

CHAPTER 3 The WHERE and ORDER BY Clauses

Oracle SQL By Example, Fourth Edition by Alice Rischert. Published by Prentice Hall. Copyright © 2008 by Pearson Education, Inc.

CHAPTER OBJECTIVES

In this chapter, you will learn about:

- ▶ The WHERE Clause: Comparison and Logical Operators
- ▶ The ORDER BY Clause

Now that you are familiar with the basic functionality of a SELECT statement and know your way around the execution environments, you are ready to delve into the power of the SQL language. The WHERE and ORDER BY clauses are two very commonly used clauses. The WHERE clause limits a result set, and the ORDER BY clause sorts the output. Performing the exercises in this chapter will help you master these fundamental SQL language elements.

LAB 3.1 The WHERE Clause

LAB OBJECTIVES

After this lab, you will be able to:

- ▶ Use Comparison and Logic Operators in a WHERE Clause
- ▶ Use NULL in a WHERE Clause

The *WHERE clause*, also called the *predicate*, provides the power to narrow down the scope of data retrieved. In fact, most SQL statements you write will probably contain a WHERE clause.

Comparison Operators

To filter a data set, you need to specify a WHERE clause condition, which results in true, false, or unknown. The condition consists of an *expression* that can be a column of any data type, a *string* or *text literal* (sometimes referred to as a *text constant* or *character literal*), a number, a function, a mathematical computation, or any combination of these. The comparison operators evaluate the expressions for the selection of the appropriate data.

[Table 3.1](#) provides a list of the most common comparison operators. You will learn about additional operators, such as EXISTS, ANY, SOME, and ALL, in [Chapter 8](#), “Subqueries,” and the OVERLAPS operator in [Chapter 5](#), “Date and Conversion Functions.” All these comparison operators can be negated with the NOT logical operator.

TABLE 3.1 SQL Comparison Operators

COMPARISON OPERATOR	DEFINITION
=	Equal
!=, <>	Not equal
>, >=	Greater than, greater than or equal to
<, <=	Less than, less than or equal to
BETWEEN ... AND ...	Inclusive of two values
LIKE	Pattern matching with wildcard characters % and _
IN (...)	List of values
IS NULL	Test for null values

102

THE EQUALITY AND INEQUALITY OPERATORS

One of the most commonly used comparison operators is the *equality* operator, denoted by the = symbol. For example, if you are asked to provide the first name, last name, and phone number of a teacher with the last name Schorin, you write the following SQL statement.

```
SELECT first_name, last_name, phone
FROM instructor
WHERE last_name = 'Schorin'
FIRST_NAME LAST_NAME PHONE
-----
Nina Schorin 2125551212

1 row selected.
```

Here, the column LAST_NAME is the left side of the equation, and the text literal 'Schorin' is the right side. (Single quotation marks are used around the text literal 'Schorin'.) This statement retrieves only rows from the INSTRUCTOR table that satisfy this condition in the WHERE clause. In this case, only one row is retrieved.

When you review the data types of the INSTRUCTOR table, you see that the LAST_NAME column's data type is VARCHAR2. This means the data contained in this column is alphanumeric. When two values are

compared to each other, they must be of the same data type; otherwise, Oracle returns an error. You will learn more about converting from one data type to another in [Chapter 5](#).

SQL is case-insensitive when it comes to column names, table names, and keywords such as SELECT. (There are some exceptions with regard to column names and table names. For more information, see [Chapter 12](#), “Create, Alter, and Drop Tables.”) When you compare a text literal to a database column, the case of the data must match exactly. The syntax of the following statement is correct, but it does not yield any rows because the instructor’s last name is obviously not in the correct case.

```
SELECT first_name, last_name, phone
FROM instructor
WHERE last_name = 'schorin'
```

no rows selected

Just as equality is useful, so is inequality.

```
SELECT first_name, last_name, phone
FROM instructor
WHERE last_name <> 'Schorin'
```

FIRST_NAME	LAST_NAME	PHONE
Fernand	Hanks	2125551212
Tom	Wojick	2125551212
...		
Marilyn	Frantzen	2125551212
Irene	Willig	2125551212

9 rows selected.

In this example, all rows except the one with the last name 'Schorin' are retrieved. Inequality can also be expressed with the != notation.

THE GREATER THAN AND LESS THAN OPERATORS

The comparison operators >, <, >=, and <= can all be used to compare values in columns. In the following example, the >=, or *greater than or equal to*, operator is used to retrieve a list of course descriptions for which the course cost is greater than or equal to 1195.

```
SELECT description, cost
  FROM course
 WHERE cost >= 1195
DESCRIPTION                                COST
-----
Technology Concepts                        1195
Intro to Information Systems               1195
...
Database System Principles                 1195
Java Developer III                         1195

26 rows selected.
```

In this example, the value 1195 is not enclosed in single quotation marks because it is a number literal.

THE BETWEEN OPERATOR

The BETWEEN operator tests for a range of values.

```
SELECT description, cost
  FROM course
 WHERE cost BETWEEN 1000 AND 1100
DESCRIPTION                                COST
-----
Unix Tips and Techniques                    1095
Intro to the Internet                      1095
Intro to the BASIC Language                1095

3 rows selected.
```

BETWEEN is inclusive of both values defining the range; the result set includes courses that cost 1000 and 1100 and everything in between. In this example, the lower end of the range must be listed first.

104

BETWEEN is most useful for number and date comparisons, but it can also be used for comparing text strings in alphabetical order. Date comparisons are discussed in [Chapter 5](#).

105

THE IN OPERATOR

The IN operator works with a *list of values*, separated by commas, contained within a set of parentheses. The following query looks for courses for which the cost is either 1095 or 1595.


```
SELECT description, cost
FROM course
WHERE cost IN (1095, 1595)
```

DESCRIPTION	COST
-----	----
Programming Techniques	1595
Unix Tips and Techniques	1095
Intro to the Internet	1095
Intro to the BASIC Language	1095

4 rows selected.

THE LIKE OPERATOR

A very useful comparison operator is LIKE, which performs pattern matching, using the percent (%) and underscore (_) characters as wildcards. The percent wildcard is used to denote multiple characters, while the underscore wildcard is used to denote a single character. The following query retrieves rows where the last name begins with the uppercase letter S and ends in anything else.

```
SELECT first_name, last_name, phone
FROM instructor
WHERE last_name LIKE 'S%'
```

FIRST_NAME	LAST_NAME	PHONE
-----	-----	-----
Nina	Schorin	2125551212
Todd	Smythe	2125551212

2 rows selected.

The % character may be placed at the beginning, end, or anywhere within the literal text, but it must always be within the single quotation marks. This is also true of the underscore wildcard character, as in the following statement.

```
SELECT first_name, last_name
FROM instructor
WHERE last_name LIKE '_o%'
FIRST_NAME                                LAST_NAME
-----
Tom                                       Wojick
Anita                                   Morris
Charles                                Lowry

3 rows selected.
```

105

The WHERE clause returns only rows where the last name begins with any character but the second letter must be a lowercase o. The rest of the last name is irrelevant.

106



The LIKE operator works well for simple pattern matching. For a more complex pattern, you might want to consider using Oracle's regular expression functionality, discussed in [Chapter 16](#), "Regular Expressions and Hierarchical Queries."

THE NOT OPERATOR

All the previously mentioned operators can be negated with the NOT comparison operator (for example, NOT BETWEEN, NOT IN, NOT LIKE).

```
SELECT phone
FROM instructor
WHERE last_name NOT LIKE 'S%'
```

This query returns all the phone numbers of instructors with a last name that does not begin with the uppercase letter S. This SQL statement does not list LAST_NAME in the SELECT list. There is no rule about columns in the WHERE clause having to exist in the SELECT list.

THE IS NULL AND IS NOT NULL OPERATORS

Recall that NULL means an unknown value. The IS NULL and IS NOT NULL operators evaluate whether a data value is NULL or not. The following SQL statement returns courses that do not have a prerequisite.

```
SELECT course_id, prereq_id
FROM prereqs
WHERE prereq_id IS NULL
```

course_id	prereq_id
1	2
2	3
3	4
4	5
5	6
6	7
7	8
8	9
9	10
10	11
11	12
12	13
13	14
14	15
15	16
16	17
17	18
18	19
19	20
20	21
21	22
22	23
23	24
24	25
25	26
26	27
27	28
28	29
29	30
30	31
31	32
32	33
33	34
34	35
35	36
36	37
37	38
38	39
39	40
40	41
41	42
42	43
43	44
44	45
45	46
46	47
47	48
48	49
49	50
50	51
51	52
52	53
53	54
54	55
55	56
56	57
57	58
58	59
59	60
60	61
61	62
62	63
63	64
64	65
65	66
66	67
67	68
68	69
69	70
70	71
71	72
72	73
73	74
74	75
75	76
76	77
77	78
78	79
79	80
80	81
81	82
82	83
83	84
84	85
85	86
86	87
87	88
88	89
89	90
90	91
91	92
92	93
93	94
94	95
95	96
96	97
97	98
98	99
99	100

Null values represent the unknown; a null cannot be equal or unequal to any value or to another null. Therefore, you should always use the IS NULL or IS NOT NULL operator when testing for nulls. There are a few exceptions when nulls are treated differently and a null can be equal to another null. One such example is the use of DISTINCT (see [Lab 2.2](#)). You will learn about the exceptions in the treatment of nulls throughout this book.

106

107

Logical Operators

To harness the ultimate power of the WHERE clause, comparison operators can be combined with the help of the *logical operators* AND and OR. These logical operators are also referred to as *Boolean operators*. They group expressions, all within the same WHERE clause of a single SQL statement.

For example, the following SQL query combines two comparison operators with the help of the AND *Boolean* operator. The result shows rows where a course costs 1095 and the course description starts with the letter I.

```
SELECT description, cost
FROM course
WHERE cost = 1095
AND description LIKE 'I%'
DESCRIPTION                                COST
-----
Intro to the Internet                        1095
Intro to the BASIC Language                 1095
2 rows selected.
```

With just the = operator in the WHERE clause, the result set contains three rows. With the addition of the AND description LIKE 'I%', the result is further reduced to two rows.

PRECEDENCE OF LOGICAL OPERATORS

When AND and OR are used together in a WHERE clause, the AND operator always takes precedence over the OR operator, meaning that any AND conditions are evaluated first. If there are multiple operators of the same precedence, the left operator is executed before the right. You can manipulate the precedence in the WHERE clause with the use of parentheses. In the following SQL statement, the AND and OR logical operators are combined.

```
SELECT description, cost, prerequisite
FROM course
WHERE cost = 1195
      AND prerequisite = 20
      OR prerequisite = 25
```

DESCRIPTION	COST	PREREQUISITE
Hands-On Windows	1195	20
Systems Analysis	1195	20
Project Management	1195	20
GUI Design Lab	1195	20
Intro to SQL	1195	20
Intro to the BASIC Language	1095	25
Database System Principles	1195	25

7 rows selected.

The preceding SQL statement selects any record that has either a cost of 1195 and a prerequisite of 20 or just a prerequisite of 25, no matter what the cost. The sixth row, Intro to the BASIC Language, is selected because it satisfies the OR expression prerequisite = 25. The seventh row, Database System Principles, satisfies only one of the AND conditions, not both. However, the row is part of the result set because it satisfies the OR condition.

Here is the same SQL statement, but with parentheses to group the expressions in the WHERE clause.

```
SELECT description, cost, prerequisite
FROM course
WHERE cost = 1195
      AND (prerequisite = 20
          OR prerequisite = 25)
```

DESCRIPTION	COST	PREREQUISITE
Hands-On Windows	1195	20
Systems Analysis	1195	20
Project Management	1195	20
GUI Design Lab	1195	20
Intro to SQL	1195	20
Database System Principles	1195	25

```
6 rows selected.
```

The first expression selects only courses where the cost is equal to 1195. If the prerequisite is either 25 or 20,

then the second condition is also true. Both expressions need to be true for the row to be displayed. These are the basic rules of logical operators. If two conditions are combined with the AND operator, both conditions must be true; if two conditions are connected by the OR operator, only one of the conditions needs to be true for the record to be selected.

The result set returns six rows instead of seven. The order in which items in the WHERE clause are evaluated is changed by the use of parentheses and results in different output.



To ensure that your SQL statements are clearly understood, it is always best to use parentheses.

NULLS AND LOGICAL OPERATORS

SQL uses *tri-value logic*; this means a condition can evaluate to true, false, or unknown. (This is in contrast to Boolean logic, where a condition must be either true or false.) A row is returned when the condition evaluates to true. The following query returns rows from the COURSE table, starting with the words “Intro

to” as the description *and* a value equal or larger than 140 in the PREREQUISITE column.

108

```
SELECT description, prerequisite
FROM course
WHERE description LIKE 'Intro to%'
AND prerequisite >= 140
```

DESCRIPTION	PREREQUISITE
Intro to Programming	140
Intro to Unix	310

2 rows selected.

109

Rows with a null value in the PREREQUISITE column are not included because null is an unknown value. This null value in the column is not greater than or equal to 140. Therefore, the row Intro to Information Systems does not satisfy both conditions and is excluded from the result set. Following is the list of course descriptions with null values in the PREREQUISITE column. It shows the row Intro to Information Systems and the null value in the PREREQUISITE column.

```
SELECT description, prerequisite, cost
FROM course
WHERE prerequisite IS NULL
```

DESCRIPTION	PREREQUISITE	COST
Technology Concepts		1195
Intro to Information Systems		1195
Java for C/C++ Programmers		1195
Operating Systems		1195

4 rows selected.

The AND truth table in [Table 3.2](#) illustrates the combination of two conditions with the AND operator. Only if *both* conditions are true is a row returned for output. In this example, with the prerequisite being null, the condition is unknown, and therefore the row is not included in the result. The comparison against a null value yields unknown unless you specifically test for it with the IS NULL or IS NOT operators.

TABLE 3.2 AND Truth Table

	TRUE	FALSE UNKNOWN
TRUE	TRUE	FALSE UNKNOWN
FALSE	FALSE	FALSE FALSE
UNKNOWN	UNKNOWN	FALSE UNKNOWN

For the OR condition, just one of the conditions needs to be true. Again, let's examine how nulls behave under this scenario, using the same query, but this time with the OR operator. The Intro to Information Systems course is now listed because it satisfies the 'Intro to%' condition only. In addition, rows such as DB

Programming with Java do not start with "Intro to" as the description but satisfy the second condition, which is a prerequisite of greater than or equal to 140.

109
110

```
SELECT description, prerequisite
FROM course
WHERE description LIKE 'Intro to%'
OR prerequisite >= 140
```

DESCRIPTION	PREREQUISITE
Intro to Information Systems	
Intro to Programming	140
Programming Techniques	204
Intro to Java Programming	80
Intro to Unix	310
Database Design	420
Internet Protocols	310
Intro to SQL	20
Oracle Tools	220
Intro to the Internet	10
Intro to the BASIC Language	25
Java Developer III	350
DB Programming with Java	350

13 rows selected

[Table 3.3](#) shows the truth table for the OR operator; it highlights the fact that just one of the conditions needs to be true for the row to be returned in the result set. It is irrelevant if the second condition evaluates to false or unknown.

TABLE 3.3 OR Truth Table

	TRUE	FALSE	UNKNOWN
TRUE	TRUE	TRUE	TRUE
FALSE	TRUE	FALSE	UNKNOWN
UNKNOWN	TRUE	UNKNOWN	UNKNOWN

When you negate a condition with the NOT operator and the value you are comparing against is a null value, it also results in a null (see [Table 3.4](#)). The following query demonstrates that none of the null prerequisites are included in the result set.

TABLE 3.4 NOT Truth Table

	TRUE	FALSE	UNKNOWN
NOT	FALSE	TRUE	UNKNOWN

```
SELECT description, prerequisite
FROM course
WHERE NOT prerequisite >= 140
```

```
DESCRIPTION                                PREREQUISITE
-----
Intro to the Internet                        10
Hands-On Windows                           20
Systems Analysis                           20
Project Management                          20
GUI Design Lab                              20
Intro to SQL                               20
Intro to the BASIC Language                 25
Database System Principles                 25
Intro to Java Programming                   80
PL/SQL Programming                         80
Intermediate Java Programming              120
Advanced Java Programming                  122
Java Developer I                           122
Java Developer II                          125
Basics of Unix Admin                      130
Network Administration                     130
Advanced Unix Admin                        132
Unix Tips and Techniques                   134

18 rows selected
```

110

111

LAB 3.1 EXERCISES

- a) Write a SELECT statement that lists the last names of students living in either zip code 10048, 11102, or 11209.
- b) Write a SELECT statement that lists the first and last names of instructors with the letter i (either uppercase or lowercase) in their last name, living in zip code 10025.
- c) Does the following statement contain an error? Explain.

```
SELECT last_name  
FROM instructor  
WHERE created_date =  
modified_by
```

- d) What do you observe when you execute the following SQL statement?

```
SELECT course_no, cost  
FROM course  
WHERE cost BETWEEN 1500 AND  
1000
```

- e) Execute the following query and determine how many rows the query returns.

```
SELECT last_name, student_id
FROM student
WHERE ROWNUM <= 10
```

111

- f) Write a SELECT statement that lists descriptions of courses for which there are prerequisites and that cost less than 1100.
- g) Write a SELECT statement that lists the cost of courses without a known prerequisite; do not repeat the cost.

112

LAB 3.1 EXERCISE ANSWERS

- a) Write a SELECT statement that lists the last names of students living in either zip code 10048, 11102, or 11209.

ANSWER: The SELECT statement selects a single column and uses the IN comparison operator in the WHERE clause.

```
SELECT last_name
FROM student
WHERE zip IN ('10048', '11102', '11209')
LAST_NAME
-----
Masser
Allende
Winnicki
Wilson
Williams
McLean
Lefkowitz

7 rows selected.
```

The statement can also be written using the equal operator (=), in combination with the logical operator OR, and yields the same result set.

```
SELECT last_name
FROM student
WHERE zip = '10048'
       OR zip = '11102'
       OR zip = '11209'
```

There are times when a SELECT statement can be written more than one way. The preceding statements are logically equivalent.

- b)** Write a SELECT statement that lists the first and last names of instructors with the letter i (either uppercase or lowercase) in their last name, living in zip code 10025.

ANSWER: The SELECT statement selects two columns and uses the LIKE, =, and AND and OR logical operators, combined with parentheses, in the WHERE clause.

```
SELECT first_name, last_name
FROM instructor
WHERE (last_name LIKE '%i%' OR last_name LIKE '%I%')
AND zip = '10025'
```

FIRST_NAME	LAST_NAME
Tom	Wojcik
Nina	Schouin

2 rows selected.

The LIKE operator must be used twice in this example because there is no way of knowing whether there is an uppercase or lowercase i anywhere in the last name. You must test for both conditions, which cannot be done using a single LIKE operator. If one of the OR conditions is true, the expression is true.

If you need to search for the % symbol within a column value, you can use a SQL function or an escape character. You'll learn more about this in [Chapter 4](#), "Character, Number, and Miscellaneous Functions."

- c) Does the following statement contain an error? Explain.

```
SELECT last_name
       FROM instructor
      WHERE created_date =
modified_by
```

ANSWER: Yes. The two columns in the WHERE clause are not the same data type, and the Oracle database returns an error when this statement is executed.

You get an error similar to the following when you execute the statement.

```
SQL> SELECT last_name
      2     FROM instructor
      3     WHERE created_date = modified_by
      4 /
      WHERE created_date = modified_by
                        *
ERROR at line 3:
ORA-01858: a non-numeric character was found where a numeric was
expected
```

There are times when the data types of columns do not agree, and you need to convert from one data type to another. You will learn about these circumstances in [Chapter 5](#). (In this exercise example, data conversion is not fruitful because the data in these two columns is of a very different nature.)

- d) What do you observe when you execute the following SQL statement?

```
SELECT course_no, cost
      FROM course
      WHERE cost BETWEEN 1500 AND
1000
no rows selected
```

ANSWER: The query returns no rows. Although there are courses that cost between

1000 and 1500, the BETWEEN clause requires the lower end of the range to be listed first. If the query is rewritten as follows, it returns rows.

```
SELECT course_no, cost
       FROM course
      WHERE cost BETWEEN 1000 AND
1500
```

113

114

BETWEEN AND TEXT LITERALS

As mentioned previously, BETWEEN is most often used for numbers and dates, which you will learn about in [Chapter 5](#). You can apply the BETWEEN functions to text columns, as shown in the next example, which utilizes the BETWEEN operator with text literals W and Z. The query lists the student's ID and last name. Any students whose last name begins with the letter Z are not included, because the STUDENT table has no student with a last name of the single letter Z. If a student's last name were spelled "waldo," this student would not be included in the result, because the WHERE clause is looking only for last names that fall between the uppercase letters W and Z.

```
SELECT student_id, last_name
       FROM student
      WHERE last_name BETWEEN 'W' AND 'Z'
     STUDENT_ID LAST_NAME
     *****
           142 Waldman
           ...
           241 Yourish
11 rows selected.
```

If you are looking for “waldo”, regardless of the case, use the OR operator to include both conditions.

```
SELECT student_id, last_name
FROM student
WHERE last_name BETWEEN 'W' AND
'Z'
OR last_name BETWEEN 'w' AND 'z'
```

Here is another example of how you can use the BETWEEN and the >= and <= operators with text literals.

```
SELECT description
FROM grade_type
WHERE description BETWEEN
'Midterm' and 'Project'
```

This would be equivalent to the following.

```
SELECT description
FROM grade_type
WHERE description >= 'Midterm'
AND description <= 'Project'
DESCRIPTION
-----
Midterm
Participation
Project

3 rows selected.
```

- e) Execute the following query and determine how many rows the query returns.

```
SELECT last_name, student_id
FROM student
WHERE ROWNUM <= 10
```

114

115

ANSWER: The query returns 10 rows. The WHERE clause uses the pseudocolumn ROWNUM, which restricts the result to the first 10 or fewer rows it finds, and there is no particular order. A pseudocolumn is not a real column that exists on a table; you can select the column, but you cannot manipulate its value.

```
SELECT ROWNUM, last_name, student_id
FROM student
WHERE ROWNUM <= 10
```

ROWNUM	LAST_NAME	STUDENT_ID
1	Kocka	230
2	Jung	232
3	Mulroy	233
4	Brendler	234
...		
9	Scrittorale	240
10	Yourish	241

```
10 rows selected.
```

The next statement shows the value of the ROWNUM pseudocolumn in the SELECT list. The first row displays the ROWNUM value 1, the second the ROWNUM value 2, and so on. The ROWNUM pseudocolumn is useful if you want to limit the number of rows returned by a query. You will see additional examples of this and other pseudocolumns in subsequent chapters.

```
SELECT description, cost, prerequisite
FROM course
WHERE prerequisite IS NOT NULL
AND cost < 1100
```

DESCRIPTION	COST	PREREQUISITE
Intro to the Internet	1095	10
Intro to the BASIC Language	1095	25
Unix Tips and Techniques	1095	134

3 rows selected.

- f) Write a SELECT statement that lists descriptions of courses for which there are prerequisites and that cost less than 1100.

ANSWER: The SELECT statement uses the IS NOT NULL and less than (<) comparison operators in the WHERE clause.

```
SELECT ROWNUM, last_name, student_id
FROM student
WHERE ROWNUM <= 10
```

ROWNUM	LAST_NAME	STUDENT_ID
1	Kocka	230
2	Jung	232
3	Mulroy	233
4	Brendler	234
...		
9	Scrittorale	240
10	Yourish	241

10 rows selected.

115

116

Both conditions need to be true for the row to be returned. If one of the conditions is not met, the row simply is not selected for output.

- g) Write a SELECT statement that lists the cost of courses without a known prerequisite; do not repeat the cost.

ANSWER: The SELECT statement selects a single column in combination with DISTINCT and uses the IS NULL comparison operator in the WHERE clause.

```
SELECT DISTINCT cost
FROM course
WHERE prerequisite IS NULL
COST
-----
1195

1 row selected.
```

116

Lab 3.1 Quiz

In order to test your progress, you should be able to answer the following questions.

1) Comparison operators always compare two values only.

_____ **a)** True

_____ **b)** False

2) The BETWEEN operator uses a list of values.

_____ **a)** True

_____ **b)** False

3) The following statement is incorrect.

```
SELECT first_name, last_name  
FROM student  
WHERE employer = NULL
```

_____ **a)** True

_____ **b)** False

4) The following statement is incorrect.

```
SELECT description  
FROM course  
WHERE cost NOT LIKE (1095, 1195)
```


_____ **a) True**

_____ **b) False**

5) The following statement is incorrect.

```
SELECT city
FROM zipcode
WHERE state != 'NY'
```

_____ **a) True**

_____ **b) False**

6) The following statement returns rows in the STUDENT table where the last name begins with the letters SM.

```
SELECT last_name, first_name
FROM student
WHERE last_name = 'SM%'
```

_____ **a) True**

_____ **b) False**

ANSWERS APPEAR IN APPENDIX A.

117

Lab 3.2 The ORDER BY Clause

LAB OBJECTIVES

After this lab, you will be able to:

- ▶ Custom Sort Query Results
- ▶ Use Column Aliases
- ▶ Understand SQL Error Messages

The SQL language's ORDER BY clause allows you to sort your query result in various ways. You learn how column aliases are useful for changing the display name of a column. As you write more SQL statements, you will inevitably make mistakes; this lab provides some suggestions to help you better understand Oracle's error messages.

Using the ORDER BY Clause

Recall from [Chapter 1](#), "SQL and Data," that data is not stored in a table in any particular order. In all the examples used thus far, the result sets display data in the order in which they happen to be returned from the database. However, you might want to view data in a

certain order, and you can use the ORDER BY clause to accomplish this by ordering the data any way you wish.

For example, the following statement retrieves a list of course numbers and descriptions for courses for which there is no prerequisite, in alphabetical order by their descriptions:

```
SELECT course_no, description
FROM course
WHERE prerequisite IS NULL
ORDER BY description
COURSE_NO DESCRIPTION
-----
      20 Intro to Information Systems
    146 Java for C/C++ Programmers
    310 Operating Systems
      10 Technology Concepts

4 rows selected.
```

By default, when ORDER BY is used, the result set is sorted in *ascending* order; or you can be explicit by adding the abbreviation ASC after the column. If descending order is desired, you use the abbreviation DESC after the column in the ORDER BY clause.

118

119

```
SELECT course_no, description
FROM course
WHERE prerequisite IS NULL
ORDER BY description DESC
COURSE_NO DESCRIPTION
-----
      10 Technology Concepts
     310 Operating Systems
    146 Java for C/C++ Programmers
      20 Intro to Information Systems

4 rows selected.
```

Instead of listing the name of the column to be ordered, you can list the sequence number of the column in the SELECT list. The next SQL statement returns the same result as the prior SQL statement but uses a different ORDER BY clause. The number 2 indicates the second column of the SELECT list.

```
SELECT course_no, description
FROM course
WHERE prerequisite IS NULL
ORDER BY 2 DESC
```

A result set can be sorted by more than one column. The columns you want to sort by need only be included in the ORDER BY clause, separated by commas. The ORDER BY clause is always the last clause in a SQL statement.

DISTINCT AND ORDER BY

The ORDER BY clause often contains columns listed in the SELECT clause, but it is also possible to use ORDER BY on columns that are not selected. One exception is columns qualified using the DISTINCT keyword: If the SELECT list contains DISTINCT, the column(s) the keyword pertains to must also be listed in the ORDER BY clause.

The next example shows that the STUDENT_ID column is not a column listed in the DISTINCT SELECT list and therefore results in an Oracle error message.

```
SQL> SELECT DISTINCT first_name,  
last_name  
2 FROM student  
3 WHERE zip = '10025'  
4 ORDER BY student_id  
5/  
ORDER BY student_id  
*
```

```
ERROR at line 4:  
ORA-01791: not a SELECTed  
expression
```

119

NULLS FIRST AND NULLS LAST

The following statement orders the COST column by the default sort order. The row with a COST column value of NULL is the last row in the sort order.

```
SELECT DISTINCT cost
FROM course
ORDER BY cost
      COST
-----
      1095
      1195
      1595

4 rows selected.
```

You can change the ordering of the nulls with the NULLS FIRST or NULLS LAST option in the ORDER BY clause, as shown in the next statement. Here, the requested order is to list the NULL value first, followed by the other values in the default ascending sort order.

```
SELECT DISTINCT cost
FROM course
ORDER BY cost NULLS FIRST
      COST
-----
      1095
      1195
      1595

4 rows selected.
```

Sorting Data Using SQL Developer's GUI Functionality

Instead of using ORDER BY, you can order data by using SQL Developer. In the Results tab, double-click one of the columns in the column header to sort. The up/down arrow (see [Figure 3.1](#)) indicates whether the sort is descending or ascending.

FIGURE 3.1 Column sorted in descending order



FIRST_NAME
Zelman
Z.A.
Yvonne
Yu
Winsome

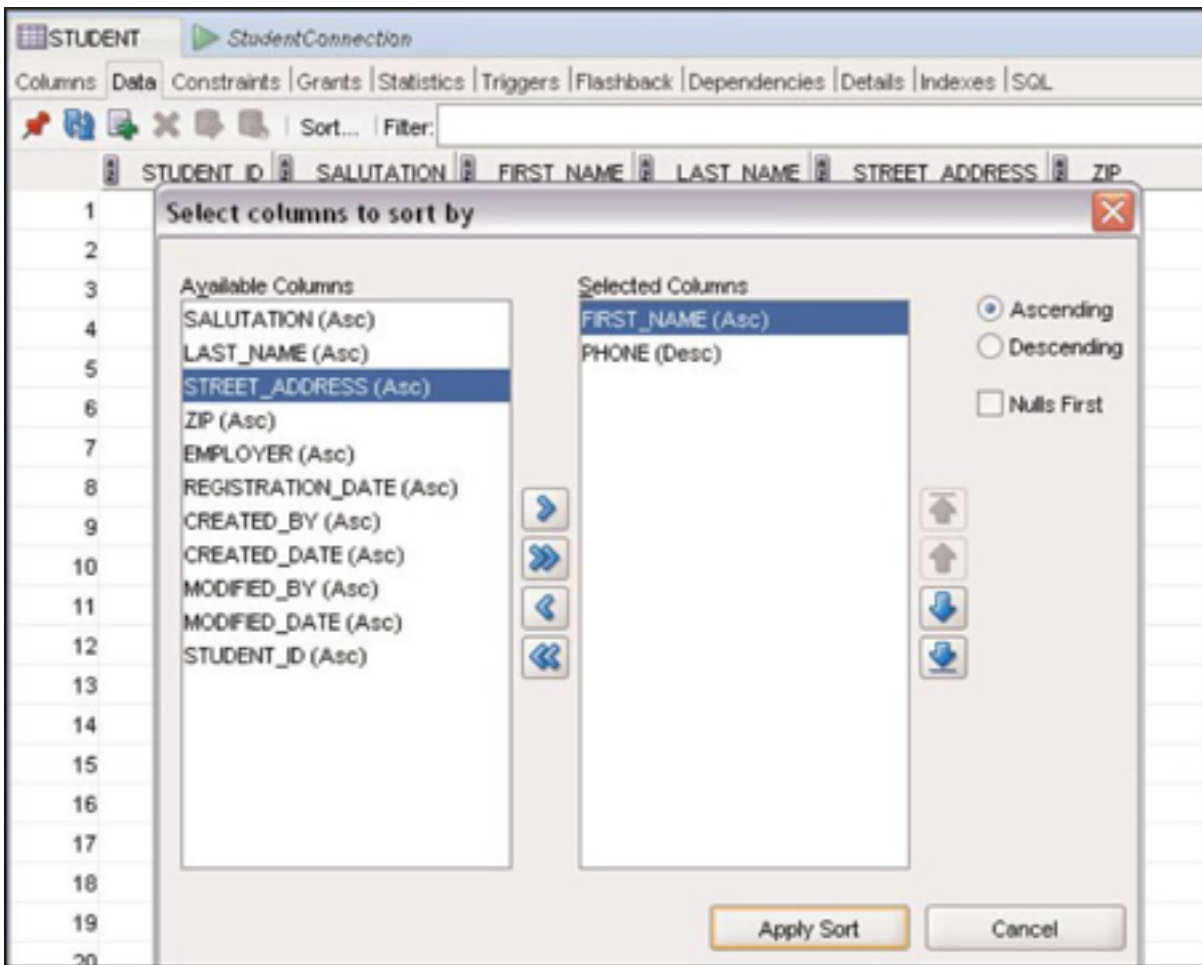
120

121

Aside from writing SQL statements, SQL Developer contains rich functionality that allows you to retrieve, filter, and sort data without having to write a SQL statement. In [Chapter 2](#), “SQL: The Basics,” you learned about the SQL Developer Data tab, which allows you to retrieve data from a table without writing a SQL statement. You can use this tab to perform sort and filter functionality. You will see both a Filter and Sort box onscreen. When you click the Sort option, a sort dialog

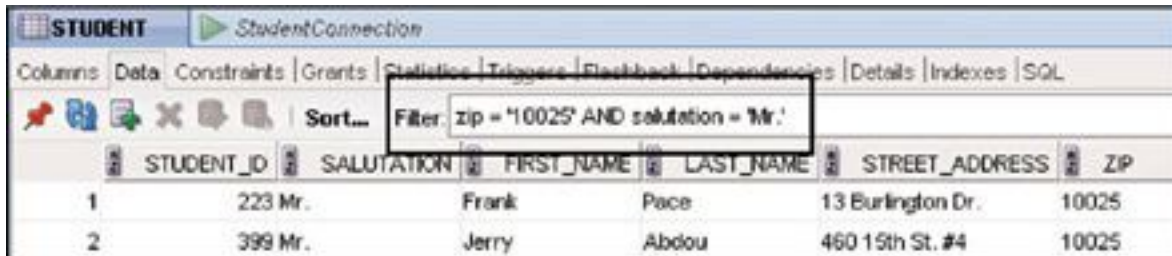
box similar to [Figure 3.2](#) appears. Here, you can choose multiple sort columns along with a variety of sort choices.

FIGURE 3.2 The Data tab sort options



The other option on the Data tab is the Filter box. Here you can enter WHERE clause conditions and then press the Enter key to retrieve only the chosen data. [Figure 3.3](#) shows the Filter box with criteria that consists of the ZIP and SALUTATION columns of the STUDENT table.

FIGURE 3.3 The Data tab's Filter criteria box



STUDENT_ID	SALUTATION	FIRST_NAME	LAST_NAME	STREET_ADDRESS	ZIP
1	223 Mr.	Frank	Pace	13 Burlington Dr.	10025
2	399 Mr.	Jerry	Abdou	460 15th St. #4	10025

121

122



Although SQL Developer allows you to perform basic tasks with a few mouse clicks, the tool does not eliminate the need to understand the SQL language, nor does it perform advanced SQL functionality. You should consider SQL Developer a useful productivity tool to make you a more efficient and intelligent user of the SQL language. The focus of this book is on the Oracle SQL language, and SQL Developer can help you with this learning experience.

Column Alias

A column alias can be used in the SELECT list to give a column or an expression an alias; it can make the result much easier to read. In the next example, different forms

of a column alias are used to take the place of the column name in the result set. An alias may contain one or more words or be spelled in exact case when enclosed in double quotation marks. The optional keyword AS can precede the alias name.

```
SELECT first_name first,
       first_name "First Name",
       first_name AS "First"
FROM student
WHERE zip = '10025'
```

FIRST	First Name	First
-----	-----	-----
Nicole	Nicole	Nicole
Jerry	Jerry	Jerry
Frank	Frank	Frank

3 rows selected.

You can use the column alias to order by a specific column.

```
SELECT first_name first, first_name "First Name",
       first_name AS "First"
FROM student
WHERE zip = '10025'
ORDER BY "First Name"
```

FIRST	First Name	First
-----	-----	-----
Frank	Frank	Frank
Jerry	Jerry	Jerry
Nicole	Nicole	Nicole

3 rows selected.

Comments in SQL Statements

Placing comments or remarks in a SQL statement is very useful for documenting purpose, thoughts, and ideas. Comments are very handy when you've developed multiple statements that are saved into a file (called a *script*).

122

You must identify a comment as such; otherwise, you will receive an error when you run the command. There are two different types of comments: single-line comments denoted with double dashes and multiline comments spanning one or multiple lines. A multiline comment starts with the opening comment `/*` and ends with the closing comment `*/`.

123

Following are examples of comments.

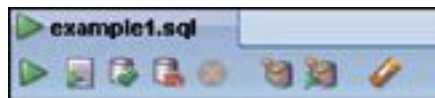
```
/* Multi-line comment
   SELECT *
     FROM student;
*/
-- This is a single-line comment!
SELECT DISTINCT state
   FROM zipcode;
SELECT instructor_id, -- Comment
within a SQL statement!
```

```
        zip /* Another comment example  
*/  
FROM instructor;
```

Saving Your SQL Statements

You might want to save some statements for later use. Clicking the Save icon or choosing File, Save from the menu brings up the Save dialog box. By default, the file extension is .sql. After you save, the StudentConnection tab is renamed to the file name. For example, [Figure 3.4](#) shows the file saved as example1.sql.

FIGURE 3.4 The file name of the saved file



You can retrieve the file by using the File, Open menu option. When you place the cursor in the statement you want to execute and click the Execute Statement button or press the F9 key, SQL Developer presents you with a Select Connection dialog box to choose the connection information.

If you are using SQL*Plus and you want to save and rerun some of your commands, see [Chapter 14](#), “The Data Dictionary, Scripting, and Reporting.”

Understanding Oracle Error Messages

As you begin to learn SQL, you will inevitably make mistakes when writing statements. Oracle returns an error number and error message to inform you of any mistake. Some error messages are easy to understand; others are not. While I cannot anticipate every possible error you may encounter, I point out common mistakes. Here are some general guidelines for dealing with Oracle errors.

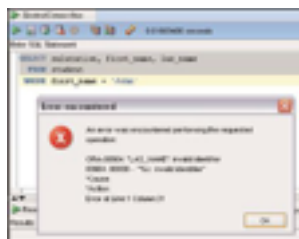
123

124

READING THE ORACLE ERROR MESSAGE CAREFULLY

Oracle tells you on which line an error occurred. [Figure 3.5](#) shows the dialog box you see after executing an erroneous SQL statement.

FIGURE 3.5 Error in a SQL statement



In this example, the error is very easy to spot, and the error message is self-explanatory. One of the column names is invalid; the ORA-00904 error says that it is an invalid identifier. Oracle points out the error by indicating the line and column position number; this indicates where within the line the error is located. In this case, it is the misspelled LAST_NAME column name.

RESOLVING ONE ERROR AT A TIME

Sometimes, you may have multiple errors in a single SQL statement. The Oracle *parser*, which checks the syntax of all statements, starts checking from the end of the entire statement.

The error message may leave you clueless about what could be wrong with the query. In fact, the statement in [Figure 3.6](#) contains three errors, one in each line.

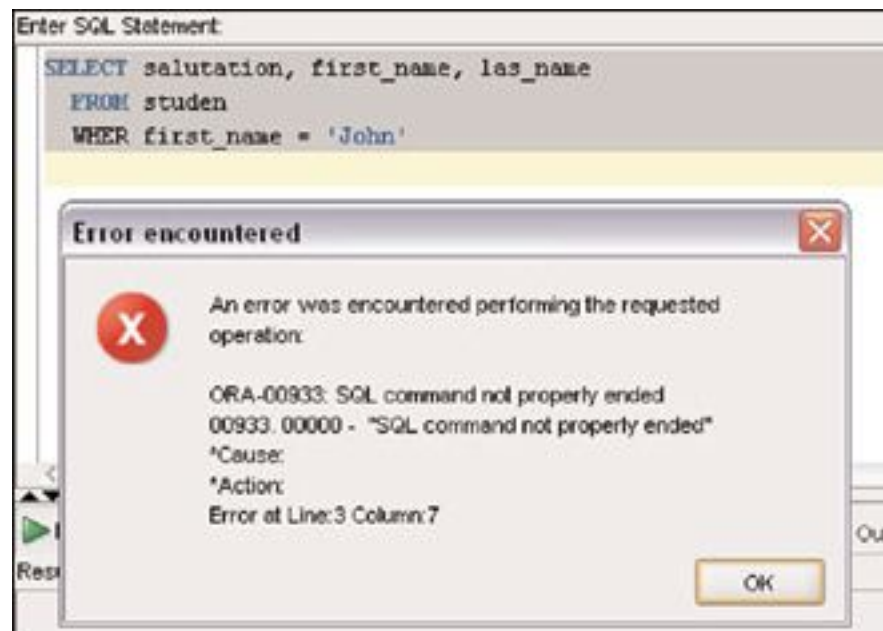
Because the parser works its way backward, it complains about the first error on line 3. The position of the column name may even suggest that there is something wrong with the spelling of the FIRST_NAME column. But, in fact, it is spelled correctly; otherwise, you would see the ORA-00904 invalid identifier error listed, as in the previous

example. The WHERE keyword is missing the final letter E; therefore, Oracle cannot interpret what you are attempting to do. The color coding in SQL Developer can help you spot the missing letter more easily because the word is not blue like the other keywords.

124

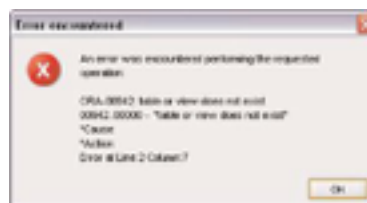
125

FIGURE 3.6 SQL keyword error



After you correct this error and re-execute, you see line 2 reported (see [Figure 3.7](#)), because, again, the parser works its way backward.

FIGURE 3.7 Table name error



Here, the table name is misspelled, and Oracle indicates that such a table does not exist.

The last error in the statement, found on line 1, is the misspelled `LAST_NAME` column name. The parser will report this error as the last error. You can avoid some misspelling errors when you use SQL Developer's syntax auto-completion feature, which allows you to simply pick from a list.

In any case, if you are unsure about the spelling of a column or table name, you can also review the Column tab to list the column names. You can also refer to [Appendix D](#), "STUDENT Database Schema," for a diagram showing the table and column names.

125

126

DOUBLE-CHECKING THE SYNTAX OF YOUR STATEMENT

Simple typos, such as a stray period or comma, a missing space, or an unpaired quotation mark, can cause very strange and seemingly unrelated error messages that may have nothing to do with the problem. Therefore, it's important to carefully reread the statement or simply retype it. After you look at a statement for a long time, the error may not be apparent. Perhaps put it aside, take a break, and look at

it with a fresh mind later, or ask someone for help in spotting the error.

LOOKING UP THE ORACLE ERROR NUMBER

You can look up the Oracle error number in the *Oracle Database Error Messages manual*. An error that starts with an ORA message number is a database-related error, whereas an error with a PLS prefixes indicates a PL/SQL language-specific error, and a TNS error number relates to a TNS connectivity problem between the database server and the client. When you found the error number in the manual, you will see the reason for the error and a recommended action on how to correct it. The recommended action may be general or very specific, again depending on what type of error occurred.

Often the most convenient way to look up error information is to review the online manual on the Oracle.com Web site. Also refer to [Appendix G](#), “Navigating the Oracle Documentation,” and [Appendix H](#), “Resources.” These appendixes provide tips on how to find the needed information.

In some operating systems, such as UNIX, Linux, and VMS, you can also use the Oracle program `oerr` to look up the error message from the operating system prompt. This does not work in the Windows environment. For example, to look up the ORA-00939 error, you type the following at the UNIX operating system prompt (indicated with the `$` sign).

```
$ oerr ora 00939
00939, 00000, " too many arguments
for function"
// *Cause: The function was
referenced with too many arguments.
// *Action: Check the function
syntax and specify only the
// required number of arguments.
$
```

LAB 3.2 EXERCISES

- a) Write a SELECT statement that lists each city and zip code in New York or Connecticut. Sort the results in ascending order by zip code.
- b) Write a SELECT statement that lists course descriptions and their prerequisite course numbers, sorted in ascending order by

description. Do not list courses that do not have a prerequisite.

- c) Show the salutation, first name, and last name of students with the last name Grant. Order the results by salutation in descending order and by first name in ascending order.
- d) Execute the following query. What do you observe about the last row returned by the query?

```
SELECT student_id, last_name
FROM student
ORDER BY last_name
```

126

127

LAB 3.2 EXERCISE ANSWERS

- a) Write a SELECT statement that lists each city and zip code in New York or Connecticut. Sort the results in ascending order by zip code.

ANSWER: The SELECT statement selects two columns, uses the equal operator and OR logical operator to combine expressions in the WHERE clause, and uses ORDER BY and a single column to sort the results in ascending order.

```
SELECT description, prerequisite
  FROM course
 WHERE prerequisite IS NOT NULL
 ORDER BY description
```

DESCRIPTION	PREREQUISITE
-----	-----
Advanced Java Programming	122
Advanced Unix Admin	132
...	
Systems Analysis	20
Unix Tips and Techniques	134

26 rows selected.

Alternatively, the WHERE clause can be written as follows.

```
WHERE state IN ('NY', 'CT')
```

- b)** Write a SELECT statement that lists course descriptions and their prerequisite course numbers, sorted in ascending order by description. Do not list courses that do not have a prerequisite.

ANSWER: The following query shows the use of the IS NOT NULL comparison operator in the WHERE clause. The result is sorted by the DESCRIPTION column in ascending order.

```
SELECT city, zip
      FROM zipcode
     WHERE state = 'NY'
          OR state = 'CT'
     ORDER BY zip
CITY                                ZIP
-----
Ansonia                            06401
Middlefield                        06455
...
Hicksville                         11802
Endicott                           13760

142 rows selected.
```

Alternatively, the ORDER BY clause can be written as follows.

```
ORDER BY 1
```

127

You can even use the column alias.

```
SELECT description "Descr",
prerequisite
      FROM course
     WHERE prerequisite IS NOT NULL
     ORDER BY "Descr"
```

128

In most of the previous examples, the SELECT list is taking up one line only. Spreading it over several lines sometimes makes it easier to read, and this is perfectly acceptable formatting.

Separating columns in the SELECT list on separate lines and indenting them makes for better readability of your statement. The following SELECT statement has multiple columns in the SELECT list.

```
SELECT description, prerequisite,
       cost, modified_date
FROM course
WHERE prerequisite IS NOT NULL
ORDER BY description
```

DESCRIPTION	PREREQUISITE	COST	MODIFIED_
Advanced Java Programming		122 1195	05-APR-07
...			
Unix Tips and Techniques		134 1095	05-APR-07

26 rows selected.

The result displays the column `MODIFIED_DATE` with an abbreviated column heading name. This is typical of the output in SQL*Plus. Because a fixed-width font is more readable than a screenshot, this book has adopted the SQL*Plus output format for many of the statements.

- c) Show the salutation, first name, and last name of students with the last name Grant. Order the result by salutation in descending order and by first name in ascending order.

ANSWER: The ORDER BY clause contains two columns: SALUTATION and FIRST_NAME. The salutation is sorted first in descending order. Within each salutation, the first name is sorted in ascending order.

```
SELECT salutation, first_name, last_name
FROM student
WHERE last_name = 'Grant'
ORDER BY salutation DESC, first_name ASC
```

SALUT	FIRST_NAME	LAST_NAME
Ms.	Eilene	Grant
Ms.	Verona	Grant
Mr.	Omaira	Grant
Mr.	Scott	Grant

4 rows selected.

Again, you can instead write the query with an ORDER BY clause.

```
ORDER BY 1 DESC, 2 ASC
```

128

Or you can use the default order for the second column, which is ASC and can be omitted.

```
ORDER BY 1 DESC, 2
```

129

If you give your column a column alias, you can also use the column alias in the ORDER BY clause.


```
SELECT salutation "Sal", first_name "First Name",
       last_name "Last Name"
FROM student
WHERE last_name = 'Grant'
ORDER BY "Sal" DESC, "First Name" ASC
```

Sal	First Name	Last Name
Ms.	Eilene	Grant
Ms.	Verona	Grant
Mr.	Omaira	Grant
Mr.	Scott	Grant

4 rows selected.

- d) Execute the following query. What do you observe about the last row returned by the query?

```
SELECT student_id, last_name
FROM student
ORDER BY last_name
```

ANSWER: The student with the STUDENT_ID of 206 has the last name entered in lowercase. When ordering the result set, the lowercase letters are listed after the uppercase letters.

```
STUDENT_ID LAST_NAME
-----
119 Abdou
399 Abdou
...
184 Zuckerberg
206 annunziato

268 rows selected.
```

Lab 3.2 Quiz

In order to test your progress, you should be able to answer the following questions.

- 1) The following is the correct order of all clauses in this SELECT statement.

```
SELECT ...  
    FROM ...  
    ORDER BY ...  
    WHERE ...
```

_____ a) True

_____ b) False

- 2) You must explicitly indicate whether an ORDER BY is ascending.

_____ a) True

_____ b) False

- 3) The following statement is correct.

```
SELECT *  
    FROM instructor  
    ORDER BY phone
```

_____ a) True

_____ **b) False**

4) The following statement is incorrect.

```
SELECT description "Description",  
prerequisite AS prereqs, course_no  
"Course#"  
FROM course  
ORDER BY 3, 2
```

_____ **a) True**

_____ **b) False**

5) You can order by a column you have not selected.

_____ **a) True**

_____ **b) False**

ANSWERS APPEAR IN APPENDIX A.

130

131

WORKSHOP

The projects in this section are meant to prompt you to utilize all the skills you have acquired throughout this chapter. The answers to these projects can be found at the companion Web site to this book, located at www.oraclesqlbyexample.com.

- 1) Create a SQL statement that retrieves data from the COURSE table for courses that cost 1195 and whose descriptions start with Intro, sorted by their prerequisites.
- 2) Create another SQL statement that retrieves data from the STUDENT table for students whose last names begin with A, B, or C and who work for Competrol Real Estate, sorted by their last names.
- 3) Write a SQL statement that retrieves all the descriptions from the GRADE_TYPE table, for rows that were modified by the user MCAFFREY.
- 4) Save all three SQL statements in a file called Workshop_Ch3.sql.