24.4 SQL

We now discuss SQL in the context of our `books` database. You'll be able to use the SQL discussed here in the examples later in the chapter. The next several subsections demonstrate SQL queries and statements using the SQL keywords in Fig. 24.10. Other SQL keywords are beyond this text's scope.

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

Fig. 24.10

SQL query keywords.

24.4.1 Basic SELECT Query

Let’s consider several SQL queries that extract 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 (`*`)** *wildcard character* indicates that all columns from the *tableName* table should be retrieved. For example, to retrieve all the data in the `Authors` table, use

```
SELECT * FROM Authors
```

Most programs do not require all the data in a table. To retrieve only specific columns, replace the `*` with a comma-separated list of column names. For example, to retrieve only the columns `AuthorID` and `LastName` for all rows in the `Authors` table, use the query

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

This query returns the data listed in Fig. 24.11.

AuthorID	LastName	AuthorID	LastName
1	Deitel	4	Quirk
2	Deitel	5	Morgano
3	Deitel		

Fig. 24.11

Sample `AuthorID` and `LastName` data from the `Authors` table.

 # Software Engineering Observation 24.2

In general, you process results by knowing in advance the column order—for example, selecting `AuthorID` *and* `LastName` *from* `Authors` *ensures that the columns will appear in the result in that exact order. Selecting columns by name avoids returning unneeded columns and protects against changes in the actual database column order. Programs can then process result columns by specifying the column number*

in the result (starting from number 1 for the first column).

 Common Programming Error 24.1

If you assume that the columns are always returned in the same order from a query that uses the asterisk (*), the program may process the results incorrectly. If the column order in the table(s) changes or if additional columns are added at a later time, the order of the columns in the result will change accordingly.

24.4.2 `WHERE` Clause

In most cases, it's necessary to locate rows in a database that satisfy certain **selection criteria**. Only rows that satisfy the selection criteria (formally called **predicates**) 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 WHE
```

For example, to select the `Title`, `EditionNumber` and `Copyright` columns from table `Titles` for which the

`Copyright` date is greater than `2013`, use the query

```
SELECT Title, EditionNumber, Copyright
   FROM Titles
   WHERE Copyright > '2013'
```

Strings in SQL are delimited by single (') rather than double (") quotes. Figure 24.12 shows the result of the preceding query.

Title	EditionNumber	Copyright
Java How to Program	10	2015
Java How to Program, Late Objects Version	10	2015
Visual Basic 2012 How to Program	6	2014
Visual C# 2012 How to Program	5	2014
C++ How to Program	9	2014
Android How to Program	2	2015
Android for Programmers: An App-Driven Approach, Volume 1	2	2014

Fig. 24.12

Sampling of titles with copyrights after 2013 from table

`Titles`.

Pattern Matching: Zero or More Characters

The `WHERE` clause criteria can contain the operators `<`, `>`, `<=`, `>=`, `=`, `<>` (not equal) and `LIKE`. 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 next query locates the rows of all the authors whose last name starts with the letter `D`:

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

This query selects the two rows shown in Fig. 24.13—three of the five authors have a last name starting with the letter `D` (followed by zero or more characters). The `%` symbol in the `WHERE` clause's `LIKE` pattern indicates that any number of characters can appear after the letter `D` in the `LastName`. The pattern string is surrounded by single-quote characters.

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

Fig. 24.13

Authors whose last name starts with `D` from the `Authors` table.

Portability Tip 24.1

See the documentation for your database system to determine whether SQL is case sensitive on your system and to determine the syntax for SQL keywords, such as the `LIKE` *operator.*

Pattern Matching: Any Character

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 `o`, followed by any number of additional

characters (specified by %):

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

The preceding query produces the row shown in Fig. 24.14, because only one author in our database has a last name that contains the letter o as its second letter.

AuthorID	FirstName	LastName
5	Michael	Morgano

Fig. 24.14

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

24.4.3 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 ORD
SELECT columnName1, columnName2, … FROM tableName ORD
```

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. 24.15), use the query

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

AuthorID	FirstName	LastName
1	Paul	Deitel
2	Harvey	Deitel
3	Abbey	Deitel
5	Michael	Morgano
4	Dan	Quirk

Fig. 24.15

Sample data from table `Authors` in ascending order by `LastName`.

Sorting in Descending Order

The default sorting order is ascending, so ASC is optional. To obtain the same list of authors in descending order by last name (Fig. 24.16), use the query

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

AuthorID	FirstName	LastName
4	Dan	Quirk
5	Michael	Morgano
1	Paul	Deitel
2	Harvey	Deitel
3	Abbey	Deitel

Fig. 24.16

Sample data from table `Authors` in descending order by `LastName`.

Sorting By Multiple Columns

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. The *sortingOrder* does not have to be identical for each column. The query

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

sorts all the rows in ascending order by last name, then by first name. If any rows have the same last-name value, they're returned sorted by first name (Fig. 24.17).

AuthorID	FirstName	LastName
3	Abbey	Deitel
2	Harvey	Deitel
1	Paul	Deitel
5	Michael	Morgano
4	Dan	Quirk

Fig. 24.17

Sample data from `Authors` in ascending order by `LastName` and `FirstName`.

Combining the `WHERE` and `ORDER BY` Clauses

The `WHERE` and `ORDER BY` clauses can be combined in one query, as in

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

which returns the `ISBN`, `Title`, `EditionNumber` and `Copyright` of each book in the `Titles` table that has a `Title` ending with `"How to Program"` and sorts them in ascending order by `Title`. The query results are shown in Fig. 24.18.

ISBN	Title	EditionNumber	Copyright
0133764036	Android How to Program	2	2015
013299044X	C How to Program	7	2013
0133378713	C++ How to Program	9	2014
0132151006	Internet & World Wide Web How to Program	5	2012
0133807800	Java How to Program	10	2015

0133406954	Visual Basic 2012 How to Program	6	2014
0133379337	Visual C# 2012 How to Program	5	2014
0136151574	Visual C++ 2008 How to Program	2	2008

Fig. 24.18

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

24.4.4 Merging Data from Multiple Tables: `INNER JOIN`

Database designers often split related data into separate tables to ensure that a database does not store data redundantly. For example, in the `books` database, the `AuthorISB` table stores the relationship data between authors and their corresponding titles. If we did not separate this information into individual tables, we'd need to include author information with each entry in the `Titles` table. This would result in the database's storing *duplicate* author information for authors who wrote multiple books. Often, it's necessary to merge data from multiple tables into a single result. Referred to as joining the

tables, this is specified by an `INNER JOIN` operator, which merges rows from two tables by matching values in columns that are common to 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—one is a primary key and the other is a foreign key in the tables being joined. 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 merges the `FirstName` and `LastName` columns from table `Authors` with 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 SQL statement to distinguish columns in different tables that have the same name. In some systems, table names qualified with the database name can be used to perform cross-database queries. As always, the query can contain an `ORDER BY` clause. Figure 24.19 shows the results of the preceding query, ordered by `LastName` and `FirstName`.

Common Programming Error 24.2

Failure to qualify names for columns that have the same name in two or more tables is an error. In such cases, the statement must precede those column names with their table names and a dot (e.g., `Authors.AuthorID`*).*

FirstName	LastName	ISBN	FirstName	LastName	ISBN
Abbey	Deitel	0132121360	Harvey	Deitel	013299044X
Abbey	Deitel	0133570924	Harvey	Deitel	0132575655
Abbey	Deitel	0133764036	Paul	Deitel	0133406954
Abbey	Deitel	0133406954	Paul	Deitel	0132990601
Abbey	Deitel	0132990601	Paul	Deitel	0132121360
Abbey	Deitel	0132151006	Paul	Deitel	0133570924
Harvey	Deitel	0132121360	Paul	Deitel	0133764036

Harvey	Deitel	0133570924	Paul	Deitel	0133378713
Harvey	Deitel	0133807800	Paul	Deitel	0136151574
Harvey	Deitel	0132151006	Paul	Deitel	0133379337
Harvey	Deitel	0133764036	Paul	Deitel	013299044X
Harvey	Deitel	0133378713	Paul	Deitel	0132575655
Harvey	Deitel	0136151574	Paul	Deitel	0133807800
Harvey	Deitel	0133379337	Paul	Deitel	0132151006
Harvey	Deitel	0133406954	Michael	Morgano	0132121360
Harvey	Deitel	0132990601	Dan	Quirk	0136151574

Fig. 24.19

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

24.4.5 `INSERT` Statement

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

```
INSERT INTO tableName (columnName1, columnName2, …, c
   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 (this list is not required if the `INSERT` operation specifies a value for every column of the table in the correct order). 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 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). Always explicitly list the columns when inserting rows. If the table's column order changes or a new column is added, using only `VALUES` may cause an error. The `INSERT` statement

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

inserts a row into the `Authors` table. The statement indicates that values are provided for the `FirstName` and `LastName` columns. The corresponding values are `'Sue'` and `'Smith'`. We do not specify an `AuthorID` in this example because `AuthorID` is an autoincremented column in the `Authors` table. For every row added to this table, the DBMS assigns a unique `AuthorID` value that is the next value in the autoincremented sequence (i.e., 1, 2, 3 and so on). In this case, Sue Red would be assigned `AuthorID` number 6. Figure 24.20 shows the `Authors` table after the `INSERT` operation. [*Note:* Not every database management system supports autoincremented columns. Check the documentation for your

DBMS for alternatives to autoincremented columns.]

 Common Programming Error 24.3

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

 Common Programming Error 24.4

It's normally an error to specify a value for an autoincrement column.

AuthorID	FirstName	LastName
1	Paul	Deitel
2	Harvey	Deitel
3	Abbey	Deitel
4	Dan	Quirk
5	Michael	Morgano

6	Sue	Red

Fig. 24.20

Sample data from table `Authors` after an `INSERT` operation.

24.4.6 UPDATE Statement

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

```
UPDATE tableName
   SET columnName1 = value1, columnName2 = value2, …,
   WHERE criteria
```

where *tableName* is the table to update. The *tableName* is followed by keyword `SET` and a comma-separated list of *columnName* = *value* pairs. The optional `WHERE` clause provides criteria that determine which rows to update. Though not required, the `WHERE` clause is typically used, unless a change is to be made to every row. The `UPDATE` statement

```
UPDATE Authors
   SET LastName = 'Black'
   WHERE LastName = 'Red' AND FirstName = 'Sue'
```

updates a row in the `Authors` table. The statement indicates that `LastName` will be assigned the value `Black` for the row where `LastName` is `Red` and `FirstName` is `Sue`. [*Note:* If there are multiple matching rows, this statement will modify *all* such rows to have the last name "Black."] If we know the `AuthorID` in advance of the `UPDATE` operation (possibly because we searched for it previously), the `WHERE` clause can be simplified as follows:

```
WHERE AuthorID = 6
```

Figure 24.21 shows the `Authors` table after the `UPDATE` operation has taken place.

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

Fig. 24.21

Sample data from table `Authors` after an `UPDATE` operation.

24.4.7 DELETE Statement

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

```
DELETE FROM tableName WHERE criteria
```

where *tableName* is the table from which to delete. The optional `WHERE` clause specifies the criteria used to determine which rows to delete. If this clause is omitted, all the table's rows are deleted. The `DELETE` statement

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

deletes the row for Sue Black in the `Authors` table. If we know the `AuthorID` in advance of the `DELETE` operation, the `WHERE` clause can be simplified as follows:

```
WHERE AuthorID = 6
```

Figure 24.22 shows the `Authors` table after the `DELETE` operation has taken place.

AuthorID	FirstName	LastName
1	Paul	Deitel

2	Harvey	Deitel
3	Abbey	Deitel
4	Dan	Quirk
5	Michael	Morgano

Fig. 24.22

Sample data from table `Authors` after a `DELETE` operation.