19

Databases, SQL and ADO.NET

Objectives

- To understand the relational database model.
- To understand basic database queries using Structured Query Language (SQL).
- To use the classes and interfaces of namespace **`System.Data`** to manipulate databases.
- To understand and use ADO.NET's disconnected model.
- To use the classes and interfaces of namespace **`System.Data.OleDb`**.

It is a capital mistake to theorize before one has data.
Arthur Conan Doyle

Now go, write it before them in a table, and note it in a book, that it may be for the time to come for ever and ever.
The Holy Bible: The Old Testament

Let's look at the record.
Alfred Emanuel Smith

Get your facts first, and then you can distort them as much as you please.
Mark Twain

I like two kinds of men: domestic and foreign.
Mae West

Outline

19.1 Introduction

A *database* is a collection of data. There are many different strategies for organizing data to facilitate easy access and manipulation of the data. A *database management system* (*DBMS*) provides mechanisms for storing and organizing data in a manner consistent with the database's format. Database management systems allow for the access and storage of data without worrying about the internal representation of databases.

Today's most popular database systems are *relational databases*. A language called *Structured Query Language* (*SQL*—pronounced as its individual letters or as "sequel") is used almost universally with relational database systems to perform *queries* (i.e., to request information that satisfies given criteria) and to manipulate data. [*Note*: The writing in this chapter assumes that SQL is pronounced as its individual letters. For this reason, we often precede SQL with the article "an" as in "an SQL database" or "an SQL statement."]

Some popular enterprise-level relational database systems include Microsoft SQL Server, Oracle™, Sybase™, DB2™, Informix™ and MySQL™. In this chapter, we present examples using *Microsoft Access*—a relational database system that comes with *Microsoft Office*.

A programming language connects to, and interacts with, relational databases via an *interface*—software that facilitates communications between a database management system and a program. C# programmers communicate with databases and manipulate their data using the next generation of *Microsoft ActiveX Data Objects*™ (ADO), *ADO.NET*. This development framework is a *disconnected* model and uses XML for data transmissions to achieve interoperability with other platforms.

19.2 Relational Database Model

The *relational database model* is a logical representation of data that allows the relationships between the data to be considered without concern for the physical structure of the data. A relational database is composed of *tables*. Figure 19.1 illustrates a sample table that might be used in a personnel system. The table name is **`Employee`** and its primary purpose is to illustrate the specific attributes of an employee. A particular row of the table is called a *record* (or *row*). This table consists of six records. The **`number`** *field* (or *column*) of each record in the table is the *primary key* for referencing data in the table. A primary key is a field (or fields) in a table that contain(s) unique data that cannot be duplicated in other records of that table. This guarantees each record can be identified by a unique value. Examples of primary-key fields are a social security number, an employee ID and a part number in an inventory system. The records of Fig. 19.1 are *ordered* by primary key. In this case, the records are in increasing order (decreasing order could be used).

number	name	department	salary	location
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

Row/Record

Primary key

Column/Field

Fig. 19.1 Relational database structure of an **`Employee`** table.

Each column of the table represents a different *field* (or *column* or *attribute*). Records normally are unique (by primary key) within a table, but particular field values may be duplicated between records. For example, three different records in the **`Employee`** table's **`Department`** field contain the number 413.

Different users of a database often are interested in different data and different relationships among those data. Some users require only subsets of the table columns. To obtain table subsets, we use SQL statements to specify the data to *select* from a table. SQL provides a complete set of commands (including ***SELECT***) that enable programmers to define complex *queries* to select data from a table. The results of a query commonly are called *result sets* (or *record sets*). For example, we might select data from the table in Fig. 19.1 to create a new result set that shows the location of each department. This result set appears in Fig. 19.2. SQL queries are discussed in Section 19.4.

department	**location**
413	New Jersey
611	Orlando
642	Los Angeles

Fig. 19.2 Result set formed by selecting **Department** and **Location** data from the **Employee** table.

19.3 Relational Database Overview: The Books Database

This section gives an overview of SQL in the context of a sample **Books** database we created for this chapter. Before we discuss SQL, we overview the tables of the **Books** database. We use this to introduce various database concepts, including the use of SQL to obtain useful information from the database and to manipulate the database. We provide a script to create the database. You can find the script in the examples directory for this chapter on the CD that accompanies this book. Section 19.6 explains how to use this script.

The database consists of four tables: **Authors**, **Publishers**, **AuthorISBN** and **Titles**. The **Authors** table (described in Fig. 19.3) consists of three fields (or columns) that maintain each author's unique ID number, first name and last name. Figure 19.4 contains the data from the **Authors** table of the **Books** database.

Field	Description
authorID	Author's ID number in the database. In the **Books** database, this integer field is defined as an *autoincremented field*. For each new record inserted in this table, the database automatically increments the **authorID** value to ensure that each record has a unique **authorID**. This field represents the table's primary key.
firstName	Author's first name (a string).
lastName	Author's last name (a string).

Fig. 19.3 **Authors** table from **Books**.

authorID	**firstName**	**lastName**
1	Harvey	Deitel
2	Paul	Deitel
3	Tem	Nieto
4	Kate	Steinbuhler
5	Sean	Santry
6	Ted	Lin
7	Praveen	Sadhu
8	David	McPhie
9	Cheryl	Yaeger
10	Marina	Zlatkina
11	Ben	Wiedermann
12	Jonathan	Liperi

Fig. 19.4 Data from the **Authors** table of **Books**.

The **Publishers** table (described in Fig. 19.5) consists of two fields representing each publisher's unique ID and name. Figure 19.6 contains the data from the **Publishers** table of the **Books** database.

Field	Description
publisherID	The publisher's ID number in the database. This autoincremented integer is the table's primary-key field.
publisherName	The name of the publisher (a string).

Fig. 19.5 **Publishers** table from **Books**.

publisherID	**publisherName**
1	Prentice Hall
2	Prentice Hall PTG

Fig. 19.6 Data from the **Publishers** table of **Books**.

The **AuthorISBN** table (described in Fig. 19.7) consists of two fields that maintain each ISBN number and its corresponding author's ID number. This table helps associate the names of the authors with the titles of their books. Figure 19.8 contains the data from the **AuthorISBN** table of the **Books** database. ISBN is an abbreviation for "International Standard Book Number"—a numbering scheme with which publishers worldwide give every book a unique identification number. [*Note*: To save space, we have split the contents of this figure into two columns, each containing the **authorID** and **isbn** fields.]

Field	Description
authorID	The author's ID number, which allows the database to associate each book with a specific author. The integer ID number in this field must also appear in the **Authors** table.
isbn	The ISBN number for a book (a string).

Fig. 19.7 **AuthorISBN** table from **Books**.

authorID	isbn	authorID	isbn
1	0130895725	2	0139163050
1	0132261197	2	013028419x
1	0130895717	2	0130161438
1	0135289106	2	0130856118
1	0139163050	2	0130125075
1	013028419x	2	0138993947
1	0130161438	2	0130852473
1	0130856118	2	0130829277
1	0130125075	2	0134569555
1	0138993947	2	0130829293
1	0130852473	2	0130284173
1	0130829277	2	0130284181
1	0134569555	2	0130895601
1	0130829293	3	013028419x
1	0130284173	3	0130161438
1	0130284181	3	0130856118
1	0130895601	3	0134569555
2	0130895725	3	0130829293

Fig. 19.8 Portion of data from table **AuthorISBN** in database **Books**.

authorID	isbn	authorID	isbn
2	0132261197	3	0130284173
2	0130895717	3	0130284181
2	0135289106	4	0130895601

Fig. 19.8 Portion of data from table **AuthorISBN** in database **Books**.

The **Titles** table (described in Fig. 19.9) consists of six fields that maintain general information about each book in the database, including the ISBN number, title, edition number, copyright year, publisher's ID number, name of a file containing an image of the book cover, and finally, the price. Figure 19.10 contains the data from the **Titles** table.

Field	Description
isbn	ISBN number of the book (a string).
title	Title of the book (a string).
editionNumber	Edition number of the book (an integer).
copyright	Copyright year of the book (a string).
publisherID	Publisher's ID number (an integer). This value must correspond to an ID number in the **Publishers** table.
imageFile	Name of the file containing the book's cover image (a string).
price	Suggested retail price of the book (a real number). [*Note*: The prices shown in this book are for example purposes only.]

Fig. 19.9 **Titles** table from **Books**.

isbn	title	edition-Number	publish-erID	copy-right	imageFile	price
0130923613	Python How to Program	1	1	2002	**python.jpg**	$69.95
0130622214	C# How to Program	1	1	2002	**cshtp.jpg**	$69.95

Fig. 19.10 Data from the **Titles** table of **Books** (part 1 of 4).

isbn	title	edition-Number	publish-erID	copy-right	imageFile	price
0130341517	Java How to Program	4	1	2002	`jhtp4.jpg`	$69.95
0130649341	The Complete Java Training Course	4	2	2002	`javactc4.jpg`	$109.95
0130895601	Advanced Java 2 Platform How to Program	1	1	2002	`advjhtp1.jpg`	$69.95
0130308978	Internet and World Wide Web How to Program	2	1	2002	`iw3htp2.jpg`	$69.95
0130293636	Visual Basic .NET How to Program	2	1	2002	`vbnet.jpg`	$69.95
0130895636	The Complete C++ Training Course	3	2	2001	`cppctc3.jpg`	$109.95
0130895512	The Complete e-Business & e-Commerce Programming Training Course	1	2	2001	`ebecctc.jpg`	$109.95
013089561X	The Complete Internet & World Wide Web Programming Training Course	2	2	2001	`iw3ctc2.jpg`	$109.95
0130895547	The Complete Perl Training Course	1	2	2001	`perl.jpg`	$109.95
0130895563	The Complete XML Programming Training Course	1	2	2001	`xmlctc.jpg`	$109.95
0130895725	C How to Program	3	1	2001	`chtp3.jpg`	$69.95
0130895717	C++ How to Program	3	1	2001	`cpphtp3.jpg`	$69.95
013028419X	e-Business and e-Commerce How to Program	1	1	2001	`ebechtp1.jpg`	$69.95
0130622265	Wireless Internet and Mobile Business How to Program	1	1	2001	`wireless.jpg`	$69.95
0130284181	Perl How to Program	1	1	2001	`perlhtp1.jpg`	$69.95

Fig. 19.10 Data from the **`Titles`** table of **`Books`** (part 2 of 4).

isbn	title	edition-Number	publish-erID	copy-right	imageFile	price
0130284173	XML How to Program	1	1	2001	**xmlhtp1.jpg**	$69.95
0130856118	The Complete Internet and World Wide Web Programming Training Course	1	2	2000	**iw3ctc1.jpg**	$109.95
0130125075	Java How to Program (Java 2)	3	1	2000	**jhtp3.jpg**	$69.95
0130852481	The Complete Java 2 Training Course	3	2	2000	**javactc3.jpg**	$109.95
0130323640	e-Business and e-Commerce for Managers	1	1	2000	**ebecm.jpg**	$69.95
0130161438	Internet and World Wide Web How to Program	1	1	2000	**iw3htp1.jpg**	$69.95
0130132497	Getting Started with Visual C++ 6 with an Introduction to MFC	1	1	1999	**gsvc.jpg**	$49.95
0130829293	The Complete Visual Basic 6 Training Course	1	2	1999	**vbctc1.jpg**	$109.95
0134569555	Visual Basic 6 How to Program	1	1	1999	**vbhtp1.jpg**	$69.95
0132719746	Java Multimedia Cyber Classroom	1	2	1998	**javactc.jpg**	$109.95
0136325890	Java How to Program	1	1	1998	**jhtp1.jpg**	$0.00
0139163050	The Complete C++ Training Course	2	2	1998	**cppctc2.jpg**	$109.95
0135289106	C++ How to Program	2	1	1998	**cpphtp2.jpg**	$49.95
0137905696	The Complete Java Training Course	2	2	1998	**javactc2.jpg**	$109.95
0130829277	The Complete Java Training Course (Java 1.1)	2	2	1998	**javactc2.jpg**	$99.95
0138993947	Java How to Program (Java 1.1)	2	1	1998	**jhtp2.jpg**	$49.95

Fig. 19.10 Data from the **Titles** table of **Books** (part 3 of 4).

isbn	title	edition-Number	publish-erID	copy-right	imageFile	price
0131173340	C++ How to Program	1	1	1994	**`cpphtp1.jpg`**	$69.95
0132261197	C How to Program	2	1	1994	**`chtp2.jpg`**	$49.95
0131180436	C How to Program	1	1	1992	**`chtp.jpg`**	$69.95

Fig. 19.10 Data from the **`Titles`** table of **`Books`** (part 4 of 4).

Figure 19.11 illustrates the relationships among the tables in the **`Books`** database. The first line in each table is the table's name. The field name in italic contains that table's primary key. A table's primary key uniquely identifies each record in the table. Every record must have a value in the primary-key field, and the value must be unique. This is known as the *Rule of Entity Integrity*. Note that the AuthorISBN has two fields in italic. This indicates that these two fields form a *compound primary key*—each record in the table must have a unique **`authorID`** and **`isbn`** combination. For example, there may exist several records with an **`authorID`** of **`2`** and several records with an **`isbn`** of **`0130895601`**, but only one record can have an **`authorID`** of **`2`** and an **`isbn`** of **`0130895601`**.

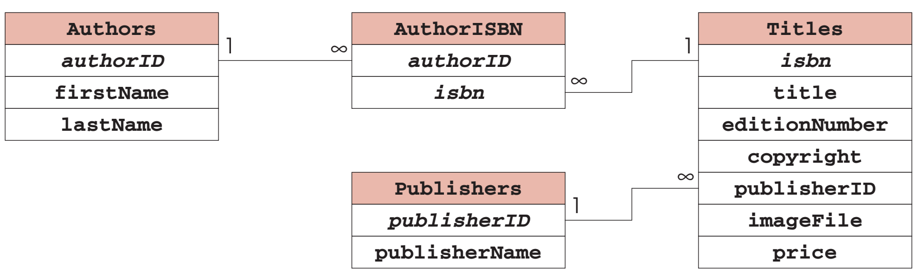

Fig. 19.11 Table relationships in **`Books`**.

Common Programming Error 19.1

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

Common Programming Error 19.2

Providing duplicate values for the primary-key field in multiple records causes the DBMS to report an error.

The lines connecting the tables in Fig. 19.11 represent the *relationships* between the tables. Consider the line between the **Publishers** and **Titles** tables. On the **Publishers** end of the line, there is a **1**, and on the **Titles** end, there is an infinity (∞) symbol, indicating a *one-to-many relationship* in which every publisher in the **Publishers** table can have an arbitrarily large number of books in the **Titles** table. Note that the relationship line links the **publisherID** field in the table **Publishers** to the **publisherID** field in table **Titles**. The **publisherID** field in the **Titles** table is a *foreign key*—a field for which every entry has a unique value in another table and where the field in the other table is the primary key for that table (e.g., **publisherID** in the **Publishers** table). Foreign keys are specified when creating a table. The foreign key helps maintain the *Rule of Referential Integrity*: Every foreign key-field value must appear in another table's primary-key field. Foreign keys enable information from multiple tables to be *joined* together for analysis purposes. There is a one-to-many relationship between a primary key and its corresponding foreign key. This means that a foreign key-field value can appear many times in its own table, but can only appear once as the primary key of another table. The line between the tables represents the link between the foreign key in one table and the primary key in another table.

Common Programming Error 19.3

Providing a foreign-key value that does not appear as a primary-key value in another table breaks the Rule of Referential Integrity and causes the DBMS to report an error.

The line between the **AuthorISBN** and **Authors** tables indicates that for each author in the **Authors** table, there can be an arbitrary number of ISBNs for books written by that author in the **AuthorISBN** table. The **authorID** field in the **AuthorISBN** table is a foreign key of the **authorID** field (the primary key) of the **Authors** table. Note again that the line between the tables links the foreign key of table **AuthorISBN** to the corresponding primary key in table **Authors**. The **AuthorISBN** table links information in the **Titles** and **Authors** tables.

Finally, the line between the **Titles** and **AuthorISBN** tables illustrates a one-to-many relationship; a title can be written by any number of authors. In fact, the sole purpose of the **AuthorISBN** table is to represent a many-to-many relationship between the **Authors** and **Titles** tables; an author can write any number of books and a book can have any number of authors.

19.4 Structured Query Language (SQL)

In this section, we provide an overview of Structured Query Language (SQL) in the context of our **Books** sample database. You will be able to use the SQL queries discussed here in the examples later in the chapter.

We discuss the SQL keywords of Fig. 19.12 in the contexts of complete SQL queries in the next several subsections—other SQL keywords are beyond the scope of this text.

[*Note*: For more information on SQL, please refer to the bibliography at the end of this chapter.]

SQL keyword	Description
SELECT	Select (retrieve) fields from one or more tables.
FROM	Tables from which to get fields or delete records. Required in every **SELECT** and **DELETE**.
WHERE	Criteria for selection that determine the rows to be retrieved.
INNER JOIN	Join records from multiple tables to produce a single set of records.
GROUP BY	Criteria for grouping records.
ORDER BY	Criteria for ordering records.
INSERT	Insert data into a specified table.
UPDATE	Update data in a specified table
DELETE	Delete data from a specified table.

Fig. 19.12 SQL query keywords.

19.4.1 Basic SELECT Query

Let us consider several SQL queries that extract information from database **Books**. A typical SQL query "selects" information from one or more tables in a database. Such selections are performed by ***SELECT** queries*. The simplest format of a **SELECT** query is

```
SELECT * FROM tableName
```

In this query, the asterisk (*****) indicates that all columns from the *tableName* table of the database should be selected. For example, to select the entire contents of the **Authors** table (i.e., all the data in Fig. 19.13), use the query

```
SELECT * FROM Authors
```

To select specific fields from a table, replace the asterisk (*****) with a comma-separated list of the field names to select. For example, to select only the fields **authorID** and **lastName** for all rows in the **Authors** table use the query

```
SELECT authorID, lastName FROM Authors
```

This query returns the data in Fig. 19.13. [*Note*: If a field name contains spaces, it must be enclosed in square brackets (**[]**) in the query. For example, if the field name is **first name**, the field name would appear in the query as **[first name]**.]

authorID	lastName
1	Deitel
2	Deitel
3	Nieto
4	Steinbuhler
5	Santry
6	Lin
7	Sadhu
8	McPhie
9	Yaeger
10	Zlatkina
11	Wiedermann
12	Liperi

Fig. 19.13 `authorID` and `lastName` from the `Authors` table.

Common Programming Error 19.4

If a program assumes that the fields in a result set are always returned in the same order from an SQL statement that uses the asterisk (`*`) to select fields, the program could process the result set incorrectly. If the field order in the database table(s) changes, the order of the fields in the result set would change accordingly.

Performance Tip 19.1

If the order of fields in a result set is unknown to a program, the program must process the fields by name. This could require a linear search of the field names in the result set. Specifying the field names to select from a table (or several tables) enables the application receiving the result set to know the order of the fields in advance. In this case, the program can process the data more efficiently, because fields can be accessed directly by column number.

19.4.2 WHERE Clause

In most cases, it is necessary to locate records in a database that satisfy certain *selection criteria*. Only records that match the selection criteria are selected. SQL uses the optional ***WHERE** clause* in a **SELECT** query to specify the selection criteria for the query. The simplest format of a **SELECT** query with selection criteria is

```
SELECT fieldName1, fieldName2, … FROM tableName WHERE criteria
```

For example, to select the **title**, **editionNumber** and **copyright** fields from those rows of table **Titles**, where the **copyright** date is greater than **1999**, use the query

```
SELECT title, editionNumber, copyright
```

```
FROM Titles
WHERE copyright > 1999
```

Figure 19.14 shows the results of the preceding query. [*Note*: When we construct a query for use in C#, we will simply create a **`String`** containing the entire query. When we display queries in the text, we often use multiple lines and indentation for readability.]

Title	editionNumber	copyright
Internet and World Wide Web How to Program	2	2002
Java How to Program	4	2002
The Complete Java Training Course	4	2002
The Complete e-Business & e-Commerce Programming Training Course	1	2001
The Complete Internet & World Wide Web Programming Training Course	2	2001
The Complete Perl Training Course	1	2001
The Complete XML Programming Training Course	1	2001
C How to Program	3	2001
C++ How to Program	3	2001
The Complete C++ Training Course	3	2001
e-Business and e-Commerce How to Program	1	2001
Internet and World Wide Web How to Program	1	2000
The Complete Internet and World Wide Web Programming Training Course	1	2000
Java How to Program (Java 2)	3	2000
The Complete Java 2 Training Course	3	2000
XML How to Program	1	2001
Perl How to Program	1	2001
Advanced Java 2 Platform How to Program	1	2002
e-Business and e-Commerce for Managers	1	2000
Wireless Internet and Mobile Business How to Program	1	2001
C# How To Program	1	2002
Python How to Program	1	2002
Visual Basic .NET How to Program	2	2002

Fig. 19.14 Titles with copyrights after 1999 from table **`Titles`**.

Performance Tip 19.2

Using selection criteria improves performance by selecting a portion of the database that is normally smaller than the entire database. Working with a smaller portion of the data is more efficient than working with the entire set of data stored in the database.

The **WHERE** clause condition can contain operators **<**, **>**, **<=**, **>=**, **=**, **<>** and **LIKE**. Operator **LIKE** is used for *pattern matching* with wildcard characters *asterisk* (*****) and *question mark* (**?**). Pattern matching allows SQL to search for similar strings that "match a pattern."

A pattern that contains an asterisk (*****) searches for strings that have zero or more characters at the asterisk character's position in the pattern. For example, the following query locates the records of all the authors whose last names start with the letter **D**:

```
SELECT authorID, firstName, lastName
FROM Authors
WHERE lastName LIKE 'D*'
```

The preceding query selects the two records shown in Fig. 19.15, because two of the authors in our database have last names starting with the letter **D** (followed by zero or more characters). The ***** in the **WHERE** clause's **LIKE** pattern indicates that any number of characters can appear after the letter **D** in the **lastName** field. Notice that the pattern string is surrounded by single-quote characters.

authorID	firstName	lastName
1	Harvey	Deitel
2	Paul	Deitel

Fig. 19.15 Authors whose last names start with **D** from the **Authors** table.

Portability Tip 19.1

Not all database systems support the **LIKE** *operator, so be sure to read your database system's documentation carefully.*

Portability Tip 19.2

Most databases use the **%** *character in place of the* ***** *in a* **LIKE** *expression.*

Portability Tip 19.3

In some databases, string data is case sensitive.

Portability Tip 19.4

In some databases, table names and field names are case sensitive.

Good Programming Practice 19.1

By convention, SQL keywords should use all uppercase letters on systems that are not case sensitive to emphasize the SQL keywords in an SQL statement.

A question mark (**`?`**) in the pattern string indicates a single character at that position in the pattern. For example, the following query locates the records of all the authors whose last names start with any character (specified with **`?`**) followed by the letter **`i`** followed by any number of additional characters (specified with **`*`**):

```
SELECT authorID, firstName, lastName
FROM Authors
WHERE lastName LIKE '?i*'
```

The preceding query produces the record in Fig. 19.16, because only one author in our database has a last name that contains the letter **`i`** as its second letter.

authorID	firstName	lastName
3	Tem	Nieto
6	Ted	Lin
11	Ben	Wiedermann
12	Jonathan	Liperi

Fig. 19.16 The authors from the **`Authors`** table whose last names contain **`i`** as their second letter.

Portability Tip 19.5

Most databases use the _ character in place of the **`?`** *in a* **`LIKE`** *expression.*

19.4.3 ORDER BY Clause

The results of a query can be arranged in ascending or descending order with the optional ***`ORDER BY`*** *clause*. The simplest form of an **`ORDER BY`** clause is

```
SELECT fieldName1, fieldName2, … FROM tableName ORDER BY field ASC
SELECT fieldName1, fieldName2, … FROM tableName ORDER BY field DESC
```

where **`ASC`** specifies ascending order (lowest to highest), **`DESC`** specifies descending order (highest to lowest) and *field* specifies the field that determines the sorting order.

For example, to obtain the list of authors in ascending order by last name (Fig. 19.17), use the query

```
SELECT authorID, firstName, lastName
FROM Authors
ORDER BY lastName ASC
```

Note that the default sorting order is ascending, so **`ASC`** is optional.

authorID	firstName	lastName
2	Paul	Deitel
1	Harvey	Deitel
6	Ted	Lin
12	Jonathan	Liperi
8	David	McPhie
3	Tem	Nieto
7	Praveen	Sadhu
5	Sean	Santry
4	Kate	Steinbuhler
11	Ben	Wiedermann
9	Cheryl	Yaeger
10	Marina	Zlatkina

Fig. 19.17 Authors from table **`Authors`** in ascending order by **`lastName`**.

To obtain the same list of authors in descending order by last name (Fig. 19.18), use the query

```
SELECT authorID, firstName, lastName
FROM Authors
ORDER BY lastName DESC
```

authorID	firstName	lastName
10	Marina	Zlatkina
9	Cheryl	Yaeger
11	Ben	Wiedermann
4	Kate	Steinbuhler
5	Sean	Santry
7	Praveen	Sadhu
3	Tem	Nieto
8	David	McPhie
12	Jonathan	Liperi
6	Ted	Lin
2	Paul	Deitel

Fig. 19.18 Authors from table **`Authors`** in descending order by **`lastName`**.

authorID	firstName	lastName
1	Harvey	Deitel

Fig. 19.18 Authors from table **Authors** in descending order by **lastName**.

Multiple fields can be used for ordering purposes with an **ORDER BY** clause of the form

```
ORDER BY field1 sortingOrder, field2 sortingOrder, ...
```

where *sortingOrder* is either **ASC** or **DESC**. Note that the *sortingOrder* does not have to be identical for each field. The query

```
SELECT authorID, firstName, lastName
FROM Authors
ORDER BY lastName, firstName
```

sorts in ascending order all the authors by last name, then by first name. If any authors have the same last name, their records are returned sorted by their first name (Fig. 19.19).

authorID	firstName	lastName
1	Harvey	Deitel
2	Paul	Deitel
6	Ted	Lin
12	Jonathan	Liperi
8	David	McPhie
3	Tem	Nieto
7	Praveen	Sadhu
5	Sean	Santry
4	Kate	Steinbuhler
11	Ben	Wiedermann
9	Cheryl	Yaeger
10	Marina	Zlatkina

Fig. 19.19 Authors from table **Authors** in ascending order by **lastName** and by **firstName**.

The **WHERE** and **ORDER BY** clauses can be combined in one query. For example, the query

```
SELECT isbn, title, editionNumber, copyright, price
FROM Titles
WHERE title
LIKE '%How to Program' ORDER BY title ASC
```

returns the isbn, title, edition number, copyright and price of each book in the **`Titles`** table that has a **`title`** ending with "**`How to Program`**" and lists them in ascending order by **`title`**. The results of the query are shown in Fig. 19.20. In the figure, note that the title "e-Business and e-Commerce How to Program" appears at the end of the list because database systems often use Unicode numeric values of the characters for comparison purposes. Remember that lowercase letters have larger numeric values than uppercase letters.

isbn	title	edition-Number	copy-right	price
0130895601	Advanced Java 2 Platform How to Program	1	2002	$69.95
0131180436	C How to Program	1	1992	$69.95
0130895725	C How to Program	3	2001	$69.95
0132261197	C How to Program	2	1994	$49.95
0130622214	C# How To Program	1	2002	$69.95
0135289106	C++ How to Program	2	1998	$49.95
0131173340	C++ How to Program	1	1994	$69.95
0130895717	C++ How to Program	3	2001	$69.95
013028419X	e-Business and e-Commerce How to Program	1	2001	$69.95
0130308978	Internet and World Wide Web How to Program	2	2002	$69.95
0130161438	Internet and World Wide Web How to Program	1	2000	$69.95
0130341517	Java How to Program	4	2002	$69.95
0136325890	Java How to Program	1	1998	$0.00
0130284181	Perl How to Program	1	2001	$69.95
0130923613	Python How to Program	1	2002	$69.95
0130293636	Visual Basic .NET How to Program	2	2002	$69.95
0134569555	Visual Basic 6 How to Pro-gram	1	1999	$69.95
0130622265	Wireless Internet and Mobile Business How to Program	1	2001	$69.95
0130284173	XML How to Program	1	2001	$69.95

Fig. 19.20 Books from table **`Titles`** whose titles end with **`How to Program`** in ascending order by **`title`**.

19.4.4 Merging Data from Multiple Tables: INNER JOIN

Often it is necessary to merge data from multiple tables into a single set of data for analysis purposes. This is referred to as *joining* the tables and is accomplished using an ***INNER JOIN*** operation in the **SELECT** query. An **INNER JOIN** merges records from two or more tables by testing for matching values in a field that is common to both tables. The simplest format of an **INNER JOIN** clause is

```
SELECT fieldName1, fieldName2, …
FROM table1
INNER JOIN table2
   ON table1.fieldName = table2.fieldName
```

The **ON** part of the **INNER JOIN** clause specifies the fields from each table that should be compared to determine which records to select. For example, the following query produces a list of authors and the ISBN numbers for the books that each author wrote:

```
SELECT firstName, lastName, isbn
FROM Authors
INNER JOIN AuthorISBN
   ON Authors.authorID = AuthorISBN.authorID
ORDER BY lastName, firstName
```

The query merges the **firstName** and **lastName** fields from table **Authors** and the **isbn** field from table **AuthorISBN** and sorts the results in ascending order by **lastName** and **firstName**. Notice the use of the syntax *tableName***.***fieldName* in the **ON** part of the **INNER JOIN**. This syntax (called a *fully qualified name*) specifies the fields from each table that should be compared to join the tables. The "*tableName***.**" syntax is required if the fields have the same name in both tables. The same syntax can be used in a query to distinguish fields in different tables that happen to have the same name. Fully qualified names that start with the database name can be used to perform cross-database queries.

Software Engineering Observation 19.1

If an SQL statement uses fields with the same name from multiple tables, the statement must qualify those field names with their table names and the dot operator (e.g., ***Authors.authorID****).*

Common Programming Error 19.5

In a query, not providing fully-qualified names for fields with the same name from two or more tables is an error.

As always, the query can contain an **ORDER BY** clause. Figure 19.21 shows the results of the preceding query. [*Note*: To save space, we split the results of the query into two columns, each containing the **firstName**, **lastName** and **isbn** fields.]

firstName	lastName	isbn	firstName	lastName	isbn
Harvey	Deitel	0130895601	Paul	Deitel	0134569555
Harvey	Deitel	0130284181	Paul	Deitel	0130829277
Harvey	Deitel	0130284173	Paul	Deitel	0130852473
Harvey	Deitel	0130829293	Paul	Deitel	0138993947
Harvey	Deitel	0134569555	Paul	Deitel	0130125075
Harvey	Deitel	0130829277	Paul	Deitel	0130856118
Harvey	Deitel	0130852473	Paul	Deitel	0130161438
Harvey	Deitel	0138993947	Paul	Deitel	013028419x
Harvey	Deitel	0130125075	Paul	Deitel	0139163050
Harvey	Deitel	0130856118	Paul	Deitel	0135289106
Harvey	Deitel	0130161438	Paul	Deitel	0130895717
Harvey	Deitel	013028419x	Paul	Deitel	0132261197
Harvey	Deitel	0139163050	Paul	Deitel	0130895725
Harvey	Deitel	0135289106	Tem	Nieto	0130284181
Harvey	Deitel	0130895717	Tem	Nieto	0130284173
Harvey	Deitel	0132261197	Tem	Nieto	0130829293
Harvey	Deitel	0130895725	Tem	Nieto	0134569555
Paul	Deitel	0130895601	Tem	Nieto	0130856118
Paul	Deitel	0130284181	Tem	Nieto	0130161438
Paul	Deitel	0130284173	Tem	Nieto	013028419x
Paul	Deitel	0130829293	Sean	Santry	0130895601

Fig. 19.21 Portion of the authors and the ISBN numbers for the books they have written in ascending order by **lastName** and **firstName**.

19.4.5 Joining Data from Tables Authors, AuthorISBN, Titles and Publishers

The **Books** database contains one predefined query (**TitleAuthor**) that produces a table containing the book title, ISBN number, author's first name, author's last name, book's copyright year and publisher's name for each book in the database. For books with multiple authors, the query produces a separate composite record for each author. The **TitleAuthor** query is shown in Fig. 18.22. A portion of the query results are shown in Fig. 18.23.

```
SELECT Titles.title, Titles.isbn, Authors.firstName,
       Authors.lastName, Titles.copyright,
       Publishers.publisherName
FROM
    ( Publishers INNER JOIN Titles
      ON Publishers.publisherID = Titles.publisherID )
    INNER JOIN
    ( Authors INNER JOIN AuthorISBN
      ON Authors.authorID = AuthorISBN.authorID )
    ON Titles.isbn = AuthorISBN.isbn
ORDER BY Titles.title
```

Fig. 19.22 Joining tables to produce a result set in which each record contains an author, title, ISBN number, copyright and publisher name.

Title	isbn	first-Name	last-Name	copy-right	publisher-Name
Advanced Java 2 Platform How to Program	0130895601	Paul	Deitel	2002	Prentice Hall
Advanced Java 2 Platform How to Program	0130895601	Harvey	Deitel	2002	Prentice Hall
Advanced Java 2 Platform How to Program	0130895601	Sean	Santry	2002	Prentice Hall
C How to Program	0131180436	Harvey	Deitel	1992	Prentice Hall
C How to Program	0131180436	Paul	Deitel	1992	Prentice Hall
C How to Program	0132261197	Harvey	Deitel	1994	Prentice Hall
C How to Program	0132261197	Paul	Deitel	1994	Prentice Hall
C How to Program	0130895725	Harvey	Deitel	2001	Prentice Hall
C How to Program	0130895725	Paul	Deitel	2001	Prentice Hall
C# How To Program	0130622214	Tem	Nieto	2002	Prentice Hall
C# How To Program	0130622214	Paul	Deitel	2002	Prentice Hall
C# How To Program	0130622214	Cheryl	Yaeger	2002	Prentice Hall
C# How To Program	0130622214	Marina	Zlatkina	2002	Prentice Hall
C# How To Program	0130622214	Harvey	Deitel	2002	Prentice Hall
C++ How to Program	0130895717	Paul	Deitel	2001	Prentice Hall
C++ How to Program	0130895717	Harvey	Deitel	2001	Prentice Hall
C++ How to Program	0131173340	Paul	Deitel	1994	Prentice Hall

Fig. 19.23 Portion of the result set produced by the query in Fig. 19.22.

Title	isbn	first-Name	last-Name	copy-right	publisher-Name
C++ How to Program	0131173340	Harvey	Deitel	1994	Prentice Hall
C++ How to Program	0135289106	Harvey	Deitel	1998	Prentice Hall
C++ How to Program	0135289106	Paul	Deitel	1998	Prentice Hall
e-Business and e-Commerce for Managers	0130323640	Harvey	Deitel	2000	Prentice Hall
e-Business and e-Commerce for Managers	0130323640	Kate	Stein-buhler	2000	Prentice Hall
e-Business and e-Commerce for Managers	0130323640	Paul	Deitel	2000	Prentice Hall
e-Business and e-Commerce How to Program	013028419X	Harvey	Deitel	2001	Prentice Hall
e-Business and e-Commerce How to Program	013028419X	Paul	Deitel	2001	Prentice Hall
e-Business and e-Commerce How to Program	013028419X	Tem	Nieto	2001	Prentice Hall

Fig. 19.23 Portion of the result set produced by the query in Fig. 19.22.

The indentation in the query of Fig. 19.22 is simply to make the query more readable. Let us now break down the query into its various parts. Lines 1 through 3 indicate the fields that will be returned by the query and their order in the returned table from left to right. This query will select fields **`title`** and **`isbn`** from table **`Titles`**, fields **`firstName`** and **`lastName`** from table **`Authors`**, field **`copyright`** from table **`Titles`** table and field **`publisherName`** from table **`Publishers`**. For the purpose of this query, we fully qualified each field name with its table name (e.g., **`Titles.isbn`**).

Lines 4 through 10 specify the **`INNER JOIN`** operations that combine information from the tables. Notice that there are three **`INNER JOIN`** operations. Remember that an **`INNER JOIN`** is performed on two tables. It is important to note that either of those two tables can be the result of another query or another **`INNER JOIN`**. Parentheses are used to nest the **`INNER JOIN`** operations and the parentheses are evaluated from the innermost set of parentheses first. We begin with the **`INNER JOIN`**

```
( Publishers INNER JOIN Titles
   ON Publishers.publisherID = Titles.publisherID )
```

which joins the **`Publishers`** table and the **`Titles`** table **`ON`** the condition that the **`publisherID`** number in each table matches. The resulting temporary table contains all the information about each book and the publisher that published it.

Moving to the other nested set of parentheses, the **`INNER JOIN`**

```
( Authors INNER JOIN AuthorISBN ON
   Authors.AuthorID = AuthorISBN.AuthorID )
```

joins the **Authors** table and the **AuthorISBN** table **ON** the condition that the **authorID** field in each table matches. Remember that the **AuthorISBN** table may have multiple entries for each **ISBN** number if there is more than one author for that book.

Next, the **INNER JOIN**

```
( Publishers INNER JOIN Titles
   ON Publishers.publisherID = Titles.publisherID )
INNER JOIN
( Authors INNER JOIN AuthorISBN
   ON Authors.authorID = AuthorISBN.authorID )
ON Titles.isbn = AuthorISBN.isbn
```

joins the two temporary tables produced by the prior inner joins **ON** the condition that the **Titles.isbn** field in the first temporary table matches the **AuthorISBN.isbn** field in the second temporary table. The result of all these **INNER JOIN** operations is a temporary table from which the appropriate fields are selected for the results of this query.

Finally, line 11 of the query

```
ORDER BY Titles.title
```

indicates that all the titles should be sorted in ascending order (the default).

19.4.6 INSERT Statement

The ***INSERT*** statement inserts a new record in a table. The simplest form of this statement is

```
INSERT INTO tableName ( fieldName1, fieldName2, …, fieldNameN )
   VALUES ( value1, value2, …, valueN )
```

where *tableName* is the table in which to insert the record. The *tableName* is followed by a comma-separated list of field names in parentheses. (This list is not required if the **INSERT INTO** operation specifies a value for every column of the table in the correct order.) The list of field names is followed by the SQL keyword **VALUES** and a comma-separated list of values in parentheses. The values specified here should match the field names specified after the table name in order and type (i.e., if *fieldName1* is supposed to be the **firstName** field, then *value1* should be a string in single quotes representing the first name). The **INSERT** statement

```
INSERT INTO Authors ( firstName, lastName )
   VALUES ( 'Sue', 'Smith' )
```

inserts a record into the **Authors** table. The statement indicates that values will be inserted for the **firstName** and **lastName** fields. The corresponding values to insert are **'Sue'** and **'Smith'**. We do not specify an **authorID** in this example, because **authorID** is an *auto-increment field* in the database. Every new record added to this table, has a unique **authorID** value that is the next value in the auto-increment sequence (i.e., 1, 2, 3 etc.) assigned to it. In this case, Sue Smith would be assigned **authorID** number 5. Figure 19.24 shows the **Authors** table after performing the **INSERT** operation.

authorID	firstName	lastName
1	Harvey	Deitel
2	Paul	Deitel
3	Tem	Nieto
4	Kate	Steinbuhler
5	Sean	Santry
6	Ted	Lin
7	Praveen	Sadhu
8	David	McPhie
9	Cheryl	Yaeger
10	Marina	Zlatkina
11	Ben	Wiedermann
12	Jonathan	Liperi
13	Sue	Smith

Fig. 19.24 Table **`Authors`** after an **`INSERT INTO`** operation to add a record.

Common Programming Error 19.6

*SQL statements use the single quote (`'`) character as a delimiter for strings. To specify a string containing a single quote (such as O'Malley) in an SQL statement, the string must have two single quotes in the position where the single-quote character appears in the string (e.g., **`'O''Malley'`**). The first of the two single-quote characters acts as an escape character for the second. Not escaping single-quote characters in a string that is part of an SQL statement is an SQL syntax error.*

19.4.7 UPDATE Statement

An ***`UPDATE`*** statement modifies data in a table. The simplest form for an **`UPDATE`** statement is

```
UPDATE tableName
   SET fieldName1 = value1, fieldName2 = value2, …, fieldNameN = valueN
   WHERE criteria
```

where *tableName* is the table in which to update a record (or records). The *tableName* is followed by keyword ***`SET`*** and a comma-separated list of field name/value pairs in the format *fieldName* **`=`** *value*. The **`WHERE`** clause specifies the criteria used to determine which record(s) to update. The **`UPDATE`** statement

```
UPDATE Authors
   SET lastName = 'Jones'
   WHERE lastName = 'Smith' AND firstName = 'Sue'
```

updates a record in the **Authors** table. The statement indicates that the **lastName** will be assigned the value **Jones** for the record in which **lastName** is equal to **Smith** and **firstName** is equal to **Sue**. If we know the **authorID** in advance of the **UPDATE** operation (possibly because we searched for the record previously), the **WHERE** clause could be simplified as follows:

```
WHERE AuthorID = 5
```

Figure 19.25 shows the **Authors** table after performing the **UPDATE** operation.

authorID	firstName	lastName
1	Harvey	Deitel
2	Paul	Deitel
3	Tem	Nieto
4	Kate	Steinbuhler
5	Sean	Santry
6	Ted	Lin
7	Praveen	Sadhu
8	David	McPhie
9	Cheryl	Yaeger
10	Marina	Zlatkina
11	Ben	Wiedermann
12	Jonathan	Liperi
13	Sue	Jones

Fig. 19.25 Table **Authors** after an **UPDATE** operation to change a record.

Common Programming Error 19.7

*Not using a **WHERE** clause with an **UPDATE** or **DELETE** statement could lead to logic errors.*

19.4.8 DELETE Statement

An SQL ***DELETE*** statement removes data from a table. The simplest form for a **DELETE** statement is

DELETE FROM *tableName* **WHERE** *criteria*

where *tableName* is the table from which to delete a record (or records). The **WHERE** clause specifies the criteria used to determine which record(s) to delete. The **DELETE** statement

```
DELETE FROM Authors
   WHERE lastName = 'Jones' AND firstName = 'Sue'
```

deletes the record for Sue Jones in the **Authors** table. If we know the **authorID** in advance of the **DELETE** operation, the **WHERE** clause could be simplified as follows:

```
WHERE authorID = 13
```

Figure 19.26 shows the **Authors** table after the **DELETE** operation.

authorID	firstName	lastName
1	Harvey	Deitel
2	Paul	Deitel
3	Tem	Nieto
4	Kate	Steinbuhler
5	Sean	Santry
6	Ted	Lin
7	Praveen	Sadhu
8	David	McPhie
9	Cheryl	Yaeger
10	Marina	Zlatkina
11	Ben	Wiedermann
12	Jonathan	Liperi

Fig. 19.26 Table **Authors** after a **DELETE** operation to remove a record.

19.5 ADO.NET Object Model

The ADO.NET object model provides an API for accessing database systems programmatically. ADO.NET was created for the .NET framework and is the next generation of *ActiveX Data Objects*™ (ADO), which was designed to interact with Microsoft's *Component Object Model*™ (COM) framework.

The primary namespaces for ADO.NET are ***System.Data***, ***System.Data.OleDb*** and ***System.Data.SqlClient***. These namespaces contain classes for working with databases and other types of datasources (e.g., XML files). The **System.Data** namespace is the root namespace for the ADO.NET API. Namespaces **System.Data.OleDb** and **System.Data.SqlClient** contain classes that enable programs to connect with and modify datasources. The **System.Data.OleDb** namespace contains classes that are designed to work with any datasource, whereas the **System.Data.SqlClient** namespace contains classes that are optimized to work with Microsoft SQL Server 2000 databases.

Class ***System.Data.DataSet***, which consists of a set of **DataTable**s and relationships among those **DataTable**s, represents a *cache* of data—data that a program stores temporarily in local memory. The structure of a **DataSet** mimics the structure of a relational database. An advantage of using class **DataSet** is that it is *disconnected*—the program does not need a persistent connection to the datasource to work with data in a **DataSet**. The program connects to the datasource only to populate the **DataSet** initially and to store any changes made in the **DataSet**. Hence, no active, permanent connection to the datasource is required.

Class ***OleDbConnection*** of namespace **System.Data.OleDb** represents a connection to a datasource. Class ***OleDbDataAdapter*** connects to a datasource using an instance of class **OleDbConnection** and can populate **DataSet**s with data from a datasource. We discuss the details of creating and populating **DataSet**s later in this chapter.

Class ***OleDbCommand*** of namespace **System.Data.OleDb** represents an arbitrary SQL command to be executed on a datasource. A program can use instances of class **OleDbCommand** to manipulate a datasource through an **OleDbConnection**. The active connection to the datasource must be closed explicitly once no further changes are to be made. Unlike **DataSet**s, **OleDbCommand** objects do not cache data in local memory.

19.6 Programming with ADO.NET: Extracting Information from a DBMS

In this section, we present two examples that introduce how to connect to a database, query the database and display the results of the query. The database used in these examples is the Microsoft Access **Books** database. It can be found in the project directory for the application of Fig. 19.27. Each program must specify the location of this database on the computer's hard drive. When executing these examples on your computer, be sure to update this location in each program. For example, in Fig. 19.27, lines 69–78 must be changed so that they specify the correct location of the database file before executing the program on your computer.

19.6.1 Connecting to and Querying an Access Data Source

The first example (Fig. 19.27) performs a simple query on the **Books** database that retrieves the entire **Authors** table and displays the data in a ***DataGrid*** (a convenient **System.Windows.Forms** component class that can display a datasource in a GUI). The program illustrates connecting to the database, querying the database and displaying the results in a **DataGrid**. The following discussion presents the key aspects of the program. [*Note*: We present all of Visual Studio's auto-generated code in Fig. 19.27. We include this code to show exactly what Visual Studio generates for the example in this section.]

```
1  // Fig. 19.16: TableDisplay.cs
2  // Displays data from a database table.
3
4  using System;
5  using System.Drawing;
6  using System.Collections;
```

Fig. 19.27 How to access and display a database's data (part 1 of 7).

```
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;

public class TableDisplay : System.Windows.Forms.Form
{
   private System.Data.OleDb.OleDbConnection oleDbConnection1;
   private System.Data.DataSet dataSet1;
   private System.Data.OleDb.OleDbDataAdapter oleDbDataAdapter1;
   private System.Data.OleDb.OleDbCommand oleDbSelectCommand1;
   private System.Data.OleDb.OleDbCommand oleDbInsertCommand1;
   private System.Data.OleDb.OleDbCommand oleDbUpdateCommand1;
   private System.Data.OleDb.OleDbCommand oleDbDeleteCommand1;
   private System.Windows.Forms.DataGrid dataGrid1;
   private System.ComponentModel.Container components = null;

   public TableDisplay()
   {
      InitializeComponent();

      // Fill dataSet1 with data
      oleDbDataAdapter1.Fill( dataSet1, "Authors" );

      // Bind data in Users table in dataSet1 to dataGrid1
      dataGrid1.SetDataBinding( dataSet1, "Authors" );
   }

   private void InitializeComponent()
   {
      this.oleDbConnection1 =
         new System.Data.OleDb.OleDbConnection();
      this.dataSet1 = new System.Data.DataSet();
      this.oleDbDataAdapter1 =
         new System.Data.OleDb.OleDbDataAdapter();
      this.oleDbSelectCommand1 =
         new System.Data.OleDb.OleDbCommand();
      this.oleDbInsertCommand1 =
         new System.Data.OleDb.OleDbCommand();
      this.oleDbUpdateCommand1 =
         new System.Data.OleDb.OleDbCommand();
      this.oleDbDeleteCommand1 =
         new System.Data.OleDb.OleDbCommand();
      this.dataGrid1 = new System.Windows.Forms.DataGrid();
      ( ( System.ComponentModel.ISupportInitialize )
          ( this.dataSet1 ) ).BeginInit();
      ( ( System.ComponentModel.ISupportInitialize )
          ( this.dataGrid1 ) ).BeginInit();
      this.SuspendLayout();

      //
      // oleDbConnection1
      //
      this.oleDbConnection1.ConnectionString =
         @"Provider=Microsoft.Jet.OLEDB.4.0;Password="""";
```

Fig. 19.27 How to access and display a database's data (part 2 of 7).

```
               User ID=Admin;Data Source=C:\Documents
               and Settings\david\Desktop
               \mod\ch19\beta2versions\data\
               Books.mdb;Mode=ReadWrite;
               Extended Properties="""";
               Jet OLEDB:System database="""";
               Jet OLEDB:Registry Path="""";
               Jet OLEDB:Database Password="""";
               Jet OLEDB:Engine Type=5;
               Jet OLEDB:Database Locking Mode=1;
               Jet OLEDB:Global Partial Bulk Ops=2;
               Jet OLEDB:Global Bulk Transactions=1;
               Jet OLEDB:New Database Password="""";
               Jet OLEDB:Create System Database=False;
               Jet OLEDB:Encrypt Database=False;
               Jet OLEDB:Don't Copy Locale on Compact=False;
               Jet OLEDB:Compact Without Replica Repair=False;
               Jet OLEDB:SFP=False";

         //
         // dataSet1
         //
         this.dataSet1.DataSetName = "NewDataSet";
         this.dataSet1.Locale =
            new System.Globalization.CultureInfo( "en-US" );

         //
         // oleDbDataAdapter1
         //
         this.oleDbDataAdapter1.DeleteCommand =
            this.oleDbDeleteCommand1;
         this.oleDbDataAdapter1.InsertCommand =
            this.oleDbInsertCommand1;
         this.oleDbDataAdapter1.SelectCommand =
            this.oleDbSelectCommand1;
         this.oleDbDataAdapter1.TableMappings.AddRange(
            new System.Data.Common.DataTableMapping[] {
               new System.Data.Common.DataTableMapping(
                  "Table", "Authors",
                  new System.Data.Common.DataColumnMapping[] {
                     new System.Data.Common.DataColumnMapping
                        ( "Number", "Number" ),
                     new System.Data.Common.DataColumnMapping
                        ( "First", "First" ),
                     new System.Data.Common.DataColumnMapping
                        ( "Last", "Last" ) } ) } ) ;
         this.oleDbDataAdapter1.UpdateCommand =
            this.oleDbUpdateCommand1;

         //
         // oleDbSelectCommand1
         //
         this.oleDbSelectCommand1.CommandText =
            "SELECT First, Last, [Number] FROM Authors";
```

Fig. 19.27 How to access and display a database's data (part 3 of 7).

```
         this.oleDbSelectCommand1.Connection = this.oleDbConnection1;

         //
         // oleDbInsertCommand1
         //
         this.oleDbInsertCommand1.CommandText =
            "INSERT INTO Authors( First, Last, [ Number ] ) " +
            "VALUES ( ?, ?, ? ); SELECT First, Last, [ Number ]" +
            " FROM Authors WHERE ( [ Number ] = ? )";
         this.oleDbInsertCommand1.Connection = this.oleDbConnection1;
         this.oleDbInsertCommand1.Parameters.Add(
            new System.Data.OleDb.OleDbParameter(
            "First", System.Data.OleDb.OleDbType.Char, 50,
            System.Data.ParameterDirection.Input, false,
           ( ( System.Byte ) ( 0 ) ),( ( System.Byte ) ( 0 ) ), "First",
            System.Data.DataRowVersion.Current, null ) );
         this.oleDbInsertCommand1.Parameters.Add(
            new System.Data.OleDb.OleDbParameter(
            "Last", System.Data.OleDb.OleDbType.Char, 50,
            System.Data.ParameterDirection.Input, false,
           ( ( System.Byte ) ( 0 ) ),( ( System.Byte ) ( 0 ) ), "Last",
            System.Data.DataRowVersion.Current, null ) );
         this.oleDbInsertCommand1.Parameters.Add(
            new System.Data.OleDb.OleDbParameter(
            "Number", System.Data.OleDb.OleDbType.Numeric, 0,
            System.Data.ParameterDirection.Input, false,
          ( ( System.Byte ) ( 10 ) ),( ( System.Byte ) ( 0 ) ), "Number",
            System.Data.DataRowVersion.Current, null ) );
         this.oleDbInsertCommand1.Parameters.Add(
            new System.Data.OleDb.OleDbParameter(
            "Select_Number", System.Data.OleDb.OleDbType.Numeric, 0,
            System.Data.ParameterDirection.Input, false,
          ( ( System.Byte ) ( 10 ) ),( ( System.Byte ) ( 0 ) ), "Number",
            System.Data.DataRowVersion.Current, null ) );

         //
         // oleDbUpdateCommand1
         //
         this.oleDbUpdateCommand1.CommandText =
            "UPDATE Authors SET First = ?," +
           "Last = ?, [ Number ] = ? WHERE ( [ Number] = ? ) AND ( Fi" +
            "rst = ? ) AND ( Last = ? ); " +
            "SELECT First, Last, [ Number ] FROM " +
            "Authors WHERE ( [ Numbe" + "r ] = ?) ";
         this.oleDbUpdateCommand1.Connection = this.oleDbConnection1;
         this.oleDbUpdateCommand1.Parameters.Add(
            new System.Data.OleDb.OleDbParameter(
            "First", System.Data.OleDb.OleDbType.Char, 50,
            System.Data.ParameterDirection.Input, false,
           ( ( System.Byte ) ( 0 ) ),( ( System.Byte ) ( 0 ) ), "First",
            System.Data.DataRowVersion.Current, null ) );
         this.oleDbUpdateCommand1.Parameters.Add(
            new System.Data.OleDb.OleDbParameter(
            "Last", System.Data.OleDb.OleDbType.Char, 50,
```

Fig. 19.27 How to access and display a database's data (part 4 of 7).

```
            System.Data.ParameterDirection.Input, false,
          ( ( System.Byte ) ( 0 ) ),( ( System.Byte ) ( 0 ) ), "Last",
            System.Data.DataRowVersion.Current, null ) );
         this.oleDbUpdateCommand1.Parameters.Add(
            new System.Data.OleDb.OleDbParameter(
            "Number", System.Data.OleDb.OleDbType.Numeric, 0,
            System.Data.ParameterDirection.Input, false,
         ( ( System.Byte ) ( 10 ) ),( ( System.Byte ) ( 0 ) ), "Number",
            System.Data.DataRowVersion.Current, null ) );
         this.oleDbUpdateCommand1.Parameters.Add(
            new System.Data.OleDb.OleDbParameter(
           "Original_Number", System.Data.OleDb.OleDbType.Numeric, 0,
            System.Data.ParameterDirection.Input, false,
         ( ( System.Byte ) ( 10 ) ),( ( System.Byte ) ( 0 ) ), "Number",
            System.Data.DataRowVersion.Original, null ) );
         this.oleDbUpdateCommand1.Parameters.Add(
            new System.Data.OleDb.OleDbParameter(
            "Original_First", System.Data.OleDb.OleDbType.Char, 50,
            System.Data.ParameterDirection.Input, false,
          ( ( System.Byte ) ( 0 ) ),( ( System.Byte ) ( 0 ) ), "First",
            System.Data.DataRowVersion.Original, null ) );
         this.oleDbUpdateCommand1.Parameters.Add(
            new System.Data.OleDb.OleDbParameter(
            "Original_Last", System.Data.OleDb.OleDbType.Char, 50,
            System.Data.ParameterDirection.Input, false,
          ( ( System.Byte ) ( 0 ) ),( ( System.Byte ) ( 0 ) ), "Last",
            System.Data.DataRowVersion.Original, null ) );
         this.oleDbUpdateCommand1.Parameters.Add(
            new System.Data.OleDb.OleDbParameter(
            "Select_Number", System.Data.OleDb.OleDbType.Numeric, 0,
            System.Data.ParameterDirection.Input, false,
         ( ( System.Byte ) ( 10 ) ),( ( System.Byte ) ( 0 ) ), "Number",
            System.Data.DataRowVersion.Current, null ) );

         //
         // oleDbDeleteCommand1
         //
         this.oleDbDeleteCommand1.CommandText =
            "DELETE FROM Authors WHERE ( [ Number ] = ? ) " +
            "AND ( First = ? ) AND ( Last = ? )";
         this.oleDbDeleteCommand1.Connection = this.oleDbConnection1;
         this.oleDbDeleteCommand1.Parameters.Add(
            new System.Data.OleDb.OleDbParameter(
            "Number", System.Data.OleDb.OleDbType.Numeric, 0,
            System.Data.ParameterDirection.Input, false,
         ( ( System.Byte ) ( 10 ) ),( ( System.Byte ) ( 0 ) ), "Number",
            System.Data.DataRowVersion.Original, null ) );
         this.oleDbDeleteCommand1.Parameters.Add(
            new System.Data.OleDb.OleDbParameter(
            "First", System.Data.OleDb.OleDbType.Char, 50,
            System.Data.ParameterDirection.Input, false,
          ( ( System.Byte ) ( 0 ) ),( ( System.Byte ) ( 0 ) ), "First",
            System.Data.DataRowVersion.Original, null ) );
         this.oleDbDeleteCommand1.Parameters.Add(
```

Fig. 19.27 How to access and display a database's data (part 5 of 7).

```
            new System.Data.OleDb.OleDbParameter(
            "Last", System.Data.OleDb.OleDbType.Char, 50,
            System.Data.ParameterDirection.Input, false,
           ( ( System.Byte ) ( 0 ) ),( ( System.Byte ) ( 0 ) ), "Last",
            System.Data.DataRowVersion.Original, null ) );

         //
         // dataGrid1
         //
         this.dataGrid1.DataMember = "";
        this.dataGrid1.Location = new System.Drawing.Point( 16, 16 ) ;
         this.dataGrid1.Name = "dataGrid1";
         this.dataGrid1.Size = new System.Drawing.Size( 264, 248 );
         this.dataGrid1.TabIndex = 0;

         //
         // TableDisplay
         //
         this.AutoScaleBaseSize = new System.Drawing.Size( 5, 13 );
         this.ClientSize = new System.Drawing.Size( 292, 273 );
         this.Controls.AddRange( new System.Windows.Forms.Control[] {
            this.dataGrid1 } );
         this.Name = "TableDisplay";
         this.Text = "TableDisplay";
         ( ( System.ComponentModel.ISupportInitialize )
            ( this.dataSet1 ) ).EndInit();
         ( ( System.ComponentModel.ISupportInitialize )
            ( this.dataGrid1 ) ).EndInit();
         this.ResumeLayout( false );

      }   // end of InitializeComponent

      [STAThread]
      static void Main()
      {
         Application.Run( new TableDisplay() );
      }
   }
```

Fig. 19.27 How to access and display a database's data (part 6 of 7).

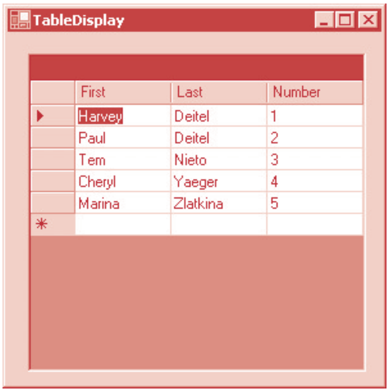

Fig. 19.27 How to access and display a database's data (part 7 of 7).

In this example, we use an Access database. To register the **Books** database as a data-source, right click the **Data Connections** node in the **Server Explorer** and double click **<Add Connection>**. In the **Provider** tab of the window that appears, choose "**Microsoft Jet 4.0 OLE DB Provider**," which is the driver for Access databases. In the **Connection** tab, click the **...** button to the right of the textbox for the database name, which opens the **Select Access Database** window. Go to the appropriate folder, select the **Books** database then click **OK**. Now this database is listed as a connection in the **Server Explorer**. Drag the database node onto the Windows Form. This creates an **OleDbConnection** to the source, which the Windows Form designer shows as **oleDbConnection1**.

Next, drag an ***OleDbDataAdapter*** from the **Toolbox**'s ***Data*** subheading onto the Windows Form designer. This displays the **Data Adapter Configuration Wizard** for configuring the **OleDbDataAdapter** instance with a custom query for populating a **DataSet**. Click **Next** to select a connection to use. Select the connection created in the previous step from the drop-down list and click **Next**. The next screen allows us to choose how the **OleDbDataAdapter** should access the database. Keep the default **Use SQL Statement** option and click **Next**. Click the "**Query Builder**" button, then select the **Authors** table from the "**Add**" menu and then **Close** that menu. Place a check mark in the "***All Columns**" box from the small "**Authors**" window. Notice how that particular window lists all columns of the **Authors** table.

Next, create a **DataSet** to store the query results. To do so, drag **DataSet** from the **Data** tab in the **Toolbox**. This displays the **Add DataSet** window. Choose the "**Untyped DataSet (no schema)**" since the query with which we populate the **DataSet** dictates the **DataSet**'s *schema*, or structure.

Figure 19.27 shows all of the auto-generated code. Normally, we omit this code from the chapter since this code consists solely of GUI components. In this case, however, there is database functionality that needs to be discussed. Furthermore, we have left the default naming conventions of Visual Studio in this example, to show exactly what auto-generated code Visual Studio creates. Normally, we would change these names to conform to our programming conventions and style.

Good Programming Practice 19.2

Use clear, descriptive variable names in code. This makes programs easier to understand.

Lines 68-79 initialize the `oleDbConnection` for this program. The `ConnectionString` property specifies the path to the database file on the computer's hard drive.

An instance of class `OleDbDataAdapter` populates the `DataSet` in this example with data from the `Books` database. The instance properties *`DeleteCommand`* (lines 90–91), *`InsertCommand`* (lines 92–93), *`SelectCommand`* (lines 94–95) and *`UpdateCommand`* (lines 107–108) are `OleDbCommand` objects that specify how the `OleDbDataAdapter` deletes, inserts, selects and updates data in the database.

Each `OleDbCommand` object must have an `OleDbConnection` with which the `OleDbCommand` can communicate with the database. Instance property `Connection` is set to the `OleDbConnection` to the `Books` database. For `oleDbUpdateCommand1`, line 159 sets the `Connection` property, and lines 153–158 set the `CommandText`.

Although Visual Studio .NET auto-generates most of this program's code, we manually enter code in the `TableDisplay` constructor (lines 23–32) for populating `dataSet1` using an `OleDbDataAdapter`. Line 28 uses `OleDbDataAdapter` method *`Fill`* to retrieve information from the database associated with the `OleDbConnection`, placing it in the `DataSet` provided as an argument. The second argument to this method is a string that specifies the name of the table in the database from which to `Fill` the `DataSet`.

Line 31 invokes `DataGrid` method `SetDataBinding` to bind the `DataGrid` to a data source. The first argument is the `DataSet`—in this case, `dataSet1`—whose data the `DataGrid` should display. The second argument is a `string` representing the name of the table within the data source we want to bind to the `DataGrid`. Once this line executes, the `DataGrid` is filled with the information in the `DataSet`—the number of rows and columns is automatically set based on the information in `dataSet1`. Notice that the columns are automatically given appropriate names, and as the second screen capture in Fig. 19.27 demonstrates, clicking any column sorts the rows by that column either in ascending or descending order.

19.6.2 Querying the Books Database

The code example in Fig. 19.30 shows how to execute SQL `SELECT` statements on a database and display the results. Although Fig. 19.30 uses only `SELECT` statements to query the data, it could be used to execute many different SQL statements with a few minor modifications.

Method `submitButton_Click` is the heart of this program. When the program invokes this event handler in response to a button click, lines 47–48 assign the `SELECT` query that the user typed in `queryTextBox` as the value of the `OleDbDataAdapter`'s `SelectCommand` property. This `string` is parsed into an SQL query and executed on the database with the `OleDbDataAdapter`'s `Fill` method (line 55). This method, as discussed in the previous section, places the data from the database into `dataSet1`.

Common Programming Error 19.8

If a **`DataSet`** *has already been* **`Fill`***ed at least once, forgetting to call a* **`DataSet`***'s* **`Clear`** *method (line 61) before using the* **`Fill`** *method subsequent times will lead to logic errors.*

To display, or redisplay, contents in the **`DataGrid`**, use method **`SetDataBinding`**. Again, the first argument is the datasource to be displayed in the table—a **`DataSet`** in this case. The second argument is the **`string`** name of the member of the first argument to be displayed (line 58). Try entering your own queries in the text area and pressing the **Submit Query** button to execute the query.

```
1  // Fig. 19.19: DisplayQueryResults.cs
2  // Displays the contents of the authors database.
3
4  using System;
5  using System.Drawing;
6  using System.Collections;
7  using System.ComponentModel;
8  using System.Windows.Forms;
9  using System.Data;
10
11 public class DisplayQueryResults : System.Windows.Forms.Form
12 {
13    private System.Data.OleDb.OleDbConnection oleDbConnection1;
14    private System.Data.DataSet dataSet1;
15    private System.Data.OleDb.OleDbDataAdapter oleDbDataAdapter1;
16    private System.Data.OleDb.OleDbCommand oleDbSelectCommand1;
17    private System.Data.OleDb.OleDbCommand oleDbInsertCommand1;
18    private System.Data.OleDb.OleDbCommand oleDbUpdateCommand1;
19    private System.Data.OleDb.OleDbCommand oleDbDeleteCommand1;
20    private System.Windows.Forms.TextBox queryTextBox;
21    private System.Windows.Forms.Button submitButton;
22    private System.Windows.Forms.DataGrid dataGrid1;
23    private System.ComponentModel.Container components = null;
24
25    public DisplayQueryResults()
26    {
27
28       InitializeComponent();
29    }
30
31    // Visual Studio.NET generated code
32
33    [STAThread]
34    static void Main()
35    {
36       Application.Run( new DisplayQueryResults() );
37    }
38
39    // perform SQL query on data
40    private void submitButton_Click( object sender,
41             System.EventArgs e)
```

Fig. 19.28 Execute SQL statements on a database (part 1 of 3).

```
   {
      try
      {
         // set the text of the SQL query to what the user typed
         // in
         oleDbDataAdapter1.SelectCommand.CommandText =
            queryTextBox.Text;

         // clear the DataSet from the previous operation
         dataSet1.Clear();

         // Fill the data set with the information that results
         // from the SQL query
         oleDbDataAdapter1.Fill( dataSet1, "Authors" );

         // Bind the DataGrid to the contents of the DatSet
         dataGrid1.SetDataBinding( dataSet1, "Authors" );
      }

      catch ( System.Data.OleDb.OleDbException ex )
      {
         MessageBox.Show( "Invalid query" );
      }

   } // end of submitButton_Click
}
```

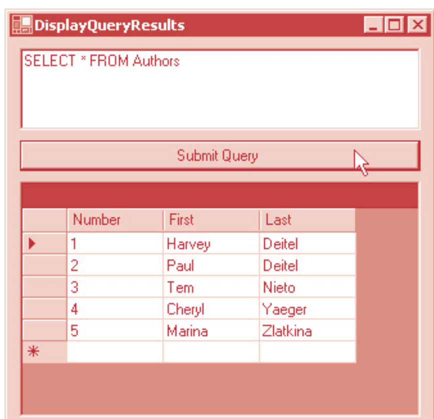

Fig. 19.28 Execute SQL statements on a database (part 2 of 3).

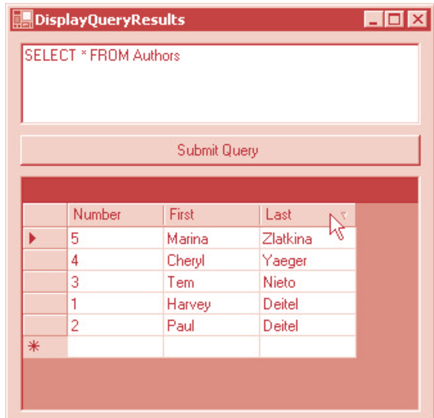

Fig. 19.28 Execute SQL statements on a database (part 3 of 3).

19.7 Programming with ADO.NET: Modifying a DBMS

Our next example implements a simple address-book application that enables the user to insert, locate and update the Microsoft Access database **Addressbook**.

The **Addressbook** application (Fig. 19.29) provides a GUI to execute SQL statements on the database. Earlier in the chapter, we saw examples that showed how to use **SELECT** statements to query a database. Here, that same functionality is provided.

```
// Fig. 19.20: AddressBook.cs
// Using SQL statements to manipulate a database.

using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;

public class AddressBook : System.Windows.Forms.Form
{
   private System.Windows.Forms.TextBox faxTextBox;
   private System.Windows.Forms.TextBox homeTextBox;
   private System.Windows.Forms.TextBox firstTextBox;
   private System.Windows.Forms.TextBox stateTextBox;
   private System.Windows.Forms.TextBox idTextBox;
   private System.Windows.Forms.TextBox lastTextBox;
   private System.Windows.Forms.TextBox postalTextBox;
   private System.Windows.Forms.TextBox addressTextBox;
   private System.Windows.Forms.TextBox cityTextBox;
```

Fig. 19.29 How to modify a database (part 1 of 10).

```
      private System.Windows.Forms.TextBox countryTextBox;
      private System.Windows.Forms.TextBox emailTextBox;
      private System.Data.DataSet dataSet1;
      private System.Data.OleDb.OleDbDataAdapter oleDbDataAdapter1;
      private System.Data.OleDb.OleDbCommand oleDbSelectCommand1;
      private System.Data.OleDb.OleDbCommand oleDbInsertCommand1;
      private System.Data.OleDb.OleDbCommand oleDbUpdateCommand1;
      private System.Data.OleDb.OleDbCommand oleDbDeleteCommand1;
      private System.Data.OleDb.OleDbConnection oleDbConnection1;
      private System.Windows.Forms.TextBox statusTextBox;
      private System.Windows.Forms.Label addressLabel;
      private System.Windows.Forms.Label cityLabel;
      private System.Windows.Forms.Label stateLabel;
      private System.Windows.Forms.Label idLabel;
      private System.Windows.Forms.Label firstLabel;
      private System.Windows.Forms.Label lastLabel;
      private System.Windows.Forms.Label postalLabel;
      private System.Windows.Forms.Label countryLabel;
      private System.Windows.Forms.Label emailLabel;
      private System.Windows.Forms.Button clearButton;
      private System.Windows.Forms.Button helpButton;
      private System.Windows.Forms.Button findButton;
      private System.Windows.Forms.Button addButton;
      private System.Windows.Forms.Button updateButton;
      private System.Windows.Forms.Label faxLabel;
      private System.Windows.Forms.Label homeLabel;
      private System.ComponentModel.Container components = null;

      public AddressBook()
      {
         InitializeComponent();
         oleDbConnection1.Open();
      }

      // Visual Studio.NET generated code

      [STAThread]
      static void Main()
      {
         Application.Run( new AddressBook() );
      }

      private void findButton_Click( object sender,
            System.EventArgs e )
      {
         try
         {
            if ( lastTextBox.Text != "" )
            {
               // clear the DataSet from the last operation
               dataSet1.Clear();

               // create SQL query to find the contact with the
               // specified last name
```

Fig. 19.29 How to modify a database (part 2 of 10).

```
            oleDbDataAdapter1.SelectCommand.CommandText =
               "SELECT * FROM addresses WHERE lastname = '" +
               lastTextBox.Text + "'";

            // fill dataSet1 with the rows resulting from the
            // query
            oleDbDataAdapter1.Fill( dataSet1 );

            // display information
            Display( dataSet1 );
            statusTextBox.Text += "\r\nQuery successful\r\n";
         }
         else
            lastTextBox.Text =
               "Enter last name here then press Find";
      }

      catch ( System.Data.OleDb.OleDbException ex )
      {
         Console.WriteLine( ex.StackTrace );
         statusTextBox.Text += ex.ToString();
      }

      catch ( InvalidOperationException ioe )
      {
         MessageBox.Show( ioe.Message );
      }

   } // end of findButton_Click

   private void addButton_Click( object sender, System.EventArgs e )
   {
      try
      {
         if ( lastTextBox.Text != "" && firstTextBox.Text != "" )
         {
            // create the SQL query to insert a row
            oleDbDataAdapter1.InsertCommand.CommandText =
               "INSERT INTO addresses (" +
               "firstname, lastname, address, city, " +
               "stateorprovince, postalcode, country, " +
               "emailaddress, homephone, faxnumber" +
               ") VALUES ('" +
               firstTextBox.Text + "', '" +
               lastTextBox.Text + "', '" +
               addressTextBox.Text + "', '" +
               cityTextBox.Text + "', '" +
               stateTextBox.Text + "', '" +
               postalTextBox.Text + "', '" +
               countryTextBox.Text + "', '" +
               emailTextBox.Text + "', '" +
               homeTextBox.Text + "', '" +
               faxTextBox.Text + "')";

```

Fig. 19.29 How to modify a database (part 3 of 10).

```
            // notify the user the query is being sent
            statusTextBox.Text += "\r\nSending query: " +
               oleDbDataAdapter1.InsertCommand.CommandText +
               "\r\n" ;

            // send query
            oleDbDataAdapter1.InsertCommand.ExecuteNonQuery();

            statusTextBox.Text += "\r\nQuery successful\r\n";
         }
         else
            statusTextBox.Text += "\r\nEnter at least first " +
               "and last name then press Add\r\n";
      }

      catch ( System.Data.OleDb.OleDbException ex )
      {
         Console.WriteLine( ex.StackTrace );
         statusTextBox.Text += ex.ToString();
      }

   }  // end of addButton_Click

   private void updateButton_Click( object sender,
      System.EventArgs e )
   {
      try
      {
         // make sure the user has already found the record
         // he or she wishes to update
         if ( idTextBox.Text != "" )
         {
            // set the SQL query to update all the fields in
            // the table where the id number matches the id
            // in idTextBox
            oleDbDataAdapter1.UpdateCommand.CommandText =
               "UPDATE addresses SET " +
               "firstname ='" + firstTextBox.Text +
               "', lastname='" + lastTextBox.Text +
               "', address='" + addressTextBox.Text +
               "', city='" + cityTextBox.Text +
               "', stateorprovince='" + stateTextBox.Text +
               "', postalcode='" + postalTextBox.Text +
               "', country='" + countryTextBox.Text +
               "', emailaddress='" + emailTextBox.Text +
               "', homephone='" + homeTextBox.Text +
               "', faxnumber='" + faxTextBox.Text +
               "' WHERE id=" + idTextBox.Text;

            // notify the user the query is being set
            statusTextBox.Text += "\r\nSending query: " +
               oleDbDataAdapter1.UpdateCommand.CommandText +
               "\r\n";

```

Fig. 19.29 How to modify a database (part 4 of 10).

```
            // execute query
            oleDbDataAdapter1.UpdateCommand.ExecuteNonQuery();

            statusTextBox.Text += "\r\nQuery successful\r\n";
         }
         else
            statusTextBox.Text += "\r\nYou may only update " +
               "an existing record. Use Find to locate the" +
               "record, then modify the information and " +
               "press Update.\r\n";
      }

      catch ( System.Data.OleDb.OleDbException ex )
      {
         Console.WriteLine( ex.StackTrace );
         statusTextBox.Text += ex.ToString();
      }

   } // end of updateButton_Click

   private void clearButton_Click( object sender,
         System.EventArgs e )
   {
      idTextBox.Clear();
      ClearTextBoxes();
   }

   private void helpButton_Click( object sender,
         System.EventArgs e )
   {
      statusTextBox.AppendText(
         "\r\nClick Find to locate a record\r\n" +
         "Click Add to insert a new record.\r\n" +
         "Click Update to update the information in a record "
         + "\r\nClick Clear to empty the textboxes" );
   }

   public void Display( DataSet dataSet )
   {
      try
      {
         // get the first DataTable - there will always be one
         DataTable dataTable = dataSet.Tables[ 0 ];

         if ( dataTable.Rows.Count != 0 )
         {
            int recordNumber = ( int ) dataTable.Rows[ 0 ][ 0 ];

            idTextBox.Text = recordNumber.ToString();
            firstTextBox.Text =
               ( string ) dataTable.Rows[ 0 ][ 1 ];
            lastTextBox.Text =
               ( string ) dataTable.Rows[ 0 ][ 2 ];
            addressTextBox.Text =
```

Fig. 19.29 How to modify a database (part 5 of 10).

```
                  ( string ) dataTable.Rows[ 0 ][ 3 ];
               cityTextBox.Text =
                  ( string ) dataTable.Rows[ 0 ][ 4 ];
               stateTextBox.Text =
                  ( string ) dataTable.Rows[ 0 ][ 5 ];
               postalTextBox.Text =
                  ( string ) dataTable.Rows[ 0 ][ 6 ];
               countryTextBox.Text =
                  ( string ) dataTable.Rows[ 0 ][ 7 ];
               emailTextBox.Text =
                  ( string ) dataTable.Rows[ 0 ][ 8 ];
               homeTextBox.Text =
                  ( string ) dataTable.Rows[ 0 ][ 9 ];
               faxTextBox.Text =
                  ( string ) dataTable.Rows[ 0 ][ 10 ];
            }

            else
               statusTextBox.Text += "\r\nNo record found\r\n";
         }

         catch( System.Data.OleDb.OleDbException ex )
         {
            Console.WriteLine( ex.StackTrace );
            statusTextBox.Text += ex.ToString();
         }

      } // end of Display

      public void ClearTextBoxes()
      {
         firstTextBox.Clear();
         lastTextBox.Clear();
         addressTextBox.Clear();
         cityTextBox.Clear();
         stateTextBox.Clear();
         postalTextBox.Clear();
         countryTextBox.Clear();
         emailTextBox.Clear();
         homeTextBox.Clear();
         faxTextBox.Clear();
      }
   }
```

Fig. 19.29 How to modify a database (part 6 of 10).

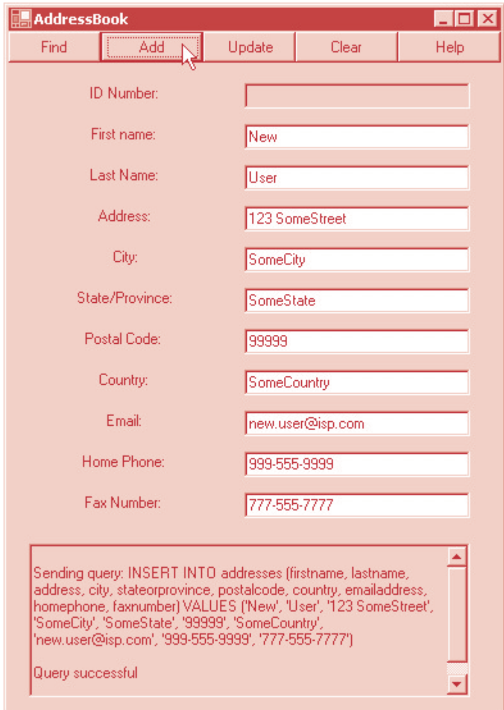

Fig. 19.29 How to modify a database (part 7 of 10).

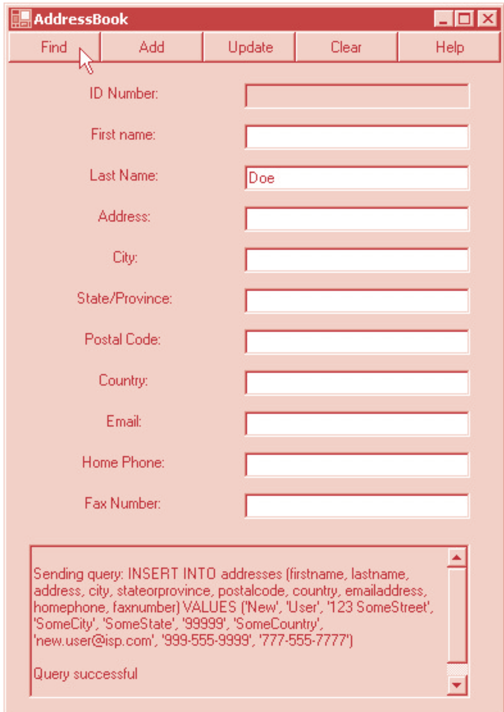

Fig. 19.29 How to modify a database (part 8 of 10).

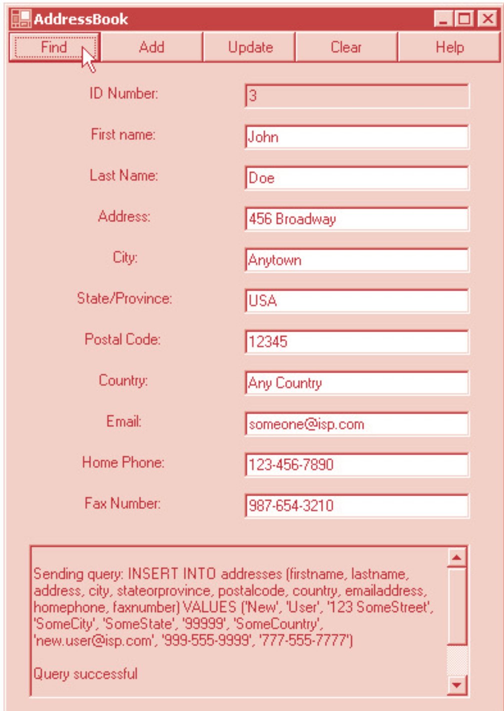

Fig. 19.29 How to modify a database (part 9 of 10).

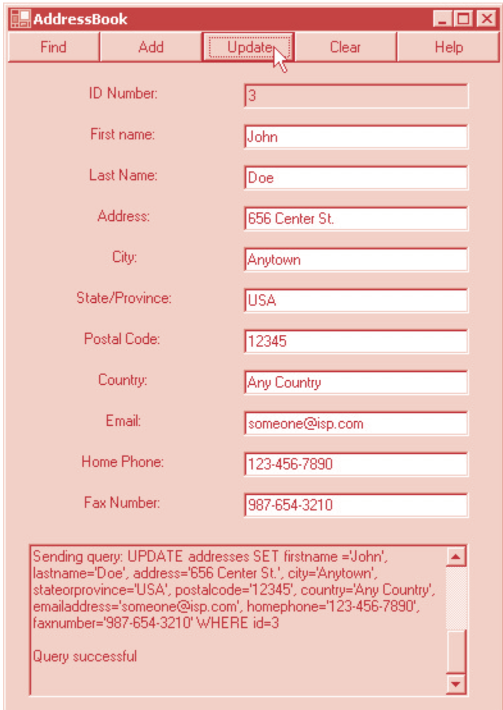

Fig. 19.29 How to modify a database (part 10 of 10).

Event handler **`findButton_Click`** performs the **`SELECT`** query on the database for the record associated with the **`string`** in **`lastTextBox`**. This represents the last-name of the person whose record the user wishes to retrieve. Line 72 invokes method **`Clear`** of class **`DataSet`** to empty the **`DataSet`** of any prior data. Lines 76–78 modify the text of the SQL query to perform the appropriate **`SELECT`** operation. This statement is executed by the **`OleDbDataAdapter`** method **`Fill`** (line 82). Notice how a different overload of that method has been used in this situation. Only the **`DataSet`** to be filled is passed as an argument. Finally, the **`TextBox`**es are updated with a call to method **`Display`** (line 85).

Methods **`addButton_Click`** and **`updateButton_Click`** perform **`INSERT`** and **`UPDATE`** operations, respectively. Each method uses members of class **`OleDbCommand`** to perform operations on a database. The instance properties **`InsertCommand`** and **`UpdateCommand`** of class **`OleDbDataAdapter`** are instances of class **`OleDbCommand`**.

Property **`CommandText`** of class **`OleDbCommand`** is a **`string`** that represents the SQL statement that the **`OleDbCommand`** object executes. Method **`addButton_Click`** sets this property of **`InsertCommand`** to execute the appropriate **`INSERT`** statement on the database (lines 113–128). Method **`updateButton_Click`** sets this property of

UpdateCommand to execute the appropriate **UPDATE** statement on the database (lines 165–177).

Method ***ExecuteNonQuery*** of class **OleDbCommand** performs the action specified by **CommandText**. Hence, the **INSERT** statement defined by **oleDbDataAdapter1.InsertCommand.CommandText** in event handler **addButton_Click** is executed when line 136 invokes method **oleDbDataAdapter1.InsertCommand.ExecuteNonQuery**. Similarly, the **UPDATE** statement defined by **oleDbDataAdapter1.DeleteCommand.CommandText** in event handler **updateButton_Click** is executed by **oleDbDataAdapter1.UpdateCommand.ExecuteNonQuery** (line 185).

The application's **Help** button prints instructions in the console at the bottom of the application window (lines 214–218). The event handler for this button is **helpButton_Click**. The **Clear** button clears the text out of the **TextBox**es. This event handler is defined in the method **clearButton_Click** and uses the helper function **ClearTextBoxes** (line 211).

19.8 Reading and Writing XML Files

A powerful feature of ADO.NET is its ability to convert data stored in a datasource to XML. Cclass **DataSet** of namespace **System.Data** provides methods ***WriteXml***, ***ReadXml*** and ***GetXml***, which enable developers to create XML documents from datasources and to convert data from XML into datasources. The application of Fig. 19.30 populates a **DataSet** with statistics about baseball players then writes the data to a files as XML. The application also displays the XML in a **TextBox**.

```
// Fig. 19.30 XMLWriter.cs
// Demonstrates generating XML from an ADO.NET DataSet

using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;

public class XMLWriter : System.Windows.Forms.Form
{
   private System.Data.OleDb.OleDbConnection baseballConnection;
   private System.Data.OleDb.OleDbDataAdapter playersDataAdapter;
   private System.Data.OleDb.OleDbCommand oleDbSelectCommand1;
   private System.Data.OleDb.OleDbCommand oleDbInsertCommand1;
   private System.Data.OleDb.OleDbCommand oleDbUpdateCommand1;
   private System.Data.OleDb.OleDbCommand oleDbDeleteCommand1;
   private System.Data.DataSet playersDataSet;
   private System.Windows.Forms.DataGrid playersDataGrid;
   private System.Windows.Forms.Button writeButton;
   private System.Windows.Forms.TextBox outputTextBox;
```

Fig. 19.30 Application that writes an XML representation of a **DataSet** to a file.

```
      private System.ComponentModel.Container components = null;

      public XMLWriter()
      {
         //
         // Required for Windows Form Designer support
         //
         InitializeComponent();

         // open database connection
         baseballConnection.Open();

         // fill DataSet with data from OleDbDataAdapter
         playersDataAdapter.Fill( playersDataSet, "Players" );

         // bind DataGrid to DataSet
         playersDataGrid.SetDataBinding( playersDataSet, "Players" );

      }

      // Visual Studio .NET-generated code

      // The main entry point for the application.
      [STAThread]
      static void Main()
      {
         Application.Run(new XMLWriter());
      }

      // write XML representation of DataSet when button clicked
     private void writeButton_Click(object sender, System.EventArgs e)
      {
         // write XML representation of DataSet to file
         playersDataSet.WriteXml( "Players.xml" );

         // display XML in TextBox
         outputTextBox.Text += "Writing the following XML:\n\n" +
            playersDataSet.GetXml() + "\n\n";

      }
   }
```

Fig. 19.30 Application that writes an XML representation of a **`DataSet`** to a file.

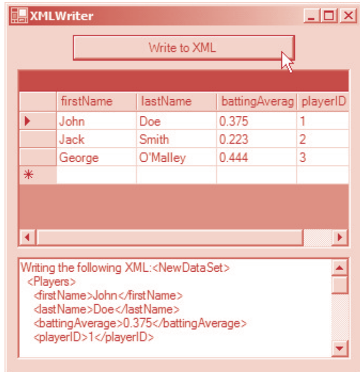

Fig. 19.30 Application that writes an XML representation of a **DataSet** to a file.

The **XMLWriter** constructor (lines 25-41) establishes a connection to the **Baseball** database on line 33. Line 36 uses method **Fill** of class **OleDbDataAdapter** to populate **playersDataSet** with data from the **Players** table in the **Baseball** database. Line 39 binds the **playersDataGrid** to **playersDataSet** to display the information to the user.

Method **writeButton_Click** defines the event handler for the **Write to XML** button. When the user clicks this button, line 56 invokes **DataSet** method **WriteXml**, which generates an XML representation of the data contained in the **DataSet** and writes the XML to the specified file. Figure 19.31 shows this XML representation. Each **Players** element represents a record in the **Players** table. The **firstName**, **lastName**, **battingAverage** and **playerID** elements correspond to the fields of the same names in the **Players** database table.

```
<?xml version="1.0" standalone="yes"?>
<NewDataSet>
   <Players>
      <firstName>John</firstName>
      <lastName>Doe</lastName>
      <battingAverage>0.375</battingAverage>
      <playerID>1</playerID>
   </Players>
   <Players>
      <firstName>Jack</firstName>
      <lastName>Smith</lastName>
      <battingAverage>0.223</battingAverage>
      <playerID>2</playerID>
   </Players>
   <Players>
      <firstName>George</firstName>
      <lastName>O'Malley</lastName>
      <battingAverage>0.444</battingAverage>
```

Fig. 19.31 XML document generated from **DataSet** in **XMLWriter**.

```
            <playerID>3</playerID>
        </Players>
    </NewDataSet>
```

Fig. 19.31 XML document generated from **`DataSet`** in **`XMLWriter`**.

SUMMARY

- A database is an integrated collection of data. A database management system (DBMS) provides mechanisms for storing and organizing data.
- Today's most popular database systems are relational databases.
- A language called Structured Query Language (SQL) is used almost universally with relational database systems to perform queries and manipulate data.
- A programming language connects to, and interacts with, relational databases via an interface—software that facilitates communications between a database management system and a program.
- C# programmers communicate with databases and manipulate their data using ADO.NET.
- A relational database is composed of tables. A row of a table is called a record.
- A primary key is a field that contains unique data that cannot be duplicated in other records.
- Each column of the table represents a different field (or attribute).
- The primary key can be composed of more than one column (or field) in the database.
- SQL provides a complete set of commands enabling programmers to define complex queries to select data from a table. The results of a query are commonly called result sets (or record sets).
- A one-to-many relationship between tables indicates that a record in one table can have many records in a separate table.
- A foreign key is a field for which every entry in one table has a unique value in another table and where the field in the other table is the primary key for that table.
- The simplest format of a **`SELECT`** query is

  `SELECT * FROM` *tableName*

  where the asterisk (**`*`**) indicates that all columns from *tableName* should be selected and *tableName* specifies the table in the database from which the data will be selected.
- To select specific fields from a table, replace the asterisk (**`*`**) with a comma-separated list of the field names to select.
- Programmers process result sets by knowing in advance the order of the fields in the result set. Specifying the field names to select guarantees that the fields are always returned in the specified order, even if the actual order of the fields in the database table(s) changes.
- The optional **`WHERE`** clause in a **`SELECT`** query specifies the selection criteria for the query. The simplest format of a **`SELECT`** query with selection criteria is

  `SELECT` *fieldName1*, *fieldName2*, … **`FROM`** *tableName* **`WHERE`** *criteria*

- The **`WHERE`** clause condition can contain operators **`<`**, **`>`**, **`<=`**, **`>=`**, **`=`**, **`<>`** and **`LIKE`**. Operator **`LIKE`** is used for 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.

- The results of a query can be arranged in ascending or descending order using the optional **ORDER BY** clause. The simplest form of an **ORDER BY** clause is

```
SELECT fieldName1, fieldName2, … FROM tableName ORDER BY field ASC
SELECT fieldName1, fieldName2, … FROM tableName ORDER BY field DESC
```

  where **ASC** specifies ascending order, **DESC** specifies descending order and *field* specifies the field on which the sort is based. The default sorting order is ascending, so **ASC** is optional.
- Multiple fields can be used for ordering purposes with an **ORDER BY** clause of the form

```
ORDER BY field1 sortingOrder, field2 sortingOrder, …
```

- The **WHERE** and **ORDER BY** clauses can be combined in one query.
- A join merges records from two or more tables by testing for matching values in a field that is common to both tables. The simplest format of a join is

```
SELECT fieldName1, fieldName2, …
   FROM table1, table2
   WHERE table1.fieldName = table2.fieldName
```

  in which the **WHERE** clause specifies the fields from each table that should be compared to determine which records will be selected. These fields normally represent the primary key in one table and the corresponding foreign key in the other table.
- If an SQL statement uses fields with the same name from multiple tables, the field name must be fully qualified with its table name and a dot operator (**.**).
- An **INSERT** statement inserts a new record in a table. The simplest form of this statement is

```
INSERT INTO tableName ( fieldName1, fieldName2, …, fieldNameN )
   VALUES ( value1, value2, …, valueN )
```

  where *tableName* is the table in which to insert the record. The *tableName* is followed by a comma-separated list of field names in parentheses. The list of field names is followed by the SQL keyword **VALUES** and a comma-separated list of values in parentheses.
- SQL statements use a single quote (**'**) as a delimiter for strings. To specify a string containing a single quote in an SQL statement, the single quote must be escaped with another single quote.
- An **UPDATE** statement modifies data in a table. The simplest form for an **UPDATE** statement is

```
UPDATE tableName
   SET fieldName1 = value1, fieldName2 = value2, …, fieldNameN = valueN
   WHERE criteria
```

  where *tableName* is the table in which to update a record (or records). The *tableName* is followed by keyword **SET** and a comma-separated list of field name/value pairs in the format *fieldName* **=** *value*. The **WHERE** clause *criteria* determine the record(s) to update.
- A **DELETE** statement removes data from a table. The simplest form for a **DELETE** statement is

```
DELETE FROM tableName WHERE criteria
```

  where *tableName* is the table from which to delete a record (or records). The **WHERE** *criteria* determine which record(s) to delete.
- MySQL is an open source DBMS written in C/C++ and provides an extremely fast low-tier User Interface to the database.

- SQLServer 2000 is a Microsoft product designed for easy integration with Web applications. Of particular interest to C# programmers is the library of specially optimized code Microsoft has provided for interfacing with SQLServer.
- Oracle9i is a commercial database system in which all types of content are supported, users can make changes to databases through an online interface, and strong protocols are used to ensure security.
- Microsoft Access 2000™ is an easy-to-use Office 2000™ database program.
- **`System.Data`**, **`System.Data.OleDb`** and **`System.Data.SqlClient`** are the three main namespaces in ADO.NET.
- The first approach to ADO.NET programming has class **`DataSet`** of the **`System.Data`** namespace at its core. Instances of this class are in-memory caches of data.
- The advantage of using class **`DataSet`** is that it is a disconnected way to modify the contents of a datasource.
- The second approach to ADO.NET programming uses **`OleDbCommand`** of the **`System.Data.OleDb`** namespace. SQL statements are executed directly on the datasource.
- Fewer connections and more operations make the first approach the better choice. More connections and fewer operations make the second approach the better choice.
- The **`System.Data.SqlClient`** namespace is specially designed optimized code to interact with an SQLServer. Both interfacing levels and security checks are eliminated with **`System.Data.SqlClient`** to enhance performance.
- **`System.Data.OleDb`** is safer, general interfacing to any database.
- It is safe to assume that something written using classes in namespace **`OleDb`** can be directly converted to use classes in namespace **`SqlClient`**.
- Use the **<Add Connection>** option to create a database connection in the "**Data Link Properties**" window.
- Use the **Data Adapter Configuration Wizard** to set up an **`OleDbDataAdapter`** and generate queries.
- If a **`DataSet`** needs to be named, use the instance property **`DataSetName`**.
- **`OleDbCommand`**s commands are what the **`OleDbDataAdapter`** executes on the database in the form of SQL queries.
- Instance property **`TableMappings`** of class **`OleDbDataAdapter`** is a **`DataTableCollection`** and is used to create **`DataTableMapping`**s.
- **`DataColumnMapping`**s are used to convert data from a database to a **`DataSet`** and vice versa.
- Instance property **`Parameters`** of class **`OleDbCommand`** is a collection of **`OleDbParameter`** objects. Adding them to an **`OleDbCommand`** is an optional way to have parameters to a command, instead of creating a lengthy, complex command string.
- **`OleDbCommand`** instance property **`Connection`** is set to the **`OleDbConnection`** that the command will be executed on, and the instance property **`CommandText`** is set to the SQL query that will be executed on the database.
- **`OleDbDataAdapter`** method **`Fill`** retrieves information from the database, and the **`OleDbConnection`** is associated with and places it in the **`DataSet`** provided as an argument.
- **`DataGrid`** method **`SetDataBinding`** binds a **`DataGrid`** to a data source.
- Method **`Clear`** of class **`DataSet`** is called to empty the **`DataSet`** of any prior data.

- The instance properties **InsertCommand** and **UpdateCommand** of class **OleDbDataAdapter** are instances of class **OleDbCommand**.
- Property **CommandText** of class **OleDbCommand** is the **string** that represents the SQL statement to be executed.
- Method **ExecuteNonQuery** of class **OleDbCommand** is called to perform the action specified by **CommandText** on the database.
- C# has the ability to readily convert data in a datasource to XML and vice versa.
- Method **WriteXml** of class **DataSet** writes the XML representation of the **DataSet** instance to the first argument passed to it. This method had several overloads that allow an output source and a character encoding for the data to be specified.
- Method **ReadXml** of class **DataSet** reads the XML representation of the first argument passed to it into ots own **DataSet**. This method has several overloads that allow an input source and a character encoding for the data to be specified.

TERMINOLOGY

% SQL wildcard character
_ SQL wildcard character
AcceptChanges method of **DataRow**
AcceptChanges method of **DataSet**
AcceptChanges method of **DataTable**
ADO.NET
AND
Application Programming Interface
ASC
ASC (ascending order)
ascending order (ASC)
asterisk (*)
atomic operation
attribute
cache
Crystal Reports
Clear method of **DataSet**
column
column number
column number in a result set
CommandText method of **OleDbCommand**
CommandText property of **OleDbCommand**
commit a transaction
connect to a database
Connection**Connection** property of **OleDbCommand**
CrystalDecisions.Windows.Forms.CrystalReportViewer class
data attribute
database
database management system (DBMS)
database table
DataColumn class
DataColumnMapping class
DataGrid class
DataRow class

DataRowCollection class
DataSet class
DataSetName property of **DataSet**
DataTable class
DataTableCollection class
DataTableMapping class
DB2
default sorting order is ascending
DELETE
DELETE FROM
DeleteCommand property of **OleDbAdapter**
DESC
disconnected
distributed computing system
escape character
ExecuteNonQuery method of **OleDbCommand**
ExecuteNonQuery property of **OleDbCommand**
ExecuteReader method of **OleDbCommand**
ExecuteScalar method of **OleDbCommand**
field
Fill method of **OleDbAdapter**
FROM
fully qualified name
GetXml method of **DataSet**
GROUP BY
Informix
in-memory cache
INSERT INTO
INSERT INTO operation
InsertCommand property of **OleDbAdapter**
interface
ItemArray property of **DataRow**
joining tables
LIKE
locate records in a database
match the selection criteria
Merge records from Tables
Microsoft SQL Server
MySQL
OleDbCommand class
OleDbConnection class
OleDbDataAdapter class
OleDbDataReader class
OleDbParameter class
Oracle
ORDER BY
ordered
ordering of records
Parameters property of **OleDbParameter**
pattern matching

percent (**%**) SQL wildcard character
primary key
query
query a database
ReadXml method of **DataSet**
record
record set
Refresh method of **DataGrid**
RejectChanges method of **DataRow**
RejectChanges method of **DataTable**
relational database
relational database model
relational database table
result set
result sets
roll back a transaction
row
Rows property of **DataTable**
rows to be retrieved
SELECT
select
select all fields from a table
SelectCommand property of **OleDbAdapter**
selecting data from a table
selection criteria
SET
SET keyword
SetDataBinding method of **DataGrid**
single quote character
SQL (Structured Query Language)
SQL keywords
SQL statement
SqlConnection class
square brackets in a query
Structured Query Language (SQL)
Sybase
System.Data namespace
System.Data.OleDb namespace
System.Data.Sqlclient namespace
table
table column
table in which record will be updated
table row
TableMappings property of **OleDbAdapter**
tableName.*fieldName*
Tables property of **DataSet**
tree structure
underscore (_) SQL wildcard character
UPDATE
Update method of **OleDbDataAdapter**

UpdateCommand property of **OleDbAdapter**
VALUES
WHERE
WriteXml method of **DataSet**
XML document

SELF-REVIEW EXERCISES

19.1 Fill in the blanks in each of the following statements:

a) The most popular database query language is __________.
b) A table in a database consists of __________ and __________.
c) Databases can be manipulated in C# as __________ objects.
d) Use class __________ to display data graphically in C#.
e) SQL keyword __________ is followed by the selection criteria that specify the records to select in a query.
f) SQL keyword __________ specifies the order in which records are sorted in a query.
g) Selecting data from multiple database tables is called __________ the data.
h) A __________ is an integrated collection of data that is centrally controlled.
i) A __________ is a field in a table for which every entry has a unique value in another table and where the field in the other table is the primary key for that table.
j) Namespace __________ contains special classes and interfaces for manipulating SQLServer databases in C#.
k) C# uses __________ to transmit data between datasources.
l) Namespace __________ is C#'s general interfacing to a database.

19.2 State which of the following are *true* or *false*. If *false*, explain why.

a) In general, ADO.NET is a disconnected model.
b) SQL can implicitly convert fields with the same name from two or mores tables to the appropriate field.
c) Only the **UPDATE** SQL statement can commit changes to a database.
d) Executing **OleDbCommand**s is not a transaction process.
e) **DataSet**s can implicitly convert XML data read with method **ReadXml** into its tables.
f) **SELECT** statements can merge data from multiple tables.
g) Crystal Reports is an example of a DBMS.
h) An **OleDbDataAdapter** can **Fill** a **DataSet**.
i) All of a **DataRow**'s values can be implicitly assigned with the instance property **ItemArray**.
j) SQLServer is an example of a managed provider.
k) Because C# uses a disconnected model, **OleDbConnection**s are optional.
l) It is always faster to assign a value to a variable than to instantiate a new **object**.

ANSWERS TO SELF-REVIEW EXERCISES

19.1 a) SQL. b) rows, columns. c) **DataSet**. d) **DataGrid**. e) **WHERE**. f) **ORDER BY**. g) joining. h) database. i) foreign key. j) **System.Data.Sql**. k) XML. l) **System.Data.OleDb**.

19.2 a) True. b) False. In a query, not providing fully-qualified names for fields with the same name from two or more tables is an error. c) False. **INSERT** and **DELETE** change the database too. Do not confuse the SQL **Update** statement with method **OleDbDataAdapter.Update**. d)

True. e) False. The **DataSet** must be **Clear**ed first or the **DataRows** must be explicitly updated. f) True. g) False. Crystal Reports creates graphical/Web representations of data. h) True. i) True. j) True. k) False. This class is required to connect to a database. l) True.

EXERCISES

19.3 Using the techniques shown in this chapter, define a complete query application for the Authors.mdb database. Provide a series of predefined queries with an appropriate name for each query-displayed in a **System.Windows.Forms.ComboBox**. Also allow the user to supply their own queries and add them to the **ComboBox**. Provide any queries you feel are appropriate.

19.4 Using the techniques shown in this chapter, define a complete query application for the **Books.mdb** database. Provide a series of predefined queries with an appropriate name for each query displayed in a **System.Windows.Forms.ComboBox**. Also, allow users to supply their own queries and add them to the **ComboBox**. Provide the following predefined queries:

a) Select all authors from the **Authors** table.
b) Select all publishers from the **Publishers** table.
c) Select a specific author and list all books for that author. Include the title, year and ISBN number. Order the information alphabetically by title.
d) Select a specific publisher and list all books published by that publisher. Include the title, year and ISBN number. Order the information alphabetically by title.
e) Provide any other queries you feel are appropriate.

19.5 Modify Exercise 19.4 to define a complete database manipulation application for the **Books.mdb** database. In addition to the querying capabilities, the user should be able to edit existing data and add new data to the database. Allow the user to edit the database in the following ways:

a) Add a new author.
b) Edit the existing information for an author.
c) Add a new title for an author (remember that the book must have an entry in the **AuthorISBN** table). Be sure to specify the publisher of the title.
d) Add a new publisher.
e) Edit the existing information for a publisher.

For each of the preceding database manipulations, design an appropriate GUI to allow the user to perform the data manipulation.

19.6 Modify the address book example of Fig. 19.20 to enable each address book entry to contain multiple addresses, phone numbers and e-mail addresses. The user of the program should be able to view multiple addresses, phone numbers and e-mail addresses. [*Note:* This is a large exercise that requires substantial modifications to the original classes in the address book example.]

19.7 Create an application that allows the user to modify all fields of a database using a transaction process model. The user should be able to find, modify and create entries. The GUI should include buttons **Accept Changes** and **Reject Changes**. Modifications to the datasource should be made when the user clicks **Accept Changes** by invoking method **Update** of the **OleDbDataAdapter** object. The **DataSet**'s **AcceptChanges** method should be invoked *after* changes are made to the datasource.

19.8 Write a program that allows the user to graphically modify a database through an XML text editor. The GUI should be able to display the contents of the database and commit any changes to the XML text to the database.

BIBLIOGRAPHY

Archer, Tom, *Inside C#*. Redmond, Washington: Microsoft Press, 2001.

Blaha, M. R.; W. J. Premerlani; and J. E. Rumbaugh, "Relational Database Design Using an Object-Oriented Methodology," *Communications of the ACM*, Vol. 31, No. 4, April 1988, pp. 414–427.

Codd, E. F., "A Relational Model of Data for Large Shared Data Banks," *Communications of the ACM*, June 1970.

Codd, E. F., "Further Normalization of the Data Base Relational Model," in *Courant Computer Science Symposia*, Vol. 6, *Data Base Systems*. Upper Saddle River, N.J.: Prentice Hall, 1972.

Codd, E. F., "Fatal Flaws in SQL," *Datamation*, Vol. 34, No. 16, August 15, 1988, pp. 45–48.

Conrad, James et al., *Introducing .NET*. Birmingham, UK: Wrox Press, 2000.

Deitel, H. M., *Operating Systems, Second Edition*. Reading, MA: Addison Wesley Pubishing, 1990.

Date, C. J., *An Introduction to Database Systems*. Reading, MA: Addison Wesley Pubishing, 1981.

Date, C. J., *An Introduction to Database Systems, Seventh Edition*. Reading, MA: Addison Wesley Pubishing, 2000.

Harvey, Burton et al., *C# Programming With the Public Beta*. Birmingham, UK: Wrox Press, 2000.

Microsoft Developer Network Library > .NET Framework SDK. <**msdn.microsoft.com/library/default.asp**>

MySQL Manual. <**www.mysql.com/doc/**>

Oracle Technology Network > Documentation. <**otn.oracle.com/docs/content.html**>

Relational Technology, *INGRES Overview*. Alameda, CA: Relational Technology, 1988.

Stonebraker, M., "Operating System Support for Database Management," *Communications of the ACM*, Vol. 24, No. 7, July 1981, pp. 412–418.

Visual Stufio .NET > ADO.NET Overview. <**msdn.microsoft.com/vstudio/nextgen/technology/adoplusdefault.asp**>

Winston, A., "A Distributed Database Primer," *UNIX World*, April 1988, pp. 54–63.