetical order.

In this example, you saw how to retrieve data from a single table. However, in most real-world applications you get your data from multiple tables that are somehow related to each other. You define this relationship in your SQL syntax using the `JOIN` keyword.

Joining Data

A `JOIN` in your query enables you to express a relationship between one or more tables. For example, you can use a `JOIN` to find all the reviews from the `Review` table that have been published in a specific genre and then select some columns from the `Review` table together with the `Name` of the genre.

The basic syntax for a `JOIN` looks like the following bolded code:

```
SELECT
  SomeColumn
FROM
  LeftTable
INNER JOIN RightTable ON LeftTable.SomeColumn = RightTable.SomeColumn
```

The first part is the standard `SELECT` part of the query that you saw earlier, and the second part introduces the keywords `INNER JOIN` to express the relationship between the two tables. This query only returns the records in the table `LeftTable` with a corresponding record in `RightTable`. For example, to return the ID and the title of a review together with the name of the genre it belongs to, you use this SQL statement:

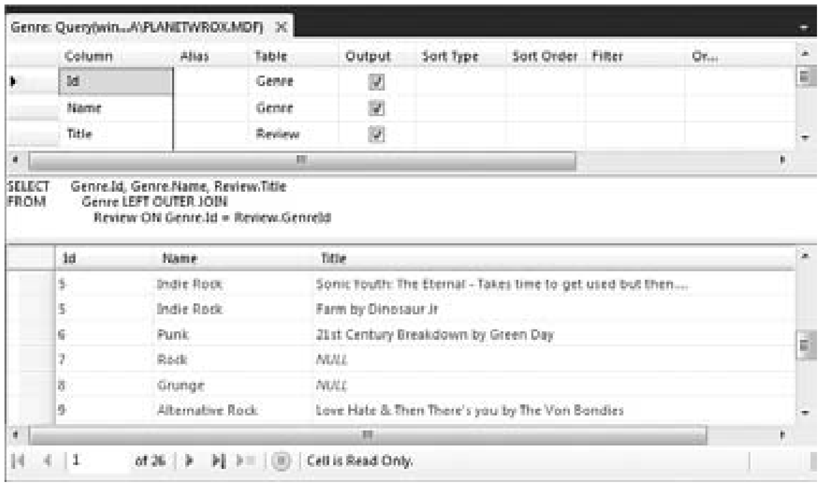
```
SELECT
  Review.Id, Review.Title, Genre.Name
FROM
  Review
INNER JOIN Genre ON Review.GenreId = Genre.Id
```

Note that in the `SELECT` statement each column is prefixed with the table name. This makes it clear what table you are referring to and avoids conflicts when multiple tables have similar column names (like the `Id` column that exists in both tables).

In addition to an `INNER JOIN` that only returns matching records, you can also use an `OUTER JOIN`. The `OUTER JOIN` enables you to retrieve records from one table regardless of whether they have a matching record in another table. The following example returns a list with all the genres in the system together with the reviews in each genre:

```
SELECT
  Genre.Id, Genre.Name, Review.Title
FROM
  Genre
LEFT OUTER JOIN Review ON Genre.Id = Review.GenreId
```

For each review assigned to a genre, a unique row is returned that contains the review's title. However, even if a genre has no reviews assigned, the row is still returned.



Column	Alias	Table	Output	Sort Type	Sort Order	Filter	Or...
Id		Genre	✓				
Name		Genre	✓				
Title		Review	✓				

```
SELECT    Genre.Id, Genre.Name, Review.Title
FROM        Genre LEFT OUTER JOIN
              Review ON Genre.Id = Review.GenreId
```

Id	Name	Title
5	Indie Rock	Sonic Youth: The Eternal - Takes time to get used but then...
5	Indie Rock	Farm by Dinosaur Jr
6	Punk	21st Century Breakdown by Green Day
7	Rock	NULL
8	Grunge	NULL
9	Alternative Rock	Love Hate & Then There's you by The Von Bondies

FIGURE 12-12

In Figure 12-12 you can see that the genre Indie Rock is repeated multiple times, for each review in the `Review` table that has been assigned to that genre. The Punk genre has only one review attached to it, so it's listed only once. Finally, the Rock and Grunge genres have no reviews associated with them. However, because the SQL statement uses a `LEFT OUTER JOIN`, those two genres (listed on the left side of the `JOIN`) are still returned. Instead of the `Title` of a review, that column now contains a `NULL` value to indicate there is no associated review.

Besides the `LEFT OUTER JOIN`, there is also a `RIGHT OUTER JOIN` that returns all the records from the table listed at the right side of the `JOIN`. `LEFT` and `RIGHT OUTER JOIN` statements are very similar, and in most cases you'll see the `LEFT OUTER JOIN`.

In addition, there are other joins including cross joins and self joins. For a detailed description of these types of joins, pick up a copy of the book *Beginning Microsoft SQL Server 2008 Programming* by Robert Vieira (ISBN: 978-0-470-25701-2).

You see how to use a very common type of join, the `INNER JOIN`, in the next Try It Out.

TRY IT OUT Joining Data

To join data from two tables, you need to write a `JOIN` statement in your code. To help you write the code, VWD adds a `JOIN` for you whenever you add a table to the Diagram pane. However, sometimes this `JOIN` is not correct, so you'll need to check the code to see if it's okay.

1. Still in your test site, on the Database Explorer (or Server Explorer), right-click the `Review` table and choose Show Table Data. You'll see all the reviews in the table appear. Next, enable the Diagram, Criteria, and SQL panes by clicking their respective buttons on the Query Designer toolbar.
2. Right-click an open spot of the Diagram pane next to the `Review` table and choose Add Table. Alternatively, choose Query Designer ➪ Add Table from the main menu.
3. In the dialog box that opens, click the `Genre` table and then click the Add button. Finally, click Close.
4. The SQL statement that VWD generated looks like this:

   ```
   SELECT * FROM Review
   INNER JOIN Genre ON Review.GenreId = Genre.Id
   ```

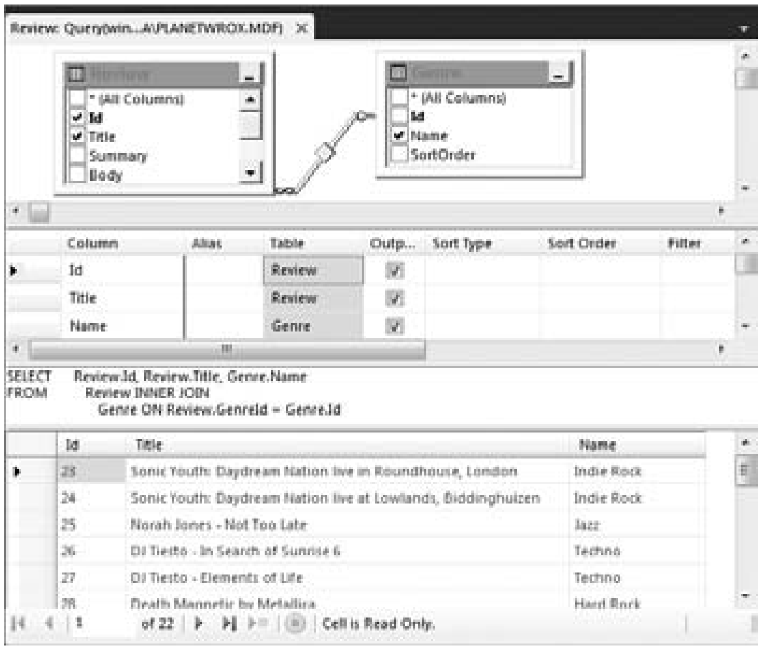
   VWD correctly detected the relationship defined in the database between the `GenreId` column of the `Review` table and the `Id` column of the `Genre` table, and applied the correct `JOIN` for you. Later you see how to define these relationships yourself.
5. To see how you can create `JOIN`s yourself without writing code directly, you'll manually recreate the `JOIN`. First, right-click the line that is drawn between the two tables in the Diagram pane and choose Remove. The SQL statement now contains a `CROSS JOIN`.
6. Next, click the `GenreId` column of the `Review` table in the Diagram pane once and drag it onto the `Id` column of the `Genre` table. As soon as you release the mouse, VWD creates a new `INNER JOIN` in the SQL pane for you with the exact same code as you saw earlier.
7. In the Criteria pane, click the left margin of the first row that contains the asterisk (`*`) symbol to select the entire row and then press the Delete key or right-click the left margin and choose Delete. This removes the asterisk from the SQL statement. Alternatively, you can delete the asterisk from the SQL pane directly.
8. In the Diagram pane place a check mark in front of the `Id` and `Title` columns of the `Review` table and in front of the `Name` column of the `Genre` table.
9. Finally, press Ctrl+R to execute the query. Your Document Window should now look like Figure 12-13, showing the results of the query at the bottom of the screen in the Results pane.

How It Works

By using a `JOIN` in your SQL statement, you tell the database how to relate records to each other. In this example, you joined the `GenreId` column of the `Review` table to the actual `Id` of the `Genre` table:

```
SELECT
  Review.Id, Review.Title, Genre.Name
FROM
  Review
INNER JOIN Genre ON Review.GenreId = Genre.Id
```

With this `JOIN`, you can retrieve data from multiple tables and present them in a single result set. SQL Server returns the correct genre name for each review, as is shown in Figure 12-13.



Column	Alias	Table	Outp...	Sort Type	Sort Order	Filter
Id		Review	✓			
Title		Review	✓			
Name		Genre	✓			

```
SELECT   Review.Id, Review.Title, Genre.Name
FROM       Review INNER JOIN
             Genre ON Review.GenreId = Genre.Id
```

Id	Title	Name
23	Sonic Youth: Daydream Nation live in Roundhouse, London	Indie Rock
24	Sonic Youth: Daydream Nation live at Lowlands, Biddinghuizen	Indie Rock
25	Norah Jones - Not Too Late	Jazz
26	DJ Tiesto - In Search of Sunrise 6	Techno
27	DJ Tiesto - Elements of Life	Techno
28	Death Magnetic by Metallica	Hard Rock

1 of 22 Cell is Read Only.

FIGURE 12-13

Besides selecting data, you also need to be able to insert data into the database. You do this with the `INSERT` statement.

Creating Data

To insert new records in a SQL Server table, you use the `INSERT` statement. It comes in a few different flavors, but in its simplest form it looks like this:

```
INSERT INTO TableName (Column1 [, Column2]) VALUES (Value1 [, Value2])
```

Just as with the `WHERE` clause, you need to enclose string and date values in single quotes, but you can enter numbers and Boolean values directly in your SQL statement. The following snippet shows how to insert a new row in the `Genre` table:

```
INSERT INTO Genre (Name, SortOrder) VALUES ('Tribal House', 20)
```

After you have created some data, you may want to edit it again. You do this with the `UPDATE` statement.

Updating Data

To update data in a table, you use the `UPDATE` statement:

```
UPDATE TableName SET Column1 = NewValue1 [, Column2 = NewValue2] WHERE
      Column3 = Value3
```

With the `UPDATE` statement, you use `Column = Value` constructs to indicate the new value of the specified column. You can have as many of these constructs as you want, with a maximum of one per column in the table. To limit the number of items that get updated, you use the `WHERE` clause, just as with selecting data as you saw earlier.

The following example updates the record that was inserted with the `INSERT` statement you saw earlier. It sets the `Name` to `Trance` and updates the `SortOrder` to `5` to move the item up a little in sorted lists. It also uses the unique ID of the new record (`13` in this example) in the `WHERE` clause to limit the number of records that get affected with the `UPDATE` statement.

```
UPDATE Genre SET Name = 'Trance', SortOrder = 5 WHERE Id = 13
```

Obviously, you may also have the need to delete existing records. It should come as no surprise that the SQL language uses the `DELETE` statement for this.

Deleting Data

Just as with the `SELECT` and `UPDATE` statements, you can use the `WHERE` clause in a `DELETE` statement to limit the number of records that get deleted. This `WHERE` clause is often very important, because you will otherwise wipe out the entire table instead of just deleting a few records.

When you write a `DELETE` statement, you don't need to specify any column names. All you need to do is indicate the table that you want to delete records from and an (optional) `WHERE` clause to limit the number of records that get deleted. The following example deletes the record that was inserted and updated in the previous two examples:

```
DELETE FROM Genre WHERE Id = 13
```

If you leave out the `WHERE` clause, all records will be deleted from the table.

You see these SQL statements at work in the next exercise.

TRY IT OUT Working with Data in the Sample Database

In this exercise, you put everything you learned so far into practice. In a series of steps, you see how to create a new record in the `Genre` table, select it again to find out its new ID, update it using the `UPDATE` statement, and finally delete the genre from the database. Although the examples themselves may seem pretty trivial, they are at the core of how SQL works. If you understand the examples from this section, you'll be able to work with the remaining SQL statements in this and coming chapters.

1. Open the Database Explorer window in your temporary test site and locate the `Genre` table in the database. Right-click it and choose Show Table Data. If the table was already open with an old query, you need to close it first by pressing Ctrl+F4. This gets rid of the existing SQL statement.

2. Click the first three buttons on the Query Designer toolbar (Diagram, Criteria, and SQL pane) to open up their respective panes.

3. In the Diagram pane, check the columns `Name` and `SortOrder`. Make sure you leave `Id` unchecked, as shown in Figure 12-14.

   Because the `Id` column gets an auto-generated value from the database, you cannot supply an explicit value for it in an `INSERT` statement.

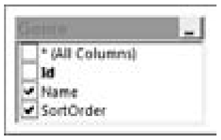


FIGURE 12-14

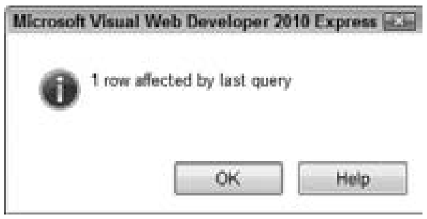
4. On the Query Designer toolbar click the Change Type button and choose the third option: Insert Values. The query in the SQL pane is updated and now contains a template for the `INSERT` statement:

```
INSERT INTO Genre (Name, SortOrder) VALUES (,)
```

5. Between the parentheses for the `VALUES`, enter a name (between apostrophes) and a sort order for your genre separated by a comma:

```
VALUES ('Folk', 15)
```

6. Press Ctrl+R to execute the query. You should get a dialog box that tells you that your action caused one row to be affected as shown in Figure 12-15.


Microsoft Visual Web Developer 2010 Express

1 row affected by last query

FIGURE 12-15

7. Click OK to dismiss the dialog box.

8. Clear out the entire SQL statement from the SQL pane (you can use Ctrl+A to select the entire SQL statement and then press the Delete key to delete it) and replace it with this code, which selects all the genres and sorts them in descending order:

```
SELECT Id, Name FROM Genre ORDER BY Id DESC
```

9. Press Ctrl+R to execute this `SELECT` statement. The Results pane shows a list of genres with the one you just inserted at the top of the list. Note the ID of the newly inserted record. It should be 13 if you haven't inserted any record before although it's okay if you have a different ID.

10. Click the Change Type button on the toolbar again, this time choosing Update. Complete the SQL statement that VWD created for you so it looks like this:

```
UPDATE
  Genre
SET
  Name = 'British Folk',
  SortOrder = 5
WHERE
  Id = 13
```

    Don't forget to replace the number 13 in the SQL statement with the ID you determined in step 9.

11. Press Ctrl+R again to execute the query and you'll get a dialog box informing you that one record has been modified.

12. Once again, clear the SQL pane and then enter and execute the following query by pressing Ctrl+R:

    ```
    SELECT Id, Name FROM Genre WHERE Id = 13
    ```

    Replace the `Id` in the `WHERE` clause with the ID of the record you determined in step 9. You should see the updated record appear.

13. On the Query Designer toolbar, click the Change Type button and choose Delete. VWD changes the SQL statement so it is now set up to delete the record with an ID of 13:

    ```
    DELETE FROM Genre WHERE (Id = 13)
    ```

14. Press Ctrl+R to execute the query and delete the record from the database. Click OK to dismiss the confirmation dialog.

15. To confirm that the record is really deleted, click the Change Type button once more and choose Select. Then choose one or more columns of the `Genre` table in the Diagram pane and press Ctrl+R again. You'll see that this time no records are returned, confirming the newly inserted genre has indeed been deleted from the database.

How It Works

In this short exercise, you carried out all four parts of the CRUD acronym, which gave you a look at the life cycle of data in a SQL Server database from creation to deletion.

You started off with an `INSERT` statement:

```
INSERT INTO Genre (Name, SortOrder) VALUES ('Folk', 15)
```

This creates a new record in the `Genre` table. As you see in the next section, the `Id` column of the `Genre` table is an *identity column*, which means that each new record gets a new, sequential ID assigned automatically.

To retrieve that ID, you used a `SELECT` statement with an `ORDER BY` clause that orders the records on their IDs in descending order, so the most recent ID was put on top of the list. Retrieving the new ID like this in a busy application is not reliable because you may end up with someone else's ID. You see later in the book how to retrieve the ID in a reliable way, but for the purposes of this exercise, the `ORDER BY` method works well enough.

Armed with the new ID, you executed an `UPDATE` statement to change the `Name` and `SortOrder` of the newly inserted genre. If you only want to update a single column with the `UPDATE` statement — say you want to change only the `Name` — you can simply leave out the other columns. For example, the following `UPDATE` statement changes only the `Name`, leaving all other columns at their original values:

```
UPDATE
  Genre
SET
  Name = 'British Folk'
WHERE
  Id = 13
```

Finally, at the end of the exercise, you executed a `DELETE` statement to get rid of the new record. It's always important to specify a `WHERE` clause when executing a `DELETE` or an `UPDATE` statement to stop you from clearing the entire table or from assigning the same value to all records.

```
DELETE FROM Genre WHERE (Id = 13)
```

This SQL statement simply deletes the record with an ID of 13. If the record exists, it gets deleted. If the record does not exist, no error is raised, but the dialog box in VWD shows you that zero records have been affected. The parentheses are not required in this example, but help in determining precedence when you have multiple conditions in your `WHERE` clause.

Up to this point, you have seen how to work with existing tables in a database. However, it's also important to understand how to create new tables with relationships yourself. This is discussed in the next section.

CREATING YOUR OWN TABLES

Creating tables in a SQL Server 2008 database is easy using the built-in database tools that are part of VWD. You see how you can create your own tables in the database after the next section, which briefly introduces you to the data types you have at your disposal in SQL Server 2008 and up.

Data Types in SQL Server

Just as with programming languages like Visual Basic .NET and C#, a SQL Server database uses different data types to store its data. SQL Server 2008 supports more than 30 different data types, most of which look similar to the types used in .NET. The following table lists the most common SQL Server data types together with a description and their .NET counterparts.

SQL 2008 DATA TYPE	DESCRIPTION	.NET DATA TYPE
`bit`	Stores Boolean values in a 0 / 1 format. (1 = True, 0 = False)	`System.Boolean`
`char / nchar`	Contains fixed-length text. When you store text shorter than the defined length, the text is padded with spaces. The `nchar` stores the data in Unicode format, which enables you to store data for many foreign languages.	`System.String`
`datetime`	Stores a date and a time.	`System.DateTime`
`datetime2`	Similar to the `datetime` type, but with a greater precision and range.	`System.DateTime`
`date`	Stores a date without the time element.	`System.DateTime`
`time`	Stores a time without the date element.	`System.TimeSpan`

continues

(continued)

SQL 2008 DATA TYPE	DESCRIPTION	.NET DATA TYPE
`decimal`	Enables you to store large, fractional numbers.	`System.Decimal`
`float`	Enables you to store large, fractional numbers.	`System.Double`
`image`	Enables you to store large binary objects such as files. Although the name suggests that you can only use it to store images, this is not the case. You can use it to store any kind of document or other binary object.	`System.Byte[]`
`tinyint`	Used to store integer numbers ranging from 0 to 255.	`System.Byte`
`smallint`	Used to store integer numbers ranging from –32,768 to 32,767.	`System.Int16`
`int`	Used to store integer numbers ranging from –2,147,483,648 to 2,147,483,647.	`System.Int32`
`bigint`	Used to store large integer numbers ranging from –9,223,372,036,854,775,808 to 9,223,372,036,854,775,807.	`System.Int64`
`text / ntext`	Used to store large amounts of text.	`System.String`
`varchar / nvarchar`	Used to store text with a variable length. The `nvarchar` stores the data in Unicode format, which enables you to store data for many foreign languages.	`System.String`
`uniqueidentifier`	Stores globally unique identifiers.	`System.Guid`

For a complete list of all the supported data types in SQL Server 2008, check out the MSDN documentation at `http://tinyurl.com/SqlDataTypes`.

Some of these data types enable you to specify the maximum length. When you define a column of type `char`, `nchar`, `varchar`, or `nvarchar` you need to specify the length in characters. For example, an `nvarchar(10)` allows you to store a maximum of 10 characters. Starting with SQL Server 2005, the version of SQL Server before SQL Server 2008, these data types also enable you to specify `MAX` as the maximum size. With the `MAX` specifier, you can store data up to 2GB in a single column. For large pieces of text, like the body of a review, you should consider the `nvarchar(max)` data type. If you have a clear idea about the maximum length for a column (like a zip code or a phone number) or you want to explicitly limit the length of it, you should specify that length instead. For example, the title of a review could be stored in an `nvarchar(200)` column to allow up to 200 characters.

Understanding Primary Keys and Identities

To uniquely identify a record in a table, you can set up a *primary key*. A primary key consists of one or more columns in a table that contains a value that is unique across all records. When you identify a column as a primary key, the database engine ensures that no two records can end up with the same value. A primary key can consist of just a single column (for example, a numeric column that contains unique numbers for each record in the table) or it can span multiple columns, where the columns together form a unique ID for the entire record.

SQL Server also supports *identity columns*. An identity column is a numeric column whose sequential values are generated automatically whenever a new record is inserted. They are often used as the primary key for a table. You see how this works in the next section when you create your own tables.

It's not a requirement to give each table a primary key, but it makes your life as a database programmer a lot easier, so it's recommended to always add one to your tables.

Creating tables, primary keys, and identity columns is really easy with VWD's database tools, as you see in the next Try It Out.

TRY IT OUT Creating Tables in the Table Designer

In this exercise you add two tables to a new database that you add to the Planet Wrox project. You should carry out the exercises in the Planet Wrox web site you have been building in the past chapters. You can close and delete the test site you created at the beginning of this chapter because you don't need it anymore. This exercise assumes you're creating tables for a local database (stored in the `App_Data` folder). If you are using a separate, remote SQL Server database you can't use VWD Express to do so. Instead, you need to get a copy of the SQL Server Management Studio Express edition that you can download from `www.microsoft.com/sql/express`.

1. Open up the Planet Wrox web site from `C:\BegASPNET\Site` in VWD, right-click the `App_Data` folder, and choose Add New Item. In the dialog box that follows, click SQL Server Database, type **`PlanetWrox.mdf`** as the name, and then click Add to add the database to your site. The Database Explorer (or Server Explorer) should open automatically showing you the new database. If it doesn't, double-click `PlanetWrox.mdf` in the Solution Explorer.
2. On the Database Explorer, right-click the Tables node and choose Add New Table, as shown in Figure 12-16.
3. In the dialog box that follows, you can enter column names and data types that together make up the table definition. Create three columns for the `Id`, `Name`, and `SortOrder` of the `Genre` table so the dialog box ends up as shown in Figure 12-17.

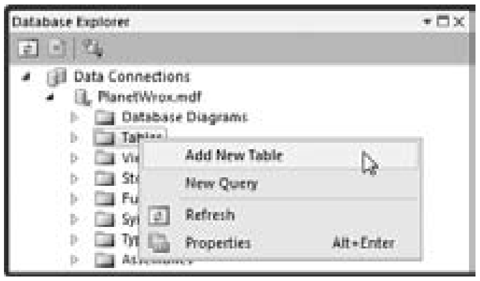
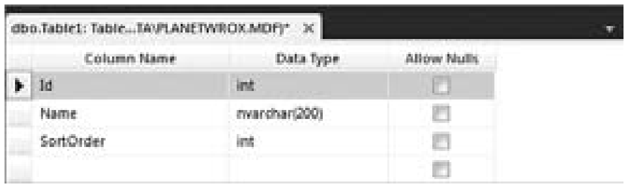


FIGURE 12-16


dbo.Table1: Table...TA\PLANETWROX.MDF)*

Column Name	Data Type	Allow Nulls
Id	int	☐
Name	nvarchar(200)	☐
SortOrder	int	☐
		☐

FIGURE 12-17

Make sure you clear the check box for all three items in the Allow Nulls column. This column determines if fields are optional or required. In the case of the `Genre` table, all three columns will be required, so you need to clear the Allow Nulls check box.

4. Next, select the entire row for the `Id` by clicking in the margin on the left (identified by the black arrow in Figure 12-17) and then on the Table Designer toolbar, visible in Figure 12-18, click the second button from the left (with the yellow key on it) to turn the `Id` column into a primary key.

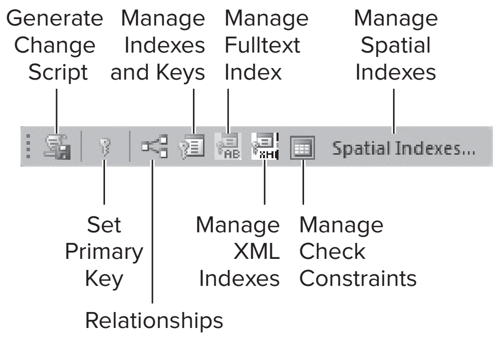
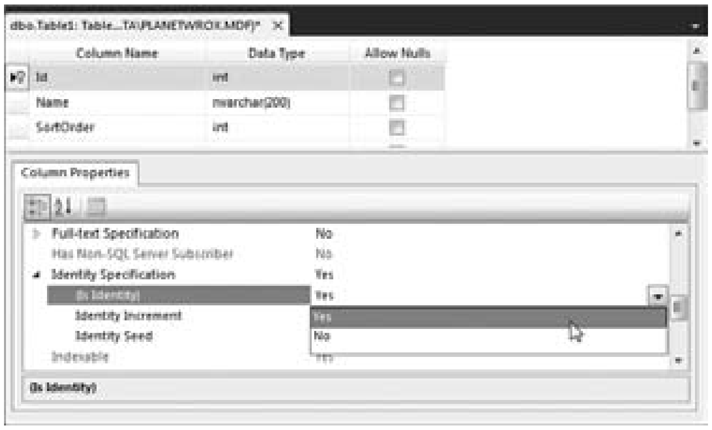


FIGURE 12-18

5. Below the table definition you see the Column Properties, a panel that looks similar to the Properties Grid in VWD. With the `Id` column still selected, scroll down a bit on the Column Properties Grid until you see Identity Specification. Expand the item and then set (Is Identity) to Yes, as shown in Figure 12-19.



dbo.Table1: Table...TA\PLANETWROX.MDF)*

Column Name	Data Type	Allow Nulls
Id	int	☐
Name	nvarchar(200)	☐
SortOrder	int	☐

Column Properties

Full-text Specification	No
Has Non-SQL Server Subscriber	No
Identity Specification	Yes
(Is Identity)	Yes
Identity Increment	Yes
Identity Seed	No
Indexable	Yes

(Is Identity)

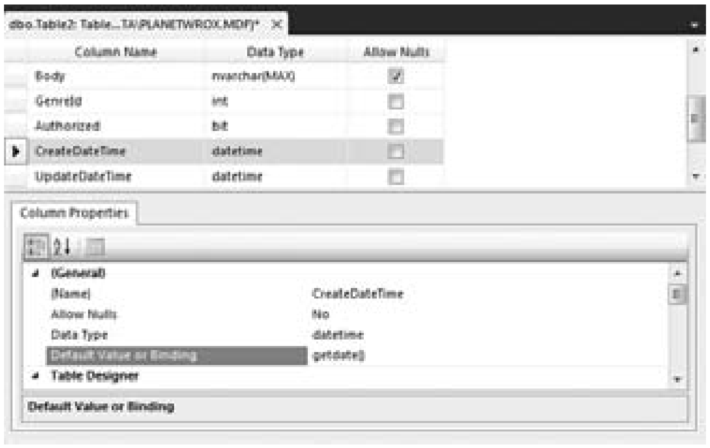
FIGURE 12-19

6. Press Ctrl+S to save your changes. A dialog box pops up that enables you to provide a name for the table. Type **`Genre`** as the name and click OK to apply your changes.

7. Create another table by following steps 2 and 3, but this time create a table with the following specifications to hold the CD and concert reviews for the Planet Wrox web site.

COLUMN NAME	DATA TYPE	ALLOW NULLS	DESCRIPTION
`Id`	`int`	No	The primary key and identity of the table.
`Title`	`nvarchar(200)`	No	Contains the title of the review.
`Summary`	`nvarchar(max)`	No	Contains a short summary or teaser text for the review.
`Body`	`nvarchar(max)`	Yes	Contains the full body text of the review.
`GenreId`	`int`	No	Contains the ID of a genre that the review belongs to.
`Authorized`	`bit`	No	Determines whether the review is authorized for publication by an administrator. Unauthorized reviews will not be visible on the web site.
`CreateDateTime`	`datetime`	No	The date and time the review is created.
`UpdateDateTime`	`datetime`	No	The date and time the review is last updated.

8. Make the `Id` column the primary key again, and set its (Is Identity) property to Yes just as you did in steps 4 and 5.
9. Click the `CreateDateTime` column once and then on the Column Properties Grid, type **`getdate()`** in the field for the Default Value or Binding property, as shown in Figure 12-20.



Column Name	Data Type	Allow Nulls
Body	nvarchar(MAX)	✓
GenreId	int	☐
Authorized	bit	☐
CreateDateTime	datetime	☐
UpdateDateTime	datetime	☐

(General)	
(Name)	CreateDateTime
Allow Nulls	No
Data Type	datetime
Default Value or Binding	getdate()

Default Value or Binding

FIGURE 12-20

10. Repeat the preceding step for the `UpdateDateTime` column.
11. When you're done, press Ctrl+S to save the table and call it `Review`.

How It Works

The Table Designer in VWD is pretty straightforward. You simply type new column names and define a data type for the column, and you're pretty much done. Some columns, such as the `Id` column in the `Genre` and `Review` tables, require a bit more work. For those columns, you set (Is Identity) to Yes. This means that SQL Server automatically assigns a new sequential number to each new record that you insert. By default, the first record in the table gets an ID of 1, and the ID of subsequent records is increased by one. You can change the default behavior by setting the Identity Increment and Identity Seed in the Identity Specification element for the column.

You also assigned a *default value* to the `CreateDateTime` and `UpdateDateTime` columns of the `Review` table. Default values are inserted by the database when you don't supply one explicitly in your SQL statements. This means that if your `INSERT` statement does not contain a value for the `CreateDateTime` or `UpdateDateTime` column, the database will insert a default value for you automatically. In the preceding Try It Out, this default value was `getdate()`, which inserts today's date and time automatically. This way, you can easily track when a review was created. In later chapters you see how to update the `UpdateDateTime` column when reviews are updated.

If you're unsure whether you followed all the steps correctly, take a look at the database that comes with the source for this chapter. It contains the correct tables that already have a few records in them. To look at the database, create a new temporary web site in VWD and then drag the database files into the `App_Data` folder. You can then access the database using the Database Explorer. You'll find a version specific to SQL Server 2005 in the `Resources` folder for this chapter as you saw earlier in this chapter.

In addition to relationships that are only defined in your own SQL queries as you saw before with the `SELECT` and `JOIN` statements, you can also create relationships in the database. The benefits of relationships and how you can create them in your database are discussed in the next section.

Creating Relationships Between Tables

Consider the tables you have created so far. You created a `Genre` table with an `Id` column to uniquely identify a genre record. You also created a `Review` table with a `GenreId` column. Clearly, this column should contain an `Id` that points to a record in the `Genre` table so you know to which genre a review belongs. Now imagine that you delete a record from the `Genre` table that has reviews attached to it. Without a relationship, the database will let you do that. However, this is causing a great deal of trouble. If you now try to display the genre together with a review, it will fail because there is no longer a matching genre. Similarly, if you want to list all the reviews in your system grouped by genre, you'll miss the ones that belong to the deleted genre.

To avoid these kinds of problems and keep your database in a healthy and consistent state, you can create a relationship between two tables. With a proper relationship set up, the database will stop you from accidentally deleting records in one table that still have other records attached to it.

Besides the protection of data, relationships also make your data model clearer. If you look at the database through a diagram (which you use in the next exercise), you'll find that relationships between tables help you better understand how tables are connected, and what data they represent.

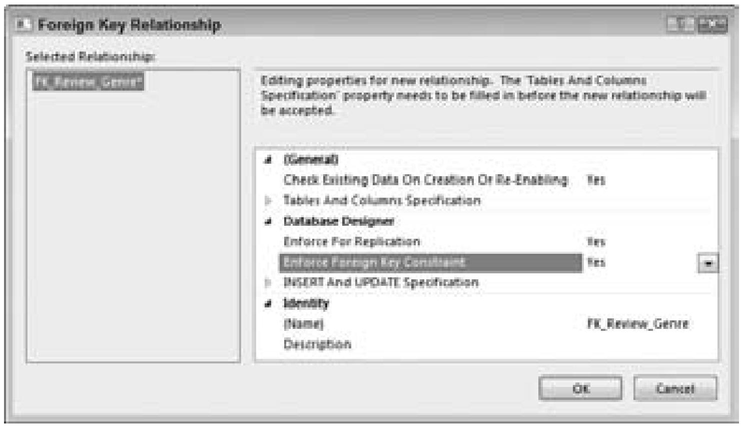
You can define a relationship by creating one between the primary key of one table, and a column in another table. The column in this second table is referred to as a *foreign key*. In the case of the

`Review` and `Genre` tables, the `GenreId` column of the `Review` table points to the primary key column `Id` of the `Genre` table, thus making `GenreId` a foreign key. In the next exercise you see how to create a relationship between two tables and then execute a SQL statement that shows how the relationship is helping you to protect your data.

TRY IT OUT Creating a Relationship between Two Tables

Before you can visually add a relationship between two tables, you need to add a diagram to your database. A diagram is a visual tool that helps you understand and define your database. On the diagram, you can drag a column from one table to another to create the relationship. In this exercise, you create a relationship between the `Review` and `Genre` tables.

1. Open the Database Explorer again for the Planet Wrox site. Right-click the Database Diagrams element (visible in Figure 12-16) and click Add New Diagram. If this is the first time you are adding a diagram to the database, you may get a dialog box asking if you want VWD to make you the owner of the database. Click Yes to proceed. Don't worry if you don't get this prompt; things will work fine without it. The prompt may be followed by another that indicates that in order to work with diagrams, VWD needs to create a few required objects. Again, click Yes to proceed.
2. In the Add Table dialog box that follows, select both tables you created in the previous Try It Out (hold down the Ctrl key while you click each item), click Add to add the tables to the diagram, and then click Close to dismiss the Add Table dialog box.
3. If necessary, arrange the tables in the diagram using drag and drop so they are positioned next to each other.
4. On the `Genre` table, click the left margin of the `Id` column (it should contain the yellow key to indicate this is the primary key of the table) and then drag it onto the `GenreId` column of the `Review` table and release your mouse.
5. Two dialog boxes pop up that enable you to customize the defaults for the relation. In the top-most window, confirm that `Id` is selected from `Genre` as the Primary Key Table and that `GenreId` is selected from `Review` as the Foreign Key Table. Click OK to dismiss the top window and confirm the columns that participate in the relationship. In the dialog box that remains, visible in Figure 12-21, notice how Enforce Foreign Key Constraint is set to Yes. This property ensures that you cannot delete a record from the `Genre` table if it still has reviews attached to it. Click OK to dismiss this dialog box as well.



Editing properties for new relationship. The 'Tables And Columns Specification' property needs to be filled in before the new relationship will be accepted.

(General)	
Check Existing Data On Creation Or Re-Enabling	Yes
Tables And Columns Specification	
Database Designer	
Enforce For Replication	Yes
Enforce Foreign Key Constraint	Yes
INSERT And UPDATE Specification	
Identity	
(Name)	FK_Review_Genre
Description	

FIGURE 12-21

6. The diagram window should now show a line between the two tables. At the side of the `Genre` table, you should see a yellow key to indicate this table contains the primary key for the relationship. At the other end, you should see the infinity symbol (the number 8 turned 90 degrees) to indicate that the `Review` table can have many records that use the same `GenreId`. You see the diagram in Figure 12-22.

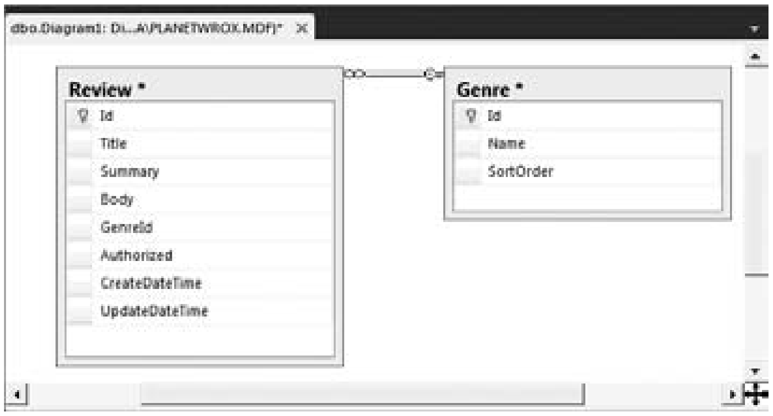
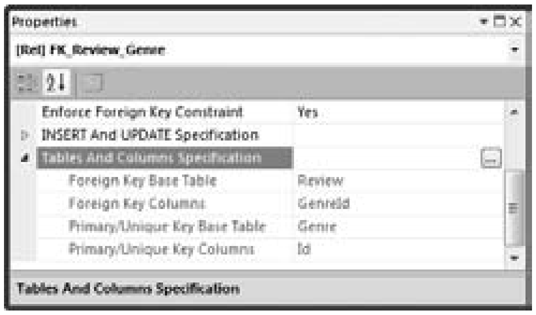


FIGURE 12-22

Note that the line between the two tables doesn't necessarily point to the correct columns. This can be confusing sometimes because you may think that other columns are actually related. To confirm the columns participating in the relationship, right-click the line between the two tables and choose Properties. The Table and Columns Specification item shows which columns and tables participate in the relationship, shown in Figure 12-23.


Properties

[Ref] FK_Review_Genre

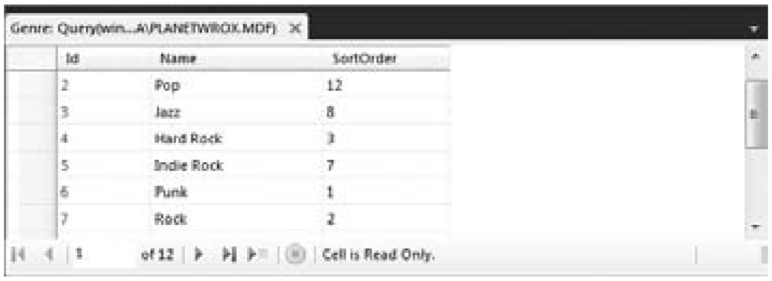
Enforce Foreign Key Constraint	Yes
INSERT And UPDATE Specification	
Tables And Columns Specification	
Foreign Key Base Table	Review
Foreign Key Columns	GenreId
Primary/Unique Key Base Table	Genre
Primary/Unique Key Columns	Id

Tables And Columns Specification

FIGURE 12-23

7. Press Ctrl+S to save the changes to the diagram. You can leave the name set to its default of Diagram1 or you can enter a more descriptive name such as **`Reviews and Genres`** and click OK. You'll get another warning that states that you are about to make changes to the `Review` and `Genre` tables. Click Yes to apply the changes.

8. Go back to the Database Explorer, right-click the `Genre` table, and choose Show Table Data. Enter a few different genres by typing a `Name` and a `SortOrder`. When you press Tab in the `SortOrder` field to tab away from the current row, the row is inserted in the database, and the `Id` column is filled with a unique, sequential number. You should end up with a list similar to the one shown in Figure 12-24.



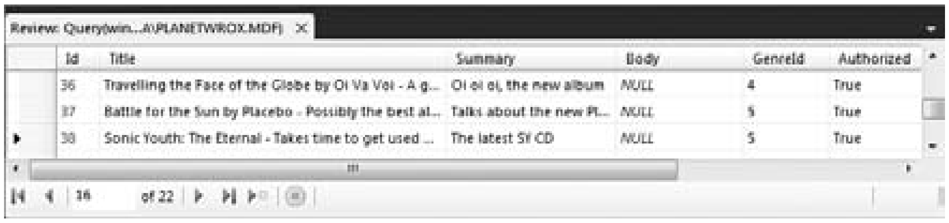
Genre: Query(win...A\PLANETWROX.MDF)

Id	Name	SortOrder
2	Pop	12
3	Jazz	8
4	Hard Rock	3
5	Indie Rock	7
6	Punk	1
7	Rock	2

1 of 12 | Cell is Read Only.

FIGURE 12-24

9. Open the `Review` table from the Database Explorer using the Show Table Data command and enter a few review records. For the `GenreId`, supply some of the new IDs you got when you inserted records in the `Genre` table. You can just make up the Title, Summary, and Body fields for now and set Authorized to True. Remember, you don't have to enter a value for the date columns. If you leave them out, the database will insert the default value for you. Notice that you can't insert a value in the `Id` column yourself. Because this column is an Identity field, the database supplies values for you automatically. If you get an error about missing values for the date columns, ensure that you entered a proper default value in the previous exercise. When you're done entering a row, click outside the row (on the new, empty row below it, for example) and press Ctrl+R to insert the row in the table. Your list of records should look similar to Figure 12-25, although your content for the columns, of course, may be different.



Review: Query(win...A\PLANETWROX.MDF)

Id	Title	Summary	Body	GenreId	Authorized
36	Travelling the Face of the Globe by Oi Va Voi - A g...	Oi oi oi, the new album	NULL	4	True
37	Battle for the Sun by Placebo - Possibly the best al...	Talks about the new Pl...	NULL	5	True
38	Sonic Youth: The Eternal - Takes time to get used ...	The latest SY CD	NULL	5	True

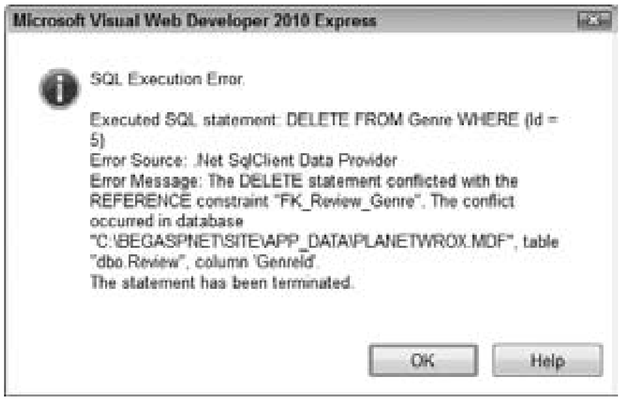
16 of 22

FIGURE 12-25

10. Right-click the `Genre` table again and choose Show Table Data. Click the SQL pane button on the Query Designer toolbar and then use the Change Type button on the same toolbar to create a `DELETE` query. Modify the query so it looks like this:

```
DELETE FROM Genre WHERE Id = 5
```

This code will attempt to delete the Indie Rock genre. However, because reviews are connected to it, the delete action should fail. Make sure that the `Id` in the `WHERE` clause



Microsoft Visual Web Developer 2010 Express

SQL Execution Error.

Executed SQL statement: DELETE FROM Genre WHERE (Id = 5)
Error Source: .Net SqlClient Data Provider
Error Message: The DELETE statement conflicted with the REFERENCE constraint "FK_Review_Genre". The conflict occurred in database "C:\BEGASPNET\SITE\APP_DATA\PLANETWROX.MDF", table "dbo.Review", column 'GenreId'.
The statement has been terminated.

FIGURE 12-26

matches one of the genre IDs you used in step 9 to link the reviews to. Press Ctrl+R to execute the query. Instead of deleting the record from the `Genre` table, VWD now shows you the dialog box you see Figure 12-26.

How It Works

When you create a relationship between two tables, the database will enforce this relationship when you try to insert, update, or delete data. In this example, records in the `Review` table have a genre that exists in the `Genre` table. When you try to delete a record from the `Genre` table, the database sees that the genre is used by a record in the `Review` table and cancels the delete operation. In Chapter 15 you learn how to handle this situation in your web site and present your user with a friendly error message.

Now that you've seen the underlying concepts in dealing with databases, you're ready for the next chapter, which shows you how to work with your database using the many available ASP.NET data controls.

PRACTICAL DATABASE TIPS

The following list provides some practical tips on working with databases:

- Because the database is often at the heart of a web site, you need to carefully consider its design. It's especially important to think of a good design up front, before you start building your site on top of it. When you have a number of pages that access your database, it will become harder to make changes — such as removing tables or renaming columns — to the data model.
- Always consider the primary key for your table. I prefer to give each table a column called `Id`. The underlying data type is then an `int` and an identity, which gives each record a unique ID automatically. Instead of an `int`, you can also consider the `uniqueidentifier` data type, which ensures uniqueness even across database or application boundaries.
- Give your database objects such as tables and columns logical names. Avoid characters such as spaces, underscores, and dashes. A name like `GenreId` is much easier to read than `colGen_ID_3`.
- Don't use `SELECT *` to get all columns from a database. By using `SELECT *` you may be selecting more columns that you actually need. By explicitly defining the columns you want to retrieve, you make your intentions to others clearer and increase the performance of your queries at the same time.
- Always create relationships between tables when appropriate. Although querying for the reviews and genres you saw in this chapter without a relationship between the two tables works just fine, relationships help you enforce the quality of your data. With proper relationships, you minimize the chance of ending up with orphaned or incorrect data.

SUMMARY

The ability to work with databases is a good addition to your set of web development skills. Most of today's dynamic web sites use databases, so it's important to understand how to work with them.

To access and manipulate data in a relational database, you use a language called Structured Query Language, or SQL for short. Among other elements, this language defines four important keywords that enable you to perform CRUD — Create, Read, Update, Delete — operations against a database.

The `SELECT` statement enables you to retrieve data from one or more tables. To access more than one table, you can use one of the available `JOIN` types to define a relationship between the tables. To limit the number of records returned by a query, you can use a `WHERE` clause. To order the items in the result set returned by your query, you use the `ORDER BY` clause. To create new records in your database you use the `INSERT` statement, and you need an `UPDATE` statement to change existing records. Finally, to delete records that you no longer need, you use the `DELETE` statement. Just like the `SELECT` and `UPDATE` statements, `DELETE` takes an optional `WHERE` clause that enables you to limit the number of records that get deleted.

The second part of this chapter showed you how to use the built-in database tools to create tables with relationships between them. In addition, you saw how a relationship between two tables enables you to protect your data from becoming corrupt or orphaned.

Although this chapter had a strong focus on the SQL that you need to write to access a database, you see in the next chapter that in many cases VWD makes accessing databases pretty easy by generating most of the code for you. However, a solid knowledge of SQL helps you in understanding and tweaking the code that VWD writes for you.

EXERCISES

1. If you try to delete a record from the `Genre` table that has matching records in the `Review` table, the `DELETE` statement fails. How is this possible?

2. If you try to delete a record from the `Review` table that has its `GenreId` set to the `Id` of an existing genre in the `Genre` table, the `DELETE` statement succeeds. Why?

3. Imagine you want to clean up your database and decide to delete all records from the `Review` table that have an `Id` of 100 or less. Write a SQL statement that deletes these records.

4. Imagine you want to delete the genre with an ID of 4. But before you delete the genre, you want to reassign reviews assigned to this genre to another genre with an ID of 11. What SQL statements do you need to accomplish this?

5. Write a SQL statement that updates the Rock genre to read Punk Rock instead. There are at least two ways to write the `WHERE` clause for this statement.

Answers to Exercises can be found in Appendix A.

▶ WHAT YOU LEARNED IN THIS CHAPTER

CRUD	The four basic SQL operations to work with data in a database: Create, Read, Update, and Delete
Foreign key	Identifies a column in a table that refers to the primary key of another table to enforce referential integrity
Identity	An automatic, sequential number assigned to new records
`JOIN`	Enables you to express the relationship between two or more tables in a query to find related data
Primary key	Consists of one or more columns in a table that uniquely identify a record in that table
Relational database	A type of database where data is stored in separate, spreadsheet-like tables that can refer to each other
Relationship	Defines the relation between one or more tables and helps you enforce referential integrity
Table	An object in a database that enables you to store data