

18

Database: SQL, MySQL, LINQ and Java DB

Chapter 18

Database: SQL, MySQL and Java DB

18.1 Introduction

- ▶ A database is an organized collection of data.
- ▶ A database management system (DBMS) provides mechanisms for storing, organizing, retrieving and modifying data for many users.
- ▶ Today's most popular database management systems are *relational databases*.
- ▶ SQL is the international standard language used almost universally with relational database systems to perform queries and manipulate data.
- ▶ Programs connect to, and interact with, relational databases systems via an interface—software that facilitates communications between a database management system and a program.

18.2 Relational Databases

- ▶ A relational database stores data in tables. Tables are composed of rows, and rows are composed of columns in which values are stored.
- ▶ A primary key provides unique values that cannot be duplicated in other rows of the same table.
- ▶ Each column of a table represents a different attribute in a row of data.
- ▶ The primary key is a column (or group of columns) with a *unique* value that cannot be duplicated in other rows.

	Number	Name	Department	Salary	Location
Row	23603	Jones	413	1100	New Jersey
	24568	Kerwin	413	2000	New Jersey
	34589	Larson	642	1800	Los Angeles
	35761	Myers	611	1400	Orlando
	47132	Neumann	413	9000	New Jersey
	78321	Stephens	611	8500	Orlando

Primary key Column

Fig. 18.1 | Employee table sample data.

Department	Location
413	New Jersey
611	Orlando
642	Los Angeles

Fig. 18.2 | Result of selecting distinct Department and Location data from table Employee.

18.3 Relational Database Overview: A books Database

- ▶ The combined columns of the AuthorISBN table represent the table's *primary key*—thus, each row in this table must be a *unique* combination of an AuthorID and an ISBN.
- ▶ This is known as the Rule of Entity Integrity.
- ▶ A one-to-many relationship between tables indicates that a row in one table can have many related rows in a separate table.
- ▶ A foreign key is a column in a table that matches the primary-key column in another table.
- ▶ The foreign key helps maintain the Rule of Referential Integrity: Every foreign-key value must appear as another table's primary-key value.

18.3 Relational Database Overview: A books Database (Cont.)

- ▶ Foreign keys also allow related data in multiple tables to be selected from those tables for analytic purposes—this is known as joining the data.

Column	Description
AuthorID	Author's ID number in the database. In the <code>books</code> database, this integer column is defined as <code>autoincremented</code> —for each row inserted in this table, the <code>AuthorID</code> value is increased by 1 automatically to ensure that each row has a unique <code>AuthorID</code> . This column represents the table's primary key.
FirstName	Author's first name (a string).
LastName	Author's last name (a string).

Fig. 18.3 | Authors table from the books database.

AuthorID	FirstName	LastName
1	Paul	Deitel
2	Harvey	Deitel
3	Abbey	Deitel
4	Michael	Morgan
5	Eric	Kern

Fig. 18.4 | Sample data from the Authors table.

Column	Description
AuthorID	The author's ID number, a foreign key to the Authors table.
ISBN	The ISBN for a book, a foreign key to the Titles table.

Fig. 18.5 | AuthorISBN table from the books database.

AuthorID	ISBN	AuthorID	ISBN
1	0132152134	2	0132575663
2	0132152134	1	0132662361
1	0132151421	2	0132662361
2	0132151421	1	0132404168
1	0132575663	2	0132404168
1	013705842X	1	0132121360
2	013705842X	2	0132121360
3	013705842X	3	0132121360
4	013705842X	4	0132121360
5	013705842X		

Fig. 18.6 | Sample data from the AuthorISBN table of books.

Column	Description
ISBN	ISBN of the book (a string). The table's primary key. ISBN is an abbreviation for "International Standard Book Number"—a numbering scheme that publishers use to give every book a unique identification number.
Title	Title of the book (a string).
EditionNumber	Edition number of the book (an integer).
Copyright	Copyright year of the book (a string).

Fig. 18.7 | Titles table from the books database.

ISBN	Title	EditionNumber	Copyright
0132152134	Visual Basic 2010 How to Program	5	2011
0132151421	Visual C# 2010 How to Program	4	2011
0132575663	Java How to Program	9	2012
0132662361	C++ How to Program	8	2012
0132404168	C How to Program	6	2010
013705842X	iPhone for Programmers: An App-Driven Approach	1	2010
0132121360	Android for Programmers: An App-Driven Approach	1	2012

Fig. 18.8 | Sample data from the Titles table of the books database .

18.3 Relational Database Overview: A books Database (Cont.)

- ▶ An entity–relationship (ER) diagram shows the *database tables* and the *relationships* among them.
- ▶ Every row must have a primary–key value, and that value must be unique in the table. This is known as the Rule of Entity Integrity.
- ▶ An infinity symbol (∞) indicates a one–to–many relationship, in which an entry from a table can have an arbitrary number of entries in another table.
- ▶ A *many–to–many* relationship indicates that multiple entries can be related between tables.



Fig. 18.9 | Table relationships in the books database.



Common Programming Error 18.1

Not providing a value for every column in a primary key breaks the Rule of Entity Integrity and causes the DBMS to report an error.

18.4 SQL

- ▶ The next several subsections discuss the keywords listed in the following slide in the context of SQL queries and statements.

SQL keyword	Description
SELECT	Retrieves data from one or more tables.
FROM	Tables involved in the query. Required in every SELECT.
WHERE	Criteria for selection that determine the rows to be retrieved, deleted or updated. Optional in a SQL query or a SQL statement.
GROUP BY	Criteria for grouping rows. Optional in a SELECT query.
ORDER BY	Criteria for ordering rows. Optional in a SELECT query.
INNER JOIN	Merge rows from multiple tables.
INSERT	Insert rows into a specified table.
UPDATE	Update rows in a specified table.
DELETE	Delete rows from a specified table.

Fig. 18.10 | SQL query keywords.

18.4.1 Basic SELECT Query

- ▶ The basic form of a SELECT query is

SELECT * FROM *tableName*

in which the asterisk (*) *wildcard character* indicates that all columns from the *tableName* should be retrieved.

- ▶ To retrieve specific columns, replace the * with a comma-separated list of column names.

AuthorID	LastName
1	Deitel
2	Deitel
3	Deitel
4	Morgan
5	Kern

Fig. 18.11 | Sample AuthorID and LastName data from the Authors table.

18.4.2 WHERE Clause

- ▶ SQL uses the optional WHERE clause in a query to specify the selection criteria for the query. The basic form of a query with selection criteria is

```
SELECT columnName1, columnName2, ... FROM  
      tableName WHERE criteria
```

- ▶ Strings in SQL are delimited by single ('') rather than double ("") quotes.

Title	EditionNumber	Copyright
Visual Basic 2010 How to Program	5	2011
Visual C# 2010 How to Program	4	2011
Java How to Program	9	2012
C++ How to Program	8	2012
Android for Programmers: An App-Driven Approach	1	2012

Fig. 18.12 | Sampling of titles with copyrights after 2005 from table Titles.

18.4.2 WHERE Clause (Cont.)

Pattern Matching: Zero or More Characters

- ▶ The WHERE clause can contain operators <, >, <=, >=, =, <> and LIKE. Operator LIKE is used for string pattern matching with wildcard characters percent (%) and underscore (_).
- ▶ A percent character (%) in a pattern indicates that a string matching the pattern can have zero or more characters at the percent character's location in the pattern.
- ▶ An underscore (_) in the pattern string indicates a single character at that position in the pattern.

AuthorID	FirstName	LastName
1	Paul	Deitel
2	Harvey	Deitel
3	Abbey	Deitel

Fig. 18.13 | Authors whose last name starts with D from the Authors table.

18.4.2 WHERE Clause (Cont.)

Pattern Matching: Any Character

- ▶ An underscore (_) in the pattern string indicates a single wildcard character at that position in the pattern.
- ▶ The following query locates the rows of all the authors whose last names start with any character (specified by _), followed by the letter o, followed by any number of additional characters (specified by %):

```
SELECT AuthorID, FirstName, LastName  
      FROM Authors  
     WHERE LastName LIKE '_o%'
```

- ▶ The preceding query produces the row shown in the next slide, because only one author in our database has a last name that contains the letter o as its second letter.

AuthorID	FirstName	LastName
4	Michael	Morgano

Fig. 18.14 | The only author from the Authors table whose last name contains o as the second letter.

18.4.3 ORDER BY Clause

- ▶ The result of a query can be sorted in ascending or descending order using the optional ORDER BY clause. The simplest form of an ORDER BY clause is

```
SELECT columnName1, columnName2, ... FROM  
      tableName ORDER BY column ASC
```

```
SELECT columnName1, columnName2, ... FROM  
      tableName ORDER BY column DESC
```

where ASC specifies ascending order, DESC specifies descending order and column specifies the column on which the sort is based. The default sorting order is ascending, so ASC is optional.

- ▶ Multiple columns can be used for ordering purposes with an ORDER BY clause of the form

```
ORDER BY column1 sortingOrder, column2  
          sortingOrder, ...
```

- ▶ The WHERE and ORDER BY clauses can be combined in one query. If used, ORDER BY must be the last clause in the query.

AuthorID	FirstName	LastName
1	Paul	Deitel
2	Harvey	Deitel
3	Abbey	Deitel
5	Eric	Kern
4	Michael	Morgano

Fig. 18.15 | Sample data from table Authors in ascending order by LastName.

AuthorID	FirstName	LastName
4	Michael	Morgano
5	Eric	Kern
1	Paul	Deitel
2	Harvey	Deitel
3	Abbey	Deitel

Fig. 18.16 | Sample data from table Authors in descending order by LastName.

AuthorID	FirstName	LastName
3	Abbey	Deitel
2	Harvey	Deitel
1	Paul	Deitel
5	Eric	Kern
4	Michael	Morgano

Fig. 18.17 | Sample data from Authors in ascending order by LastName and FirstName.

ISBN	Title	Editio n- Number	Copy- right
0132404168	C How to Program	6	2010
0132662361	C++ How to Program	8	2012
0132575663	Java How to Program	9	2012
0132152134	Visual Basic 2005 How to Program	5	2011
0132151421	Visual C# 2005 How to Program	4	2011

Fig. 18.18 | Sampling of books from table Titles whose titles end with How to Program in ascending order by Title.

18.4.4 Merging Data from Multiple Tables: INNER JOIN

- ▶ An INNER JOIN operator merges rows from two tables by matching values in columns that are common to the tables. The basic form for the INNER JOIN operator is:

```
SELECT columnName1, columnName2, ...
FROM table1
INNER JOIN table2
    ON table1.columnName = table2.columnName
```

- ▶ The ON clause of the INNER JOIN specifies the columns from each table that are compared to determine which rows are merged.
- ▶ The following query produces a list of authors accompanied by the ISBNs for books written by each author:

```
SELECT FirstName, LastName, ISBN
FROM Authors
INNER JOIN AuthorISBN
    ON Authors.AuthorID = AuthorISBN.AuthorID
ORDER BY LastName, FirstName
```

18.4.4 Merging Data from Multiple Tables: INNER JOIN (Cont.)

- ▶ Note the use of the syntax *tableName. columnName* in the ON clause. This syntax, called a qualified name, specifies the columns from each table that should be compared to join the tables.
- ▶ The “*tableName.*” syntax is required if the columns have the same name in both tables. The same syntax can be used in any SQL statement to distinguish columns in different tables that have the same name.
- ▶ In some systems, table names qualified with the database name can be used to perform cross-database queries. As always, the query can contain an ORDER BY clause.

FirstName	LastName	ISBN	FirstName	LastName	ISBN
Abbey	Deitel	013705842X	Paul	Deitel	0132151421
Abbey	Deitel	0132121360	Paul	Deitel	0132575663
Harvey	Deitel	0132152134	Paul	Deitel	0132662361
Harvey	Deitel	0132151421	Paul	Deitel	0132404168
Harvey	Deitel	0132575663	Paul	Deitel	013705842X
Harvey	Deitel	0132662361	Paul	Deitel	0132121360
Harvey	Deitel	0132404168	Eric	Kern	013705842X
Harvey	Deitel	013705842X	Michael	Morgano	013705842X
Harvey	Deitel	0132121360	Michael	Morgano	0132121360
Paul	Deitel	0132152134			

Fig. 18.19 | Sampling of authors and ISBNs for the books they have written in ascending order by LastName and FirstName.

18.4.5 INSERT Statement

- ▶ An INSERT statement inserts a new row into a table. The basic form of this statement is

```
INSERT INTO tableName ( columnName1,  
                         columnName2, ..., columnNameN )  
        VALUES ( value1, value2, ..., valueN )
```

where *tableName* is the table in which to insert the row. The *tableName* is followed by a comma-separated list of column names in parentheses. The list of column names is followed by the SQL keyword VALUES and a comma-separated list of values in parentheses.

- ▶ Always explicitly list the columns when inserting rows. If the table's column order changes or a new column is added, using only VALUES may cause an error.

18.4.5 INSERT Statement (Cont.)

- ▶ The INSERT statement

```
INSERT INTO Authors ( FirstName, LastName )
    VALUES ( 'Sue', 'Red' )
```

inserts a row into the Authors table.

- ▶ The statement indicates that values are provided for the FirstName and LastName columns.
- ▶ The corresponding values are 'Sue' and 'Red'.
- ▶ We do not specify an AuthorID in this example because AuthorID is an autoincremented column.
- ▶ For every row added to this table, the DBMS assigns a unique AuthorID value that is the next value in the autoincremented sequence (i.e., 1, 2, 3 and so on). In this case, Sue Red would be assigned AuthorID number 6.

AuthorID	FirstName	LastName
1	Paul	Deitel
2	Harvey	Deitcl
3	Abbey	Deitel
4	Michael	Morgano
5	Eric	Kern
6	Sue	Red

Fig. 18.20 | Sample data from table Authors after an INSERT operation.

18.4.6 UPDATE Statement

- An UPDATE statement modifies data in a table. Its basic form is

```
UPDATE tableName
      SET columnName1 = value1,
          columnName2 = value2, ...,
          columnNameN = valueN
      WHERE criteria
```

where *tableName* is the table in which to update data. The *tableName* is followed by keyword SET and a comma-separated list of column name/value pairs in the format *columnName* = *value*. The optional WHERE clause *criteria* determines which rows to update.

AuthorID	FirstName	LastName
1	Paul	Deitel
2	Harvey	Deitel
3	Abbey	Deitel
4	Michael	Morgano
5	Eric	Kern
6	Sue	Black

Fig. 18.21 | Sample data from table Authors after an UPDATE operation.

18.4.7 DELETE Statement

- ▶ A SQL DELETE statement removes rows from a table. Its basic form is

```
DELETE FROM tableName WHERE  
      criteria
```

where *tableName* is the table from which to delete a row (or rows). The optional WHERE *criteria* determines which rows to delete. If this clause is omitted, all the table's rows are deleted.

AuthorID	FirstName	LastName
1	Paul	Deitel
2	Harvey	Deitel
3	Abbey	Deitel
4	Michael	Morgano
5	Eric	Kern

Fig. 18.22 | Sample data from table Authors after a DELETE operation.

18.5 MySQL

- ▶ MySQL (pronounced “my sequel”) is a robust and scalable relational database management system (RDBMS) that was created by the Swedish consulting firm TcX in 1994.
- ▶ MySQL, now owned by Oracle, is a multiuser, multithreaded RDBMS server that uses SQL to interact with and manipulate data.
- ▶ Multithreading capabilities enable MySQL database to perform multiple tasks concurrently, allowing the server to process client requests efficiently.
- ▶ Implementations of MySQL are available for Windows, Mac OS X, Linux and UNIX.

18.6 (Optional) Microsoft Language Integrated Query (LINQ)

- ▶ The next several sections introduce C#'s LINQ (Language Integrated Query) capabilities.
- ▶ LINQ allows you to write query expressions, similar to SQL queries, that retrieve information from a wide variety of data sources, not just databases.
- ▶ We use LINQ to Objects in this section to query arrays and Lists, selecting elements that satisfy a set of conditions—this is known as filtering.

18.6.1 Querying an Array of int Values Using LINQ

- ▶ Repetition statements that filter arrays focus on the process of getting the results—iterating through the elements and checking whether they satisfy the desired criteria.
- ▶ LINQ specifies the conditions that selected elements must satisfy. This is known as **declarative programming**—as opposed to **imperative programming** (which we've been doing so far) in which you specify the actual steps to perform a task.

18.6.1 Querying an Array of int Values Using LINQ (Cont.)

- ▶ The next several statements assume that the integer array
`int[] values = { 2, 9, 5, 0, 3, 7, 1, 4, 8, 5 };`

is declared. The query

```
var filtered =
    from value in values
    where value > 4
    select value;
```

specifies that the results should consist of all the ints in the values array that are greater than 4 (i.e., 9, 5, 7, 8 and 5). It *does not* specify *how* those results are obtained—the C# compiler generates all the necessary code automatically, which is one of the great strengths of LINQ.

18.6.1 Querying an Array of int Values Using LINQ (Cont.)

The from Clause and Implicitly Typed Local Variables

- ▶ A LINQ query begins with a `from` clause, which specifies a range variable (`value`) and the data source to query (`values`).
- ▶ The range variable represents each item in the data source (one at a time), much like the control variable in a `foreach` statement. We do not specify the range variable's type.
- ▶ Implicitly typed local variables enable the compiler to *infer* a local variable's type based on the context in which it's used.

18.6.1 Querying an Array of int Values Using LINQ (Cont.)

The var Keyword and Implicitly Typed Local Variables

- ▶ You can also declare a local variable and let the compiler infer the variable's type based on the variable's initializer. To do so, the **var keyword** is used in place of the variable's type when declaring the variable. Consider the declaration

```
var x = 7;
```

- ▶ Here, the compiler *infers* that the variable x should be of type **int**, because the compiler assumes that whole-number values, like 7, are of type **int**.
- ▶ Similarly, in the declaration

```
var y = -123.45;
```

the compiler infers that y should be of type **double**, because the compiler assumes that floating-point number values, like -123.45, are of type **double**

18.6.1 Querying an Array of int Values Using LINQ (Cont.)

The where Clause

- ▶ If the condition in the where clause evaluates to true, the element is *selected*—i.e., it's included in the results.
- ▶ An expression that takes an element of a collection and returns true or false by testing a condition on that element is known as a predicate.

The select Clause

- ▶ For each item in the data source, the select clause determines what value appears in the results. A LINQ query typically ends with a select clause.

18.6.1 Querying an Array of int Values Using LINQ (Cont.)

Iterating Through the Results of the LINQ Query

- ▶ The foreach statement

```
foreach ( var element in filtered )
    Console.WriteLine( " {0}" , element );
```

displays the query results.

- ▶ A foreach statement can iterate through the contents of an array, collection or the results of a LINQ query, allowing you to process each element in the array, collection or query.
- ▶ The preceding foreach statement iterates over the query result filtered, displaying each of its items.

18.6.1 Querying an Array of int Values Using LINQ (Cont.)

The orderby Clause

- ▶ The orderby clause sorts the query results in ascending order. The query

```
var sorted =  
    from value in values  
    orderby value  
    select value;
```

sorts the integers in array values into ascending order and assigns the results to variable sorted.

To sort in descending order, use descending in the orderby clause, as in

```
orderby value descending
```

- ▶ An ascending modifier also exists but isn't normally used, because it's the default.

18.6.1 Querying an Array of int Values Using LINQ (Cont.)

- ▶ The following two queries generate the same results, but in different ways:

```
var sortFilteredResults =  
    from value in filtered  
    orderby value descending  
    select value;  
  
var sortAndFilter =  
    from value in values  
    where value > 4  
    orderby value descending  
    select value;
```

18.6.1 Querying an Array of int Values Using LINQ (Cont.)

- ▶ The first uses LINQ to sort the results of the filtered query presented earlier in this section.
- ▶ The second query uses both the where and orderby clauses.

An Aside: Interface `IEnumerable<T>`

- ▶ `foreach` iterates over any so-called `IEnumerable<T>` object, which just happens to be what a LINQ query returns.
`IEnumerable<T>` is an interface that describes the functionality of any object that can be iterated over and thus offers methods to access each element.

18.6.2 Querying an Array of Employee Objects Using LINQ

- ▶ LINQ can be used with most data types, including strings and user-defined classes.
- ▶ The query result's `Any` method returns `true` if there's at least one element, and `false` if there are no elements.
- ▶ The query result's `First` method returns the first element in the result.
- ▶ You should check that the query result is not empty before calling `First`.
- ▶ LINQ defines many more extension methods, such as `Count`, which returns the number of elements in the results.
- ▶ The `Distinct` extension method removes duplicate elements, causing all elements in the result to be unique.

18.6.2 Querying an Array of Employee Objects Using LINQ

- ▶ The syntax

```
new { e.FirstName, Last = e.LastName }
```

creates a new object of an anonymous type (a type with no name), which the compiler generates for you based on the properties listed in the curly braces ({}).

- ▶ In this case, the anonymous type consists of properties for the first and last names of the selected Employee.
- ▶ The LastName property is assigned to the property Last in the select clause.
- ▶ This is an example of a projection—it performs a transformation on the data.

```
1 // Fig. 18.23: Employee.cs
2 // Employee class with FirstName, LastName and MonthlySalary properties.
3 public class Employee
4 {
5     private decimal monthlySalaryValue; // monthly salary of employee
6
7     // auto-implemented property FirstName
8     public string FirstName { get; set; }
9
10    // auto-implemented property LastName
11    public string LastName { get; set; }
12
13    // constructor initializes first name, last name and monthly salary
14    public Employee( string first, string last, decimal salary )
15    {
16        FirstName = first;
17        LastName = last;
18        MonthlySalary = salary;
19    } // end constructor
20}
```

Fig. 18.23 | Employee class. (Part 1 of 2.)

```
21 // property that gets and sets the employee's monthly salary
22 public decimal MonthlySalary
23 {
24     get
25     {
26         return monthlySalaryValue;
27     } // end get
28     set
29     {
30         if ( value >= 0M ) // if salary is nonnegative
31         {
32             monthlySalaryValue = value;
33         } // end if
34     } // end set
35 } // end property MonthlySalary
36
37 // return a string containing the employee's information
38 public override string ToString()
39 {
40     return string.Format( "{0,-10} {1,-10} {2,10:C}",
41                         FirstName, LastName, MonthlySalary );
42 } // end method ToString
43 } // end class Employee
```

Fig. 18.23 | Employee class. (Part 2 of 2.)

```
1 // Fig. 18.24: LINQWithArrayOfObjects.cs
2 // LINQ to Objects using an array of Employee objects.
3 using System;
4 using System.Linq;
5
6 public class LINQWithArrayOfObjects
{
7     public static void Main( string[] args )
8     {
9         // initialize array of employees
10        Employee[] employees = {
11            new Employee( "Jason", "Red", 5000M ),
12            new Employee( "Ashley", "Green", 7600M ),
13            new Employee( "Matthew", "Indigo", 3587.5M ),
14            new Employee( "James", "Indigo", 4700.77M ),
15            new Employee( "Luke", "Indigo", 6200M ),
16            new Employee( "Jason", "Blue", 3200M ),
17            new Employee( "Wendy", "Brown", 4236.4M ) }; // end init list
18
19        // display all employees
20        Console.WriteLine( "Original array:" );
21        foreach ( var element in employees )
22            Console.WriteLine( element );
```

Fig. 18.24 | LINQ to Objects using an array of Employee objects.
(Part 1 of 6.)

```
24
25 // filter a range of salaries using && in a LINQ query
26 var between4K6K =
27     from e in employees
28     where e.MonthlySalary >= 4000M && e.MonthlySalary <= 6000M
29     select e;
30
31 // display employees making between 4000 and 6000 per month
32 Console.WriteLine( string.Format(
33     "\nEmployees earning in the range {0:C}-{1:C} per month:",
34     4000, 6000 ) );
35 foreach ( var element in between4K6K )
36     Console.WriteLine( element );
37
38 // order the employees by last name, then first name with LINQ
39 var nameSorted =
40     from e in employees
41     orderby e.LastName, e.FirstName
42     select e;
43
44 // header
45 Console.WriteLine( "\nFirst employee when sorted by name:" );
46
```

Fig. 18.24 | LINQ to Objects using an array of Employee objects.
(Part 2 of 6.)

```
47      // attempt to display the first result of the above LINQ query
48      if ( nameSorted.Any() )
49          Console.WriteLine( nameSorted.First() );
50      else
51          Console.WriteLine( "not found" );
52
53      // use LINQ to select employee last names
54      var lastNames =
55          from e in employees
56          select e.LastName;
57
58      // use method Distinct to select unique last names
59      Console.WriteLine( "\nUnique employee last names:" );
60      foreach ( var element in lastNames.Distinct() )
61          Console.WriteLine( element );
62
63      // use LINQ to select first and last names
64      var names =
65          from e in employees
66          select new { e.FirstName, Last = e.LastName };
67
68      // display full names
69      Console.WriteLine( "\nNames only:" );
```

Fig. 18.24 | LINQ to Objects using an array of Employee objects.
(Part 3 of 6.)

```
70     foreach ( var element in names )
71         Console.WriteLine( element );
72
73     Console.WriteLine();
74 } // end Main
75 } // end class LINQWithArrayOfObjects
```

Fig. 18.24 | LINQ to Objects using an array of Employee objects.
(Part 4 of 6.)

Original array:

Jason	Red	\$5,000.00
Ashley	Green	\$7,600.00
Matthew	Indigo	\$3,587.50
James	Indigo	\$4,700.77
Luke	Indigo	\$6,200.00
Jason	Blue	\$3,200.00
Wendy	Brown	\$4,236.40

Employees earning in the range \$4,000.00-\$6,000.00 per month:

Jason	Red	\$5,000.00
James	Indigo	\$4,700.77
Wendy	Brown	\$4,236.40

First employee when sorted by name:

Jason	Blue	\$3,200.00
-------	------	------------

Unique employee last names:

Red
Green
Indigo
Blue
Brown

Fig. 18.24 | LINQ to Objects using an array of Employee objects.
(Part 5 of 6.)

Names only:

```
{ FirstName = Jason, Last = Red }
{ FirstName = Ashley, Last = Green }
{ FirstName = Matthew, Last = Indigo }
{ FirstName = James, Last = Indigo }
{ FirstName = Luke, Last = Indigo }
{ FirstName = Jason, Last = Blue }
{ FirstName = Wendy, Last = Brown }
```

Fig. 18.24 | LINQ to Objects using an array of Employee objects.
(Part 6 of 6.)

18.6.3 Querying a Generic Collection Using LINQ

- ▶ You can use LINQ to Objects to query Lists just as arrays.
- ▶ LINQ's `let` clause allows you to create a new range variable.
 - Useful if you need to store a temporary result for use later in the LINQ query.
 - Typically, `let` declares a new range variable to which you assign the result of an expression that operates on the query's original range variable.
- ▶ `string` method `ToUpper` converts each item to uppercase, then stores the result in the new range variable `uppercaseString`.
- ▶ The `where` clause uses `string` method `StartsWith` to determine whether `uppercaseString` starts with the character "R".

18.6.3 Querying a Generic Collection Using LINQ (Cont.)

- ▶ deferred execution
 - the query executes only when you access the results—such as iterating over them or using the Count method—not when you define the query.
 - Allows you to create a query once and execute it many times.
 - Any changes to the data source are reflected in the results each time the query executes.
- ▶ C# has a feature called collection initializers, which provide a convenient syntax (similar to array initializers) for initializing a collection.

```
1 // Fig. 18.25: LINQwithListCollection.cs
2 // LINQ to Objects using a List< string >.
3 using System;
4 using System.Linq;
5 using System.Collections.Generic;
6
7 public class LINQwithListCollection
8 {
9     public static void Main( string[] args )
10    {
11        // populate a List of strings
12        List< string > items = new List< string >();
13        items.Add( "aQua" ); // add "aQua" to the end of the List
14        items.Add( "RusT" ); // add "RusT" to the end of the List
15        items.Add( "yElLow" ); // add "yElLow" to the end of the List
16        items.Add( "rEd" ); // add "rEd" to the end of the List
17
18        // convert all strings to uppercase; select those starting with "R"
19        var startsWithR =
20            from item in items
21            let uppercaseString = item.ToUpper()
22            where uppercaseString.StartsWith( "R" )
23            orderby uppercaseString
24            select uppercaseString;
```

Fig. 18.25 | LINQ to Objects using a List<string>. (Part I of 2.)

```
25
26     // display query results
27     foreach ( var item in startsWithR )
28         Console.WriteLine( "{0} ", item );
29
30     Console.WriteLine(); // output end of line
31
32     items.Add( "rUbY" ); // add "rUbY" to the end of the List
33     items.Add( "SaFfRon" ); // add "SaFfRon" to the end of the List
34
35     // display updated query results
36     foreach ( var item in startsWithR )
37         Console.WriteLine( "{0} ", item );
38
39     Console.WriteLine(); // output end of line
40 } // end Main
41 } // end class LINQWithListCollection
```

```
RED RUST
RED RUBY RUST
```

Fig. 18.25 | LINQ to Objects using a List<string>. (Part 2 of 2.)

18.7 (Optional) LINQ to SQL

- ▶ LINQ to SQL enables you to access data in *SQL Server databases* using the same LINQ syntax introduced in the previous section.
- ▶ You interact with the database via classes that are automatically generated from the database schema by the IDE's LINQ to SQL Designer.

18.7 (Optional) LINQ to SQL (Cont.)

- ▶ For each table in the database, the IDE creates two classes:
 - A class that represents a row of the table. LINQ to SQL creates objects of this class—called row objects—to store the data from individual rows of the table.
 - A class that represents the table: LINQ to SQL creates an object of this class to store a collection of row objects that correspond to all of the rows in the table.

18.7 (Optional) LINQ to SQL (Cont.)

- ▶ Relationships between tables are also taken into account in the generated classes:
 - In a row object's class, an additional property is created for each foreign key.
 - In the class for a row object, an additional property is created for the collection of row objects with foreign-keys that reference the row object's primary key.

18.7 (Optional) LINQ to SQL (Cont.)

IQueryable Interface

- ▶ LINQ to SQL works through the `IQueryable` interface, which inherits from the `IEnumerable` interface.
- ▶ When a LINQ to SQL query on an `IQueryable` object executes against the database, the results are loaded into objects of the corresponding LINQ to SQL classes for convenient access in your code.

18.8 (Optional) Querying a Database with LINQ

- ▶ The IDE provides *visual programming* tools and *wizards* that simplify accessing data in applications. These tools
 - establish database connections
 - create the objects necessary to view and manipulate the data through Windows Forms GUI controls—a technique known as data binding.

18.8 (Optional) Querying a Database with LINQ (Cont.)

- ▶ All of the controls in this GUI are automatically generated when we drag a data source that represents the Authors table onto the Form in Design view.
- ▶ The BindingNavigator is a collection of controls that allow you to navigate through the records in the DataGridView that fills the rest of the window.
- ▶ The BindingNavigator controls also allow you to add records, delete records and save your changes to the database.

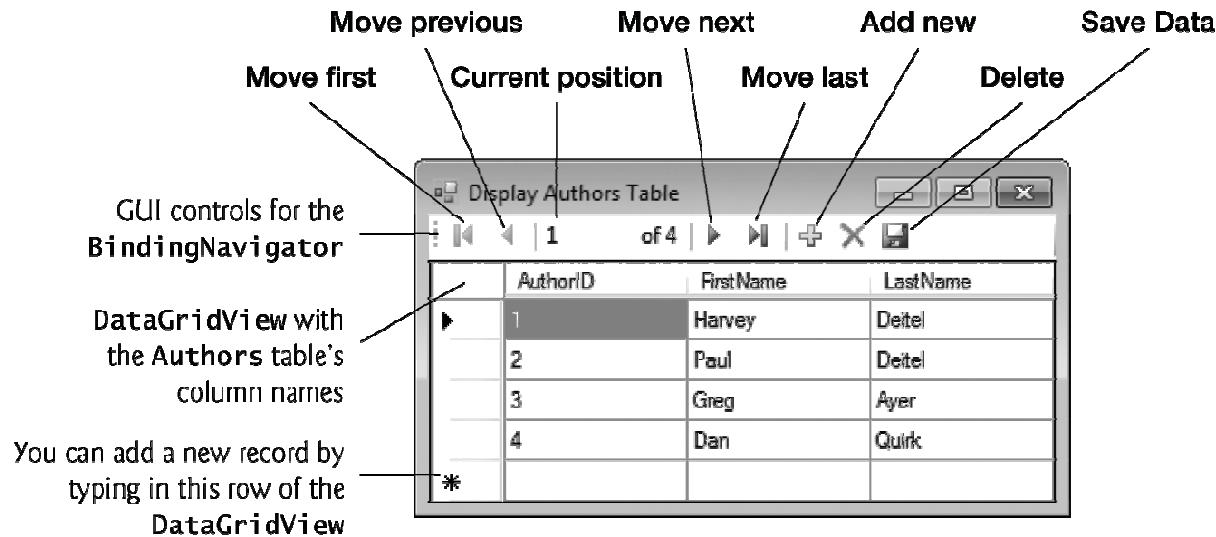


Fig. 18.26 | GUI for the **Display Authors Table** application.

18.8.1 Creating LINQ to SQL Classes

Step 1: Creating the Project

- ▶ Create a new Windows Forms Application named `DisplayTable`.
- ▶ Change the name of the source file to `DisplayAuthorsTable.cs`.
- ▶ The IDE updates the Form's class name to match the source file.
- ▶ Set the Form's Text property to Display Authors Table.

18.8.1 Creating LINQ to SQL Classes (Cont.)

Step 2: Adding a Database to the Project and Connecting to the Database

- ▶ To interact with a database, you must create a connection to the database.
 - In Visual C# 2010 Express, select View > Other Windows > Database Explorer to display the Database Explorer window. If you're using a full version of Visual Studio, select View > Server Explorer to display the Server Explorer.
 - From this point forward, we'll refer to the Database Explorer. If you have a full version of Visual Studio, substitute Server Explorer for Database Explorer in the steps.

18.8.1 Creating LINQ to SQL Classes (Cont.)

- ▶ Click the Connect to Database icon at the top of the Database Explorer.
 - If the Choose Data Source dialog appears, select Microsoft SQL Server Database File from the Data source: list.
 - If you check the Always use this selection CheckBox, the IDE will use this type of database file by default when you connect to databases in the future.
 - Click Continue to display the Add Connection dialog.

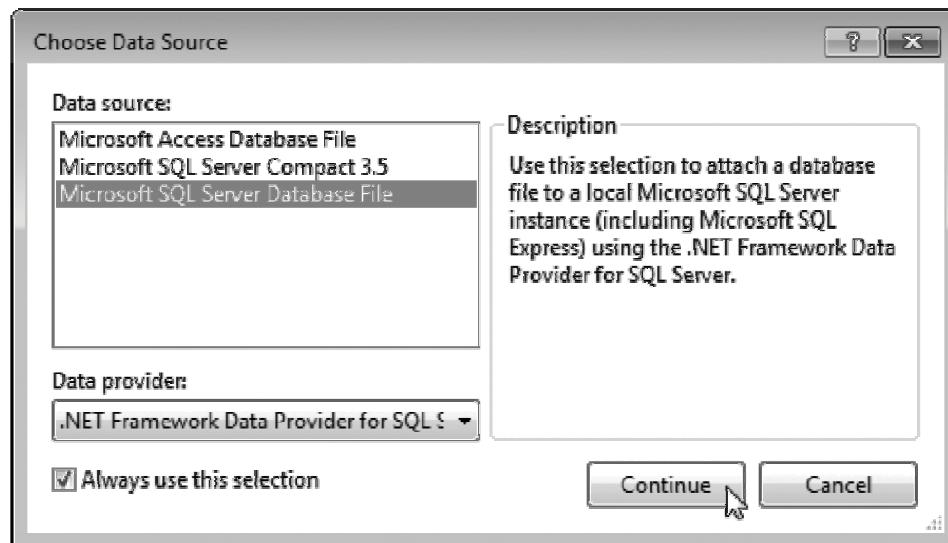


Fig. 18.27 | Choose Data Source dialog.

18.8.1 Creating LINQ to SQL Classes (Cont.)

- ▶ In the Add Connection dialog, the Data source: TextBox reflects your selection from the Choose Data Source dialog.
 - You can click the Change... Button to select a different type of database.
 - Next, click Browse... to locate and select the Books.mdf file in the Databases directory included with this chapter's examples.
 - You can click Test Connection to verify that the IDE can connect to the database through SQL Server Express. Click OK to create the connection.



Fig. 18.28 | Add Connection dialog.

18.8.1 Creating LINQ to SQL Classes (Cont.)

Step 3: Generating the LINQ to SQL classes

- ▶ Right click the project name in the Solution Explorer and select Add > New Item... to display the Add New Item dialog.
- ▶ Select the LINQ to SQL Classes template, name the new item Books.dbml and click the Add button. The Object Relational -Designer window will appear.
- ▶ You can also double click the Books.dbml file in the Solution Explorer to open the Object Relational Designer.

Drag the database's
tables here to generate
the LINQ to SQL classes



Fig. 18.29 | Object Relational Designer window.

18.8.1 Creating LINQ to SQL Classes (Cont.)

- ▶ Expand the Books.mdf database node in the Database Explorer, then expand the Tables node.
- ▶ Drag the Authors, Titles and AuthorISBN tables onto the Object Relational Designer.
- ▶ The IDE prompts whether you want to copy the database to the project directory. Select Yes.
- ▶ Save the Books.dbml file.

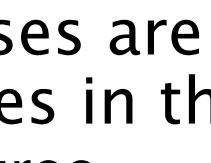


Fig. 18.30 | Object Relational Designer window showing the selected tables from the Books database and their relationships.

18.8.2 Data Bindings Between Controls and the LINQ to SQL Classes

18.8.2 Data Bindings Between Controls and the LINQ to SQL Classes

Step 1: Adding the Author LINQ to SQL Class as a Data Source

- ▶ Select Data > Add New Data Source... to display the Data Source Configuration Wizard.
- ▶ The LINQ to SQL classes are used to create objects representing the tables in the database, so we'll use an Object data source.
- ▶ In the dialog, select Object and click Next >. Expand the tree view as shown in  and ensure that Author is checked. An object of this class will be used as the data source.
- ▶ Click Finish.

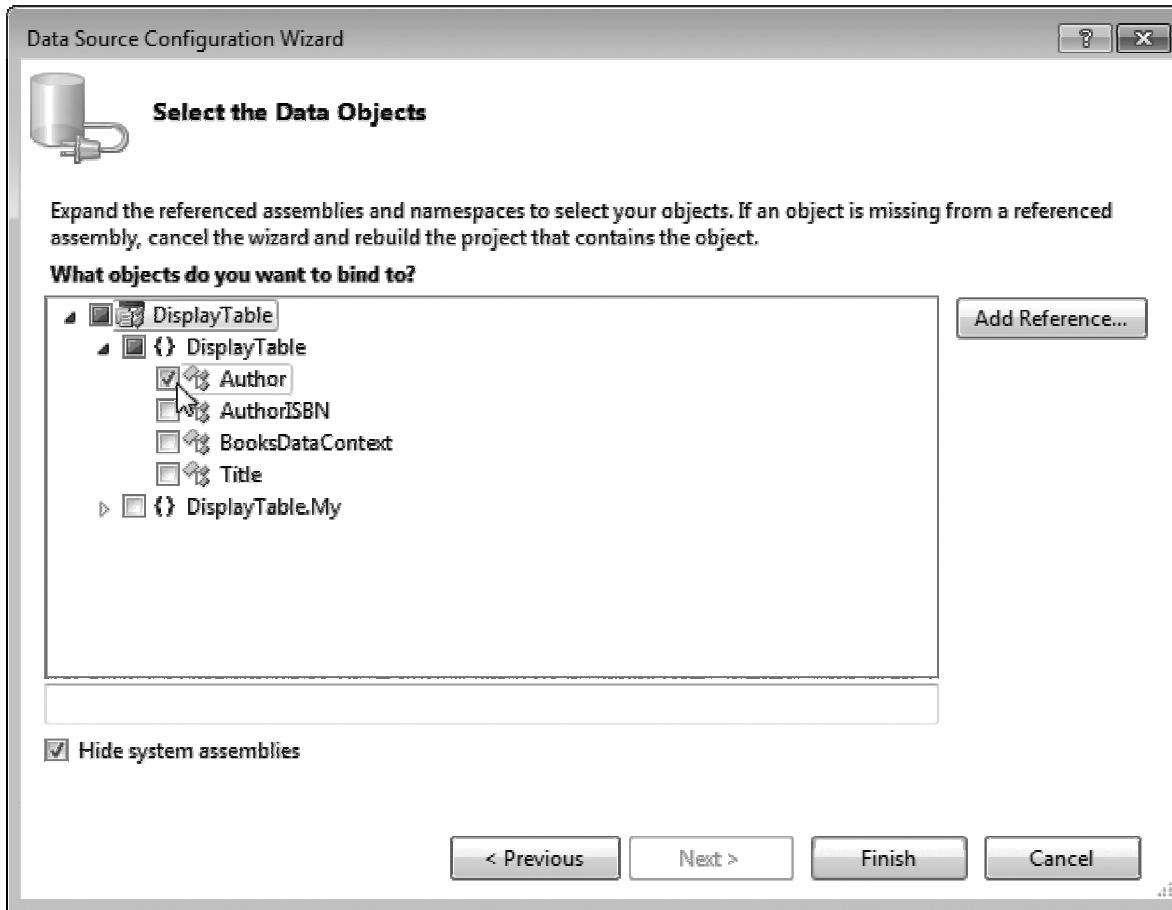


Fig. 18.31 | Selecting the Author LINQ to SQL class as the data source.



Fig. 18.32 | Data Sources window showing the Author class as a data source.

18.8.2 Data Bindings Between Controls and the LINQ to SQL Classes (Cont.)

Step 2: Creating GUI Elements

- ▶ Switch to Design view for the `DisplayAuthorsTable` class.
- ▶ Click the Author node in the Data Sources window—it should change to a drop-down list.
- ▶ Open the drop-down by clicking the down arrow and ensure that the `DataGridView` option is selected—this is the GUI control that will be used to display and interact with the data.
- ▶ Drag the Author node from the Data Sources window onto the Form in Design view.

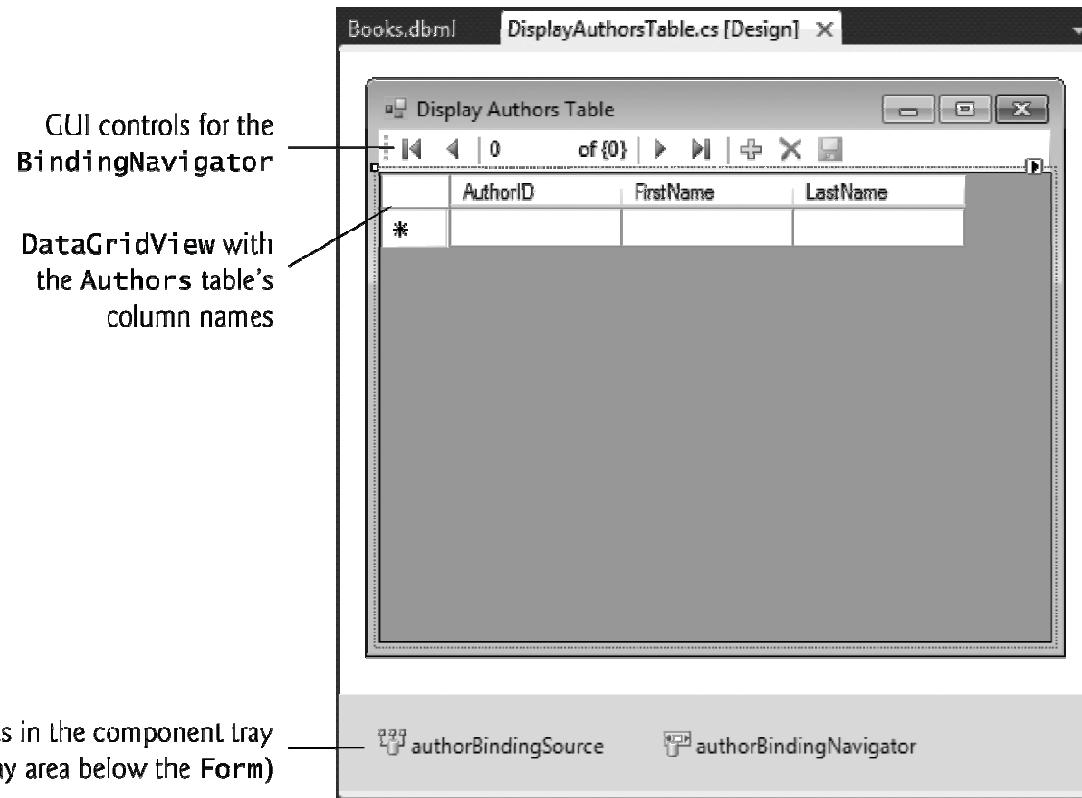


Fig. 18.33 | Component tray holds nonvisual components in Design view.

18.8.2 Data Bindings Between Controls and the LINQ to SQL Classes (Cont.)

Step 3: Connecting the BooksDataContext to the authorBindingSource

- ▶ The final step is to connect the BooksDataContext to the authorBindingSource (created earlier in this section), so that the application can interact with the database.

```
1 // Fig. 18.34: DisplayAuthorsTable.cs
2 // Displaying data from a database table in a DataGridView.
3 using System;
4 using System.Linq;
5 using System.Windows.Forms;
6
7 namespace DisplayTable
8 {
9     public partial class DisplayAuthorsTable : Form
10    {
11        // constructor
12        public DisplayAuthorsTable()
13        {
14            InitializeComponent();
15        } // end constructor
16
17        // LINQ to SQL data context
18        private BooksDataContext database = new BooksDataContext();
19    }
```

Fig. 18.34 | Displaying data from a database table in a DataGridView. (Part 1 of 3.)

```
20      // load data from database into DataGridView
21  private void DisplayAuthorsTable_Load( object sender, EventArgs e )
22  {
23      // use LINQ to order the data for display
24      authorBindingSource.DataSource =
25          from author in database.Authors
26          orderby author.AuthorID
27          select author;
28  } // end method DisplayAuthorsTable_Load
29
30  // click event handler for the Save Button in the
31  // BindingNavigator saves the changes made to the data
32  private void authorBindingNavigatorSaveItem_Click(
33      object sender, EventArgs e )
34  {
35      Validate(); // validate input fields
36      authorBindingSource.EndEdit(); // indicate edits are complete
37      database.SubmitChanges(); // write changes to database file
38  } // end method authorBindingNavigatorSaveItem_Click
39  } // end class DisplayAuthorsTable
40 } // end namespace DisplayTable
```

Fig. 18.34 | Displaying data from a database table in a
DataGridView. (Part 2 of 3.)

The screenshot shows a Windows application window titled "Display Authors Table". The window has a standard title bar with minimize, maximize, and close buttons. Below the title bar is a toolbar with icons for back, forward, search, and other controls. The main area contains a DataGridView control displaying data from a database table. The table has four columns: AuthorID, FirstName, and LastName. The data is as follows:

	AuthorID	FirstName	LastName
▶	1	Harvey	Deitel
	2	Paul	Deitel
	3	Greg	Ayer
	4	Dan	Quirk
*			

Fig. 18.34 | Displaying data from a database table in a DataGridView. (Part 3 of 3.)

18.8.2 Data Bindings Between Controls and the LINQ to SQL Classes (Cont.)

Step 4: Saving Modifications Back to the Database

- ▶ By default, the BindingNavigator's Save Data Button is disabled. To enable it, right click the icon and select Enabled.
- ▶ Double click the icon to create its click event handler.

18.8.2 Data Bindings Between Controls and the LINQ to SQL Classes (Cont.)

Step 5: Configuring the Database File to Persist Changes

- ▶ To persist changes for all executions, select the database in the Solution Explorer and set the Copy to Output Directory property in the Properties window to Copy if newer.

18.9 (Optional) Dynamically Binding LINQ to SQL Query Results

The Display Query Results application allows the user to select a query from the ComboBox at the bottom of the window, then displays the results of the query.

a) Results of the “All titles” query, which shows the contents of the **Titles** table ordered by the book titles

The screenshot shows a Windows application window titled "Display Query Results". The window has a title bar with standard window controls (minimize, maximize, close). Below the title bar is a toolbar with icons for back, forward, search, and other database operations. The main area is a grid table with four columns: ISBN, Title, EditionNumber, and Copyright. The table contains 8 rows of data. At the bottom of the grid, there is a footer bar with the text "All titles" and a dropdown arrow.

	ISBN	Title	EditionNumber	Copyright
*	0132404168	C How to Program	5	2007
*	0136152503	C++ How to Program	6	2008
*	0131752421	Internet & World Wide Web How to Program	4	2008
*	0132222205	Java How to Program	7	2007
*	0136053033	Simply Visual Basic 2008	3	2009
*	013605305X	Visual Basic 2008 How to Program	4	2009
*	013605322X	Visual C# 2008 How to Program	3	2009
*	0136151574	Visual C++ 2008 How to Program	2	2008

Fig. 18.35 | Sample execution of the Display Query Results application.

b) Results of the
"Titles with 2008
copyright" query

	ISBN	Title	EditionNumber	Copyright
▶	0136152503	C++ How to Program	6	2008
	0131752421	Internet & World Wide Web How to Program	4	2008
	0136151574	Visual C++ 2008 How to Program	2	2008
*				

Titles with 2008 copyright

c) Results of the
"Titles ending with
"How to Program"
query

	ISBN	Title	EditionNumber	Copyright
▶	0132404168	C How to Program	5	2007
	0136152503	C++ How to Program	6	2008
	0131752421	Internet & World Wide Web How to Program	4	2008
	0132222205	Java How to Program	7	2007
	013605305X	Visual Basic 2008 How to Program	4	2009
	013605322X	Visual C# 2008 How to Program	3	2009
	0136151574	Visual C++ 2008 How to Program	2	2008
*				

Titles ending with "How to Program"

Fig. 18.35 | Sample execution of the Display Query Results application.

18.9.1 Creating the Display Query Results GUI

Step 1: Creating the Project

- ▶ First, create a new Windows Forms Application named `DisplayQueryResult`. Rename the source file to `TitleQueries.cs`. Set the Form's Text property to Display Query Results.

Step 2: Creating the LINQ to SQL Classes

- ▶ Follow the steps in [to add the Books database to the project and generate the LINQ to SQL classes.](#)

18.9.1 Creating the Display Query Results GUI (Cont.)

Step 3: Creating a DataGridView to Display the Titles Table

- ▶ Follow *Steps 1 and 2* in Section 18.8.2 to create the data source and the DataGridView.
- ▶ Select the Title class as the data source, and drag the Title node from the Data Sources window onto the form.

18.9.1 Creating the Display Query Results GUI (Cont.)

- ▶ *Step 4: Adding a ComboBox to the Form*
- ▶ In Design view, add a ComboBox named queriesComboBox below the DataGridView on the Form.
- ▶ Set the ComboBox's Dock property to Bottom and the DataGridView's Dock property to Fill.
- ▶ Add the names of the queries to the ComboBox.
- ▶ Open the ComboBox's String Collection Editor by right clicking the ComboBox and selecting Edit Items.
- ▶ You can also access the String Collection Editor from the smart tag menu.

18.9.1 Creating the Display Query Results GUI (Cont.)

- ▶ In the String Collection Editor, add the following three items to `queriesComboBox`—one for each of the queries we'll create:
 - All titles
 - Titles with 2008 copyright
 - Titles ending with "How to Program"

18.9.2 Coding the Display Query Results Application

- ▶ Next you must write code that executes the appropriate query each time the user chooses a different item from queriesComboBox.

```
1 // Fig. 18.36: TitleQueries.cs
2 // Displaying the result of a user-selected query in a DataGridView.
3 using System;
4 using System.Linq;
5 using System.Windows.Forms;
6
7 namespace DisplayQueryResult
8 {
9     public partial class TitleQueries : Form
10    {
11        public TitleQueries()
12        {
13            InitializeComponent();
14        } // end constructor
15
16        // LINQ to SQL data context
17        private BooksDataContext database = new BooksDataContext();
18    }
```

Fig. 18.36 | Displaying the result of a user-selected query in a DataGridView. (Part 1 of 4.)

```
19      // load data from database into DataGridView
20  private void TitleQueries_Load( object sender, EventArgs e )
21  {
22      // write SQL to standard output stream
23      database.Log = Console.Out;
24
25      // set the ComboBox to show the default query that
26      // selects all books from the Titles table
27      queriesComboBox.SelectedIndex = 0;
28 } // end method TitleQueries_Load
29
30 // Click event handler for the Save Button in the
31 // BindingNavigator saves the changes made to the data
32 private void titleBindingNavigatorSaveItem_Click(
33     object sender, EventArgs e )
34 {
35     Validate(); // validate input fields
36     titleBindingSource.EndEdit(); // indicate edits are complete
37     database.SubmitChanges(); // write changes to database file
38
39     // when saving, return to "all titles" query
40     queriesComboBox.SelectedIndex = 0;
41 } // end method titleBindingNavigatorSaveItem_Click
```

Fig. 18.36 | Displaying the result of a user-selected query in a DataGridView. (Part 2 of 4.)

```
42 // loads data into titleBindingSource based on user-selected query
43 private void queriesComboBox_SelectedIndexChanged(
44     object sender, EventArgs e )
45 {
46     // set the data displayed according to what is selected
47     switch ( queriesComboBox.SelectedIndex )
48     {
49         case 0: // all titles
50             // use LINQ to order the books by title
51             titleBindingSource.DataSource =
52                 from book in database.Titles
53                 orderby book.Title1
54                 select book;
55             break;
56         case 1: // titles with 2008 copyright
57             // use LINQ to get titles with 2008
58             // copyright and sort them by title
59             titleBindingSource.DataSource =
60                 from book in database.Titles
61                 where book.Copyright == "2008"
62                 orderby book.Title1
63                 select book;
64             break;
65 }
```

Fig. 18.36 | Displaying the result of a user-selected query in a DataGridView. (Part 3 of 4.)

```
66     case 2: // titles ending with "How to Program"
67         // use LINQ to get titles ending with
68         // "How to Program" and sort them by title
69         titleBindingSource.DataSource =
70             from book in database.Titles
71             where book.Title1.EndsWith( "How to Program" )
72             orderby book.Title1
73             select book;
74         break;
75     } // end switch
76
77     titleBindingSource.MoveFirst(); // move to first entry
78 } // end method queriesComboBox_SelectedIndexChanged
79 } // end class TitleQueries
80 } // end namespace DisplayQueryResult
```

Fig. 18.36 | Displaying the result of a user-selected query in a DataGridView. (Part 4 of 4.)

18.10 Java DB/Apache Derby

- ▶ The Java SE 6 and 7 Development Kits (JDKs) come bundled with the open source, pure Java database Java DB (the Oracle branded version of Apache Derby) with the JDK.