21.4 SQL

We now overview SQL in the context of the `Books` database. Though LINQ to SQL and the Visual C# IDE hide the SQL used to manipulate databases, it is nevertheless important to understand SQL basics. Knowing the types of operations you can perform will help you develop more advanced database-intensive applications.

Figure 21.10 lists some common **SQL keywords** used to form complete **SQL statements**—we discuss these keywords in the next several subsections. Other SQL keywords exist, but they are beyond the scope of this text.

SQL keyword	Description
`SELECT`	Retrieves data from one or more tables.
`FROM`	Specifies the tables involved in a query. Required in every query.
`WHERE`	Specifies optional criteria for selection that determine the rows to be retrieved, deleted or updated.
`ORDER BY`	Specifies optional criteria for ordering rows (e.g., ascending, descending).
`INNER JOIN`	Specifies optional operator for merging rows from multiple tables.
`INSERT`	Inserts rows in a specified table.
`UPDATE`	Updates rows in a specified table.
`DELETE`	Deletes rows from a specified table.

Fig. 21.10 | Common SQL keywords.

21.4.1 Basic SELECT Query

Let us consider several SQL queries that retrieve information from database `Books`. A SQL **query** "selects" rows and columns from one or more tables in a database. Such selections are performed by queries with the **SELECT** keyword. The basic form of a **SELECT query** is

```
SELECT * FROM tableName
```

in which the asterisk (*) indicates that all the columns from the *tableName* table should be retrieved. For example, to retrieve all the data in the `Authors` table, use

```
SELECT * FROM Authors
```

Note that the rows of the `Authors` table are not guaranteed to be returned in any particular order. You will learn how to specify criteria for sorting rows in Section 21.4.3.

Most programs do not require all the data in a table—in fact, selecting all the data from a large table is discouraged, as it can cause performance problems. To retrieve only specific columns from a table, replace the asterisk (*) with a comma-separated list of the column names. For example, to retrieve only the columns `AuthorID` and `LastName` for all the rows in the `Authors` table, use the query

```
SELECT AuthorID, LastName FROM Authors
```

This query returns only the data listed in Fig. 21.11.

AuthorID	LastName
1	Deitel
2	Deitel
3	Ayer
4	Quirk

Fig. 21.11 | `AuthorID` and `LastName` data from the `Authors` table.

21.4.2 WHERE Clause

When users search a database for rows that satisfy certain **selection criteria** (formally called **predicates**), only rows that satisfy the selection criteria are selected. SQL uses the optional **`WHERE` clause** in a query to specify the selection criteria for the query. The basic form of a query with selection criteria is

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

For example, to select the `BookTitle`, `EditionNumber` and `Copyright` columns from table `Titles` for which the `Copyright` date is more recent than `2007`, use the query

```
SELECT BookTitle, EditionNumber, Copyright
FROM Titles
WHERE Copyright > '2007'
```

Note that string literals in SQL are delimited by single quotes instead of double quotes as in C#. In SQL, double quotes are used around table and column names that would otherwise be invalid—names containing SQL keywords, spaces, or other punctuation characters. Figure 21.12 shows the result of the preceding query.

BookTitle	EditionNumber	Copyright
Internet & World Wide Web How to Program	4	2008
Simply Visual Basic 2008	3	2009
Visual Basic 2008 How to Program	4	2009
Visual C# 2008 How to Program	3	2009
Visual C++ 2008 How to Program	2	2008
C++ How to Program	6	2008

Fig. 21.12 | Books with copyright dates after 2007 from table `Titles`.

The `WHERE`-clause criteria can contain the comparison operators `<`, `>`, `<=`, `>=`, `=` (equality), `<>` (inequality) and `LIKE`, as well as the logical operators `AND`, `OR` and `NOT` (discussed in Section 21.4.6). Operator **`LIKE`** is used for **pattern matching** with wildcard characters **percent** (`%`) and **underscore** (`_`). Pattern matching allows SQL to search for strings that match a given pattern.

A pattern that contains a percent character (`%`) searches for strings that have zero or more characters at the percent character's position in the pattern. For example, the following query locates the rows 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 rows shown in Fig. 21.13, because two of the four authors in our database have a last name 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` column. Note that the pattern string is surrounded by single-quote characters.

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

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

The preceding query produces the row shown in Fig. 21.14, because only one author in our database has a last name that contains the letter `y` as its second letter.

AuthorID	FirstName	LastName
1	Harvey	Deitel
2	Paul	Deitel

Fig. 21.13 | Authors from the `Authors` table whose last names start with `D`.

AuthorID	FirstName	LastName
3	Greg	Ayer

Fig. 21.14 | The only author from the `Authors` table whose last name contains `y` as the second letter.

21.4.3 ORDER BY Clause

The rows in the result of a query can be sorted into ascending or descending order by using the optional **`ORDER BY` clause**. The basic form of a query with an `ORDER BY` clause is

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

where **`ASC`** specifies ascending order (lowest to highest), **`DESC`** specifies descending order (highest to lowest) and *column* specifies the column on which the sort is based. For example, to obtain the list of authors in ascending order by last name (Fig. 21.15), use the query

```
SELECT AuthorID, FirstName, LastName
FROM Authors
ORDER BY LastName ASC
```

The default sorting order is ascending, so `ASC` is optional in the preceding query. To obtain the same list of authors in descending order by last name (Fig. 21.16), use

```
SELECT AuthorID, FirstName, LastName
FROM Authors
ORDER BY LastName DESC
```

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

Fig. 21.15 | Authors from table `Authors` in ascending order by `LastName`.

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

Fig. 21.16 | Authors from table `Authors` in descending order by `LastName`.

Multiple columns can be used for sorting with an `ORDER BY` clause of the form

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

where *sortingOrder* is either `ASC` or `DESC`. Note that the *sortingOrder* does not have to be identical for each column. For example, the query

```
SELECT BookTitle, EditionNumber, Copyright
FROM Titles
ORDER BY Copyright DESC, BookTitle ASC
```

returns the rows of the `Titles` table sorted first in descending order by copyright date, then in ascending order by title (Fig. 21.17). This means that rows with higher `Copyright` values are returned before rows with lower `Copyright` values, and any rows that have the same `Copyright` values are sorted in ascending order by title.

The `WHERE` and `ORDER BY` clauses can be combined. If used, `ORDER BY` must be the last clause in the query. For example, the query

```
SELECT ISBN, BookTitle, EditionNumber, Copyright
FROM Titles
WHERE BookTitle LIKE '%How to Program'
ORDER BY BookTitle ASC
```

returns the `ISBN`, `BookTitle`, `EditionNumber` and `Copyright` of each book in the `Titles` table that has a `BookTitle` ending with "`How to Program`" and sorts them in ascending order by `BookTitle`. The query results are shown in Fig. 21.18.

BookTitle	EditionNumber	Copyright
Simply Visual Basic 2008	3	2009
Visual Basic 2008 How to Program	4	2009
Visual C# 2008 How to Program	3	2009
C++ How to Program	6	2008
Internet & World Wide Web How to Program	4	2008
Visual C++ 2008 How to Program	2	2008
C How to Program	5	2007
Java How to Program	7	2007

Fig. 21.17 | Data from `Titles` in descending order by `Copyright` and ascending order by `BookTitle`.

ISBN	BookTitle	EditionNumber	Copyright
0132404168	C How to Program	5	2007
0136152503	C++ How to Program	6	2008
0131752421	Internet & World Wide Web How to Program	4	2008

Fig. 21.18 | Books from table `Titles` whose `BookTitles` end with `How to Program` in ascending order by `BookTitle`. (Part 1 of 2.)

ISBN	BookTitle	EditionNumber	Copyright
0132222205	Java How to Program	7	2007
013605305X	Visual Basic 2008 How to Program	4	2009
013605322X	Visual C# 2008 How to Program	3	2009
0136151574	Visual C++ 2008 How to Program	2	2008

Fig. 21.18 | Books from table `Titles` whose `BookTitles` end with `How to Program` in ascending order by `BookTitle`. (Part 2 of 2.)

21.4.4 Retrieving Data from Multiple Tables: INNER JOIN

Database designers typically **normalize** databases—i.e., split related data into separate tables to ensure that a database does not store redundant data. For example, the `Books` database has tables `Authors` and `Titles`. We use an `AuthorISBN` table to store "links" between authors and titles. If we did not separate this information into individual tables, we would need to include author information with each entry in the `Titles` table. This would result in the database storing duplicate author information for authors who have written more than one book.

Redundant data in a database increases the likelihood of errors when manpulating the data. Figure 21.1 contains redundant information between the `Department` and `Location` columns—for each department number, there is a single location and vice versa. This relationship is not enforced by the table's structure. Normalization eliminates redundant data and allows the DBMS to prevent problems that could arise if queries depend on the one-to-one mapping between `Department` and `Location`.

Often, it is desirable to merge data from multiple tables into a single result—this is referred to as joining the tables. There are several kinds of joins, but the most common one is specified by an **`INNER JOIN` operator** in the query. An `INNER JOIN` merges rows from two tables by testing for matching values in a column that is common to the tables (though the column names can differ among the tables). The basic form of an `INNER JOIN` is:

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

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

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

The query combines the `FirstName` and `LastName` columns from table `Authors` and the `ISBN` column from table `AuthorISBN`, sorting the results in ascending order by `LastName` and `FirstName`. Note the use of the syntax *tableName*`.`*columnName* in the `ON` clause. This syntax (called a **qualified name**) specifies the columns from each table that should be compared to join the tables. The "*tableName*`.`" syntax is required if the columns have the same

name in both tables. The same syntax can be used in any query to distinguish columns that have the same name in different tables.

Common Programming Error 21.4

In a SQL query, failure to qualify names for columns that have the same name in two or more tables is an error.

As always, the query can contain an `ORDER BY` clause. Figure 21.19 depicts the results of the preceding query, ordered by `LastName` and `FirstName`.

FirstName	LastName	ISBN
Greg	Ayer	0136053033
Harvey	Deitel	0131752421
Harvey	Deitel	0132222205
Harvey	Deitel	0132404168
Harvey	Deitel	0136053033
Harvey	Deitel	013605305X
Harvey	Deitel	013605322X
Harvey	Deitel	0136151574
Harvey	Deitel	0136152503
Paul	Deitel	0131752421
Paul	Deitel	0132222205
Paul	Deitel	0132404168
Paul	Deitel	0136053033
Paul	Deitel	013605305X
Paul	Deitel	013605322X
Paul	Deitel	0136151574
Paul	Deitel	0136152503
Dan	Quirk	0136151574

Fig. 21.19 | Authors and ISBNs for their books in ascending order by `LastName` and `FirstName`.

21.4.5 INSERT Statement

The **`INSERT` statement** inserts a row into a table. The basic form of this statement is

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

where *tableName* is the table in which to insert the row. The *tableName* is followed by a comma-separated list of column names in parentheses. The list of column names is followed by the SQL keyword **`VALUES`** and a comma-separated list of values in parentheses. The values specified here must match up with the columns specified after the table name in both order and type (e.g., if *columnName1* is supposed to be the `FirstName` column, then *value1* should be a string in single quotes representing the first name). Although the

list of column names is not required if the `INSERT` operation specifies a value for every table column in the correct order, you should always explicitly list the columns when inserting rows—if the order of the columns in the table changes, using only `VALUES` may cause an error. The `INSERT` statement

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

inserts a row into the `Authors` table. The statement indicates that the values `'Sue'` and `'Smith'` are provided for the `FirstName` and `LastName` columns, respectively.

Some database tables allow `NULL` columns—that is, columns without values. Though the capitalization is different, `NULL` in SQL is similar to the idea of `null` in C#. All of the columns in the `Books` database are required, so they must be given values in an `INSERT` statement.

We do not specify an `AuthorID` in this example, because `AuthorID` is an identity column in the `Authors` table (see Fig. 21.3). For every row added to this table, SQL Server assigns a unique `AuthorID` value that is the next value in an autoincremented sequence (i.e., 1, 2, 3 and so on). In this case, Sue Smith would be assigned `AuthorID` number `5`. Figure 21.20 shows the `Authors` table after the `INSERT` operation.

Common Programming Error 21.5

It is an error to specify a value for an identity column in an `INSERT` statement.

Common Programming Error 21.6

SQL uses the single-quote (') character to delimit strings. To specify a string containing a single quote (e.g., O'Malley) in a SQL statement, there must be 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 a SQL statement is a syntax error.

AuthorID	FirstName	LastName
1	Harvey	Deitel
2	Paul	Deitel
3	Greg	Ayer
4	Dan	Quirk
5	Sue	Smith

Fig. 21.20 | Table `Authors` after an `INSERT` operation.

21.4.6 UPDATE Statement

An **`UPDATE` statement** modifies data in a table. The basic form of the `UPDATE` statement is

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

where *tableName* is the table to update. The *tableName* is followed by keyword **SET** and a comma-separated list of column name/value pairs in the format *columnName* = *value*. The optional `WHERE` clause provides criteria that determine which rows to update. While it is not required, the `WHERE` clause is almost always used, in an `UPDATE` statement because omitting it updates all rows in the table—an uncommon operation. The `UPDATE` statement

```
UPDATE Authors
SET LastName = 'Jones'
WHERE LastName = 'Smith' AND FirstName = 'Sue'
```

updates a row in the `Authors` table. Keyword **AND** is a logical operator that, like the C# `&&` operator, returns true *if and only if* both of its operands are true. Thus, the preceding statement assigns to `LastName` the value `Jones` for the row in which `LastName` is equal to `Smith` *and* `FirstName` is equal to `Sue`. [*Note:* If there are multiple rows with the first name "Sue" and the last name "Smith," this statement modifies all such rows to have the last name "Jones."] Figure 21.21 shows the `Authors` table after the `UPDATE` operation has taken place. SQL also provides other logical operators, such as **OR** and **NOT**, which behave like their C# counterparts `||` and `!`.

AuthorID	FirstName	LastName
1	Harvey	Deitel
2	Paul	Deitel
3	Greg	Ayer
4	Dan	Quirk
5	Sue	Jones

Fig. 21.21 | Table `Authors` after an `UPDATE` operation.

21.4.7 DELETE Statement

A **DELETE statement** removes rows from a table. Its basic form is

```
DELETE FROM tableName WHERE criteria
```

where *tableName* is the table from which to delete. The optional `WHERE` clause specifies the criteria used to determine which rows to delete. As with the `UPDATE` statement, the `DELETE` applies to all rows of the table if the `WHERE` clause is omitted. The `DELETE` statement

```
DELETE FROM Authors
WHERE LastName = 'Jones' AND FirstName = 'Sue'
```

deletes the row for Sue Jones in the `Authors` table. `DELETE` statements can delete multiple rows if the rows all meet the criteria in the `WHERE` clause. Figure 21.22 shows the `Authors` table after the `DELETE` operation has taken place.

SQL Wrap-Up

This concludes our SQL introduction. We demonstrated several commonly used SQL keywords, formed SQL queries that retrieved data from databases and formed other SQL statements that manipulated data in a database. Next, we introduce LINQ to SQL, which allows

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

Fig. 21.22 | Table `Authors` after a `DELETE` operation.

C# applications to interact with databases. As you will see, LINQ to SQL translates LINQ queries like the ones you wrote in Chapter 9 into SQL statements like those presented here.