PRINTED BY: Monica Gonzalez <monicazalez1@hotmail.com>. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

CHAPTER 4 Character, Number, and Miscellaneous Functions

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:

- Character Functions
- Number Functions
- Miscellaneous Single-Row Functions

Functions are a very useful and powerful part of the SQL language. They can transform data in a way that is different from the way it is stored in a database. A function is a type of formula whose result is one of two things: either a *transformation*, such as a change in the name of a student to uppercase letters, or *information*, such as the length of a word in a column. Most functions share similar characteristics, including a name, and typically at least one input parameter, also called an *argument*, inside a pair of matching parentheses.

```
function_name(input_parameter)
```

PRINTED BY: Monica Gonzalez <monicazalez1@hotmail.com>. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

Each function in this chapter and the next chapter is performed on a single row. This chapter discusses the character, number, and miscellaneous functions. Chapter 5, "Date and Conversion Functions," reviews the date-related functions together with data type conversion functions. Single-row functions are in contrast to aggregate functions, which are performed against multiple rows. You will learn about aggregate functions in Chapter 6, "Aggregate Functions, GROUP BY, and HAVING Clauses."

Data Types

Each value in Oracle has a data type associated with it. A data type determines the value's attributes and acceptable values. For example, you cannot enter a text value into a NUMBER data type column or enter an invalid date, such as 32-DEC-2008, into a DATE data type column. In most SQL operations, you use the NUMBER, VARCHAR2, and DATE data types. These are the 133
134 commonly used data types where the vast majority of data is stored. This chapter concentrates on functions related to character and numeric data.

PRINTED BY: Monica Gonzalez <monicazalez1@hotmail.com>. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

Reading Syntax Diagrams

In this chapter and the following chapters, you will learn about many essential SQL functions and commands. The syntax of the individual commands or functions is listed together with many examples of usage. Table 4.1 lists the symbols that describe the syntax usage.

PRINTED BY: Monica Gonzalez <monicazalez1@hotmail.com>. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

TABLE 4.1 Syntax Symbols

SYMBOL	**USAGE**
[ ]	Square brackets enclose syntax options.
{ }	Braces enclose items of which only one is required.
|	A vertical bar denotes options.
...	Three dots indicate that the preceding expression can be repeated.
Delimiters	Delimiters other than brackets, braces, bars, or the three dots must be entered exactly as shown in the syntax. Examples of such delimiters are commas, parentheses, and so on.
CAPS	Words in all capital letters indicate the Oracle keywords that identify the individual elements of the SQL command or the name of the SQL function. The case of the keyword or command does not matter but for readability is in uppercase letters.

PRINTED BY: Monica Gonzalez <monicazalez1@hotmail.com>. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

<u>UNDERLINE</u>	Default values are underlined.

LAB 4.1 Character Functions

LAB OBJECTIVES

After this lab, you will be able to:

- Use a Character Function in a SQL Statement
- Concatenate Strings

All character functions require alphanumeric input parameters. The input can be a *text literal* or *character literal*, sometimes referred to as a *string*, or *text, constant*, or a column of data type VARCHAR2, CHAR, or CLOB. Text literals are always surrounded by single quotation marks. This lab discusses the most frequently used character functions.

The LOWER Function

The LOWER function transforms data into lowercase. The following query shows how both a column and a text constant serve as individual parameters for the LOWER function.

PRINTED BY: Monica Gonzalez <monicazalez1@hotmail.com>. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

```
SELECT state, LOWER(state), LOWER('State')
  FROM zipcode
ST LO LOWER
-- -- -----
PR pr state
MA ma state
...
NY ny state
NY ny state

227 rows selected.
```

The first column in the SELECT list displays the STATE column in the ZIPCODE table, without any transformation. The second column uses the LOWER function to display the values of the STATE column in lowercase letters. The third column of the SELECT list transforms the text constant ‘State’ into lowercase letters. Text constants used in a SELECT statement are repeated for every row of resulting output. 135

136

The UPPER and INITCAP Functions

The UPPER function is the exact opposite of the LOWER function: It transforms data into uppercase. The INITCAP function capitalizes the first letter of a word and lowercases the rest of the word.

```
SELECT UPPER(city) as "Upper Case City",   state,
       INITCAP(state)
  FROM zipcode
 WHERE zip = '10035'
Upper Case City           ST IN
------------------------- -- --
NEW YORK                  NY Ny

1 row selected.
```

PRINTED BY: Monica Gonzalez <monicazalez1@hotmail.com>. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

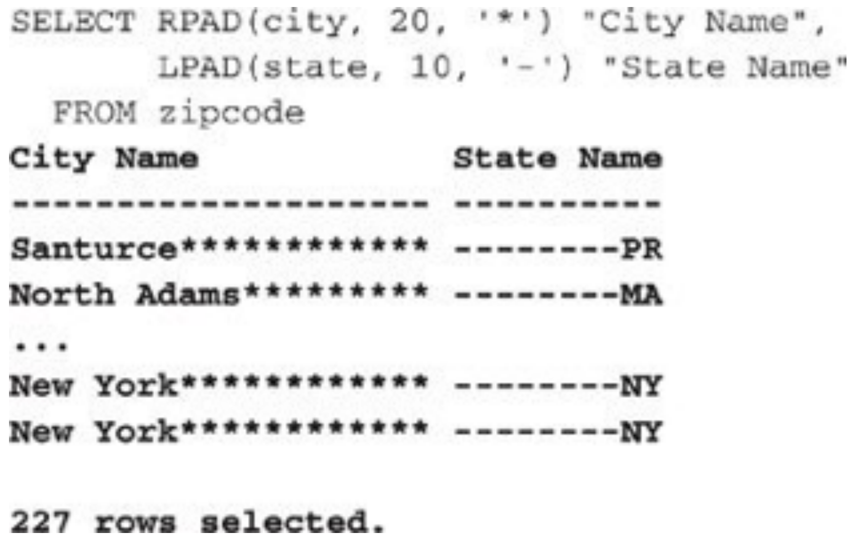
The syntax of the UPPER, LOWER, and INITCAP function is listed here.

```
UPPER(char)
LOWER(char)
INITCAP(char)
```

The LPAD and RPAD Functions

The LPAD and RPAD functions also transform data: They *left pad* and *right pad* strings, respectively. When you pad a string, you add to it. These functions can add characters, symbols, or even spaces to your result set. Unlike the LOWER, UPPER, or INITCAP functions, these functions take more than one parameter as their input.

This SELECT statement displays cities right padded with asterisks and states left padded with dashes.



```
SELECT RPAD(city, 20, '*') "City Name",
       LPAD(state, 10, '-') "State Name"
  FROM zipcode
City Name            State Name
-------------------- ----------
Santurce************ --------PR
North Adams********* --------MA
...
New York************ --------NY
New York************ --------NY

227 rows selected.
```

PRINTED BY: Monica Gonzalez <monicazalez1@hotmail.com>. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

The CITY column is right padded with the asterisk (*) character, up to a length of 20 characters. The STATE column is left padded with (-), up to a total length of 10 characters. Both the LPAD and RPAD functions use three parameters, separated by commas. The first input parameter accepts either a text literal or a column of data type VARCHAR2, CHAR, or CLOB. The second 136

137 argument specifies the total length to which the string should be padded. The third optional argument indicates the character(s) the string should be padded with. If this parameter is not specified, the string is padded with spaces by default.

The syntax for the LPAD and RPAD functions is as follows.

```
LPAD(char1, n [, char2])
RPAD(char1, n [, char2])
```

Shown as *char1*, is the string to perform the function on, *n* represents the length the string should be padded to, and *char2* is the optional parameter (denoted by the brackets) used to specify which character(s) to pad the string with. The next SELECT statement shows an example of the LPAD function with the third optional argument missing, thus left padding the column with spaces.

PRINTED BY: Monica Gonzalez <monicazalez1@hotmail.com>. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

```
SELECT LPAD(city, 20) AS "City Name"
  FROM zipcode
City Name
--------------------
            Santurce
         North Adams
...
            New York
            New York

227 rows selected.
```

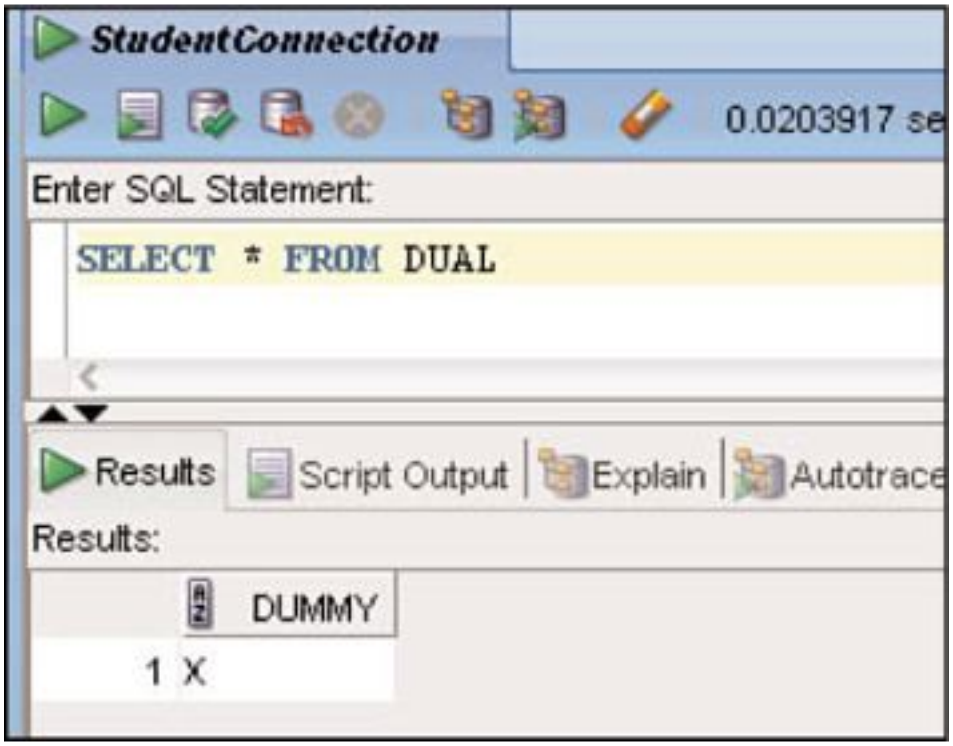
The DUAL Table

DUAL is a table unique to Oracle. It contains a single row and a single column called DUMMY (see Figure 4.1) and holds no significant data of its own. It can be used in conjunction with functions to select values that do not exist in tables, such as text literals or today’s date.

A single row is always returned in the result set. Some of the subsequent SQL examples are not concerned with specific rows but instead use literals to demonstrate the purpose of a function.

PRINTED BY: Monica Gonzalez <monicazalez1@hotmail.com>. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

FIGURE 4.1 Selecting from the DUAL table



```
SELECT * FROM DUAL
```

```
    DUMMY
1 X
```

You might wonder why the DUAL table does not appear in the list of tables below StudentConnection. This table does not belong to the STUDENT schema. It belongs to the user SYS, which is the owner of all Oracle system tables. This table, like many other Oracle system tables, is accessible to all the users via a public synonym. We will further discuss synonyms and these special SYS tables in Chapter 15, "Security."

If you want to see more about the DUAL table's definition, right-click the table and choose the Popup

PRINTED BY: Monica Gonzalez <monicazalez1@hotmail.com>. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

Describe option or in SQL*Plus, use the DESCRIBE command to display the structure of the table.

The LTRIM, RTRIM, and TRIM Functions

LTRIM and RTRIM are the opposite of LPAD and RPAD because they *trim*, or remove, unwanted characters, symbols, or spaces in strings. In this example, you see the use of the DUAL table to trim the zero (0) from the left, the right, and both sides. If both the left and right sides of the string are trimmed, you need to nest the function. The result of one function provides the input for the other function.

```
SELECT LTRIM('0001234500', '0') left,
       RTRIM('0001234500', '0') right,
       LTRIM(RTRIM('0001234500', '0'), '0') both
  FROM dual
LEFT    RIGHT    BOTH
------- -------- -----
1234500 00012345 12345

1 row selected.
```

Here is the syntax for the LTRIM and RTRIM functions. The optional parameter *char2* is used to specify which character(s) to trim from the string. If *char2* is not specified, the string is trimmed of spaces.

PRINTED BY: Monica Gonzalez <monicazalez1@hotmail.com>. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

```
LTRIM(char1 [, char2])
RTRIM(char1 [, char2])
```

The TRIM function removes leading characters, trailing characters, or both, effectively doing the job of LTRIM and RTRIM in one function. If you want the function to act like LTRIM, specify LEADING as the first parameter; for RTRIM, use the TRAILING option; for both, either specify the BOTH keyword or omit it altogether.

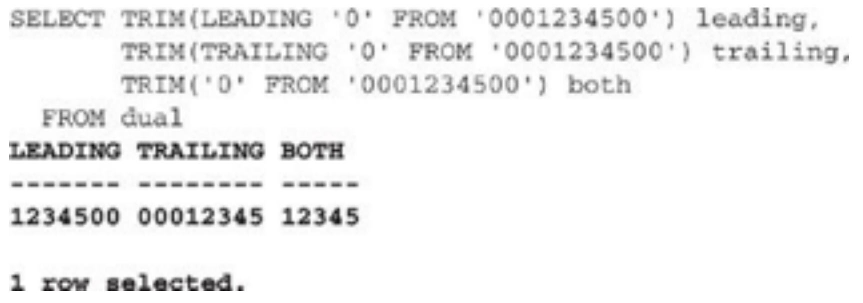
The syntax for TRIM is as follows. The parameter named *char1* indicates the *single* character to be removed; char2 is the string to be trimmed. If you don't specify *char1*, blank spaces are assumed.

```
TRIM([LEADING|TRAILING|BOTH] char1
FROM char2)
```

The next example shows the use of LEADING, TRAILING, and BOTH (if neither LEADING nor TRAILING is specified); the result is identical to the result of the previous query.

138

139



```
SELECT TRIM(LEADING '0' FROM '0001234500') leading,
       TRIM(TRAILING '0' FROM '0001234500') trailing,
       TRIM('0' FROM '0001234500') both
  FROM dual
LEADING TRAILING BOTH
------- -------- -----
1234500 00012345 12345

1 row selected.
```

PRINTED BY: Monica Gonzalez <monicazalez1@hotmail.com>. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

To trim blank spaces only, you can use the following syntax.

```
TRIM(char2)
```

Here is an example of a string with blank characters. Only leading and trailing blanks are trimmed, and blank spaces in the middle of the string are ignored.

```
SELECT TRIM('   00012345  00  ') AS "Blank Trim"
  FROM dual
Blank Trim
------------
00012345  00

1 row selected.
```

The SUBSTR Function

SUBSTR transforms a string, returning a *substring* or *subset* of a string, based on its input parameters. The following query displays student last names, the *first* five characters of those last names in the second column, and the *remaining* characters of those last names in the third column.

```
SELECT last_name,
       SUBSTR(last_name, 1, 5),
       SUBSTR(last_name, 6)
  FROM student
LAST_NAME                 SUBST SUBSTR(LAST_NAME,6)
------------------------- -------------------------
Eakheit                   Eakhe it
Millstein                 Mills tein
...
Mastandora                Masta ndora
Torres                    Torre s

268 rows selected.
```

PRINTED BY: Monica Gonzalez <monicazalez1@hotmail.com>. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

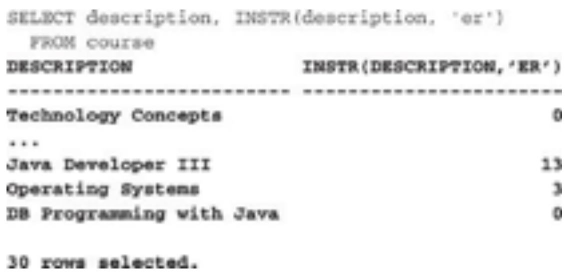
The SUBSTR function’s first input parameter is a string; the second is the starting position of the subset; the third is optional, indicating the length of the subset. If the third parameter is not used, the default is to display the remainder of the string. Here is the syntax for SUBSTR.

```
SUBSTR(char1, starting_position [,
substring_length])
```

If starting_position is a negative number, Oracle starts counting from the end of the string; you will see some examples of this in this chapter as part of the exercises.

The INSTR Function

INSTR, meaning *in string*, looks for the occurrence of a string inside another string, returning the starting position of the search string within the target string. Unlike the other string functions, INSTR does not return another string; rather, it returns a number. The following query displays course descriptions and the position in which the first occurrence of the string ‘er’, if any, in the DESCRIPTION column appears.



```
SELECT description, INSTR(description, 'er')
  FROM course
DESCRIPTION               INSTR(DESCRIPTION,'ER')
------------------------- -----------------------
Technology Concepts                             0
...
Java Developer III                             13
Operating Systems                               3
DB Programming with Java                        0

30 rows selected.
```

PRINTED BY: Monica Gonzalez <monicazalez1@hotmail.com>. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

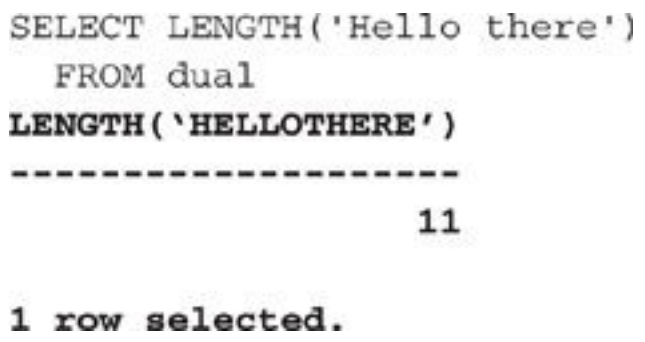
As you can see in the second-to-last row of the result set, the string 'er' starts in the third position of Operating Systems. The last row, DB Programming with Java, does not contain an 'er' string, and the result is therefore 0. The syntax for INSTR is as follows.

```
INSTR(char1, char2
[,starting_position [,
occurrence]])
```

INSTR can take two optional input parameters. The third parameter allows you to specify the start position for the search. The fourth parameter specifies which occurrence of the string to look for. When these optional parameters are not used, the default value is 1.

The LENGTH Function

The LENGTH function determines the length of a string, expressed as a number. The following SQL statement selects a text literal from the DUAL table in conjunction with the LENGTH function.



```
SELECT LENGTH('Hello there')
  FROM dual
LENGTH('HELLOTHERE')
--------------------
                  11

1 row selected.
```

PRINTED BY: Monica Gonzalez <monicazalez1@hotmail.com>. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

Functions in WHERE and ORDER BY Clauses

The use of functions is not restricted to the SELECT list; they are also used in other SQL clauses. In a WHERE clause, a function restricts the output to rows that only evaluate to the result of the function. In an ORDER BY clause, rows are sorted based on the result of a function. The next query uses the SUBSTR function in the WHERE clause to search for student last names that begin with the string ‘Mo’. The arguments are the LAST_NAME column of the STUDENT table, starting with the first character of the column, for a length of two characters.

```
SELECT first_name, last_name
  FROM student
 WHERE SUBSTR(last_name, 1, 2) = 'Mo'
FIRST_NAME                LAST_NAME
------------------------- ---------
Edgar                     Moffat
Angel                     Moskowitz
Vinnie                    Moon
Bernadette                Montanez

4 rows selected.
```

PRINTED BY: Monica Gonzalez <monicazalez1@hotmail.com>. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

Alternatively, you can achieve the same result by replacing the SUBSTR function with the following WHERE clause.

```
WHERE last_name LIKE 'Mo%'
```

The following SQL statement selects student first and last names, where the value in the FIRST_NAME column contains a period, and also orders the result set based on the length of students' last names.

```
SELECT first_name, last_name
  FROM student
 WHERE INSTR(first_name, '.') > 0
 ORDER BY LENGTH(last_name)
FIRST_NAME                LAST_NAME
------------------------- ---------
Suzanne M.                Abid
D.                        Orent
...
V.                        Saliternan
Z.A.                      Scrittorale

21 rows selected.
```

Nested Functions

As you saw earlier, in the example of LPAD and RPAD, functions can be nested within each other. Nested functions are evaluated starting from the inner function and working outward.

PRINTED BY: Monica Gonzalez <monicazalez1@hotmail.com>. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

The following example shows the city column formatted in uppercase and right padded with periods.

```
SELECT RPAD(UPPER(city), 20,'.')
  FROM zipcode
 WHERE state = 'CT'
RPAD(UPPER(CITY),20,
--------------------
ANSONIA.............
MIDDLEFIELD.........
...
STAMFORD............
STAMFORD............

19 rows selected.
```

Here is a more complicated but useful example. You may have noticed that middle initials in the STUDENT table are entered in the same column as the first name. To separate the middle initial from the first name, nest the SUBSTR and INSTR functions. First, determine the position of the middle initial’s period in the FIRST_NAME column with the INSTR function. From this position, deduct the number 1. This brings you to the position before the period, where the middle initial starts, which is where you want the SUBSTR function to start. The WHERE clause selects only rows where the third or any subsequent character of the first name contains a period.

PRINTED BY: Monica Gonzalez <monicazalez1@hotmail.com>. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

```
SELECT first_name,
       SUBSTR(first_name, INSTR(first_name, '.')-1) mi,
       SUBSTR(first_name, 1, INSTR(first_name, '.')-2) first
  FROM student
 WHERE INSTR(first_name, '.') >= 3
FIRST_NAME                MI   FIRST
------------------------- ---- ------
Austin V.                 V.   Austin
John T.                   T.   John
...
Suzanne M.                M.   Suzanne
Rafael A.                 A.   Rafael

7 rows selected.
```

For example, in the row for Austin V., the position of the period is 9, but you need to start at 8 to include the middle initial letter. The last column of the result lists the first name without the middle initial. This is accomplished by starting with the first character of the string and ending the string before the position where the middle initial starts. The key is to determine the ending position of the string with the INSTR function and count back two characters.

When using nested functions, a common pitfall is to misplace matching parentheses or forget the second half of the pair altogether. Start by writing a nested function

PRINTED BY: Monica Gonzalez <monicazalez1@hotmail.com>. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

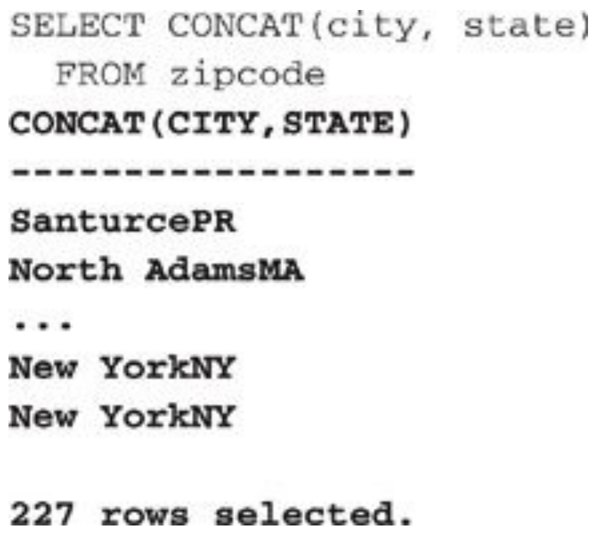
from the inside out. Count the number of left parentheses and make sure it matches the number of right parentheses.

Concatenation

Concatenation connects strings *together* to become one. Strings can be concatenated to produce a single column in the result set. There are two methods of concatenation in Oracle: One is with the CONCAT function, the other is the concatenation operator (||), which is two *vertical bars* or *pipe symbols*. The syntax of the CONCAT function is as follows.

```
CONCAT(char1, char2)
```

When you want to concatenate cities and states together using the CONCAT function, you can use the function as follows.



```
SELECT CONCAT(city, state)
  FROM zipcode
CONCAT(CITY,STATE)
------------------
SanturcePR
North AdamsMA
...
New YorkNY
New YorkNY

227 rows selected.
```

PRINTED BY: Monica Gonzalez <monicazalez1@hotmail.com>. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

The result set is difficult to read without spaces between cities and states. The CONCAT function takes only two parameters. By using the || operator, you can easily concatenate several strings.

```
SELECT city||state||zip
  FROM zipcode
CITY||STATE||ZIP
-----------------
SanturcePR00914
North AdamsMA01247
...
New YorkNY10005
New YorkNY10035

227 rows selected.
```

For a result set that is easier to read, concatenate the strings with spaces and separate the CITY and STATE columns with a comma.

```
CITY||','||STATE||''||ZIP
-----------------------
Santurce, PR  00914
North Adams, MA  01247
...
New York, NY  10005
New York, NY  10035

227 rows selected.
```

PRINTED BY: Monica Gonzalez <monicazalez1@hotmail.com>. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

The REPLACE Function

The REPLACE function *replaces* one string with another string. In the following example, when the string ‘hand’ is found within the string ‘My hand is asleep’, it is replaced by the string ‘foot’.

```
SELECT REPLACE('My hand is asleep', 'hand', 'foot')
  FROM dual
REPLACE('MYHANDISA
------------------
My foot is asleep

1 row selected.
```

The following is the syntax for the REPLACE function.

```
REPLACE(char, if, then)
```

The second parameter looks to see if a string exists within the first parameter. If so, it displays the third parameter. If the second parameter is not found, then the original string is displayed.

```
SELECT REPLACE('My hand is asleep', 'x', 'foot')
  FROM dual
REPLACE('MYHANDISA
------------------
My hand is asleep

1 row selected.
```

PRINTED BY: Monica Gonzalez <monicazalez1@hotmail.com>. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

The TRANSLATE Function

Unlike REPLACE, which replaces an entire string, the TRANSLATE function provides a one-for-one character substitution. For instance, it allows you to determine whether all the phone numbers in the STUDENT table follow the same format. In the next query, TRANSLATE substitutes the # character for every character from 0 to 9. Then the values are checked against the '###-###-####' format.

```
SELECT phone
  FROM student
 WHERE TRANSLATE(
  phone, '0123456789',
   '##########') <> '###-###-####'
no rows selected
```

If any phone number is entered in an invalid format, such as 'abc-ddd-efgh' or '555-1212', the query returns the row(s) with the incorrect phone format. The following is the syntax for the TRANSLATE function.

```
TRANSLATE(char, if, then)
```

PRINTED BY: Monica Gonzalez <monicazalez1@hotmail.com>. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

The SOUNDEX Function

The SOUNDEX function allows you to compare differently spelled words that phonetically sound alike. The next query uses the SOUNDEX function to display students whose last name sounds like Martin.

```
SELECT student_id, last_name
  FROM student
 WHERE SOUNDEX(last_name) = SOUNDEX('MARTIN')
STUDENT_ID LAST_NAME
---------- ---------
       110 Martin
       324 Marten
       393 Martin

3 rows selected.
```

Which Character Function Should You Use?

It's easy to confuse character functions. When deciding which one to use, ask yourself exactly what is needed in your result set. Are you looking for the position of a string in a string? Do you need to produce a subset of a string? Do you need to know how long a string is? Or do you need to replace a string with something else? Table 4.2 lists the character functions discussed in this lab.

PRINTED BY: Monica Gonzalez <monicazalez1@hotmail.com>. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

TABLE 4.2 Character Functions

FUNCTION	**PURPOSE**
LOWER(*char*)	Converts to lowercase
UPPER(*char*)	Converts to uppercase
INITCAP(*char*)	Capitalizes the first letter
LPAD(*char1*, *n*, [*char2*])	Left pads
RPAD(*char1*, *n*, [*char2*])	Right pads
LTRIM(*char1*, [*char2*])	Left trims
RTRIM(*char1*, [*char2*])	Right trims
TRIM([LEADING|TRAILING|BOTH]*char1* FROM *char2*)	Trims leading, trailing or both sides
SUBSTR(*char1, starting_position, [substring_length*])	Cuts out a piece of a string
INSTR(*char1*, *char2*, [*starting_position*, [*occurrence*]])	Determines the starting location of a string
LENGTH(*char*)	Returns the length of a string
CONCAT(*char1*, *char2*)	Concatenates two strings
REPLACE(*char*, *if*, *then*)	Replaces a string with another string

PRINTED BY: Monica Gonzalez <monicazalez1@hotmail.com>. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

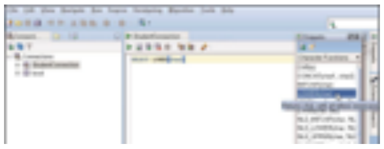
SOUNDEX(*char*)	Returns phonetic representation
TRANSLATE(*char*, *if*, *then*)	Substitutes individual character

SQL Developer Snippets

SQL Developer introduced a feature called *snippets*, which are SQL functions or syntax examples. You can even create your own snippets of frequently used statements. Snippets are helpful in both SQL statements and PL/SQL programming. You find the Snippets window tab located on the right side of the Enter SQL Statement box (see Figure 4.2). You can also invoke it by choosing View, Snippets.

Figure 4.2 shows an example of the LOWER function snippet within a SQL statement. To insert a snippet, drag it into the Enter SQL Statement box. Now you can replace in the desired column or literal shown as the char parameter and complete the rest of the SQL statement. Hovering over the function reveals a brief description of the function.

FIGURE 4.2 Inserting a snippet



PRINTED BY: Monica Gonzalez <monicazalez1@hotmail.com>. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

Snippets are useful for storing frequently used SQL statements.

Searching, Replacing, and Validating Text

In addition to the LIKE operator and the character functions SUBSTR and INSTR, Oracle offers additional search capabilities that come in the form of Oracle Text and regular expressions.

Oracle Text expands the text search capabilities with word and theme searching, using the operators CONTAINS, CATSEARCH, and MATCHES. The database returns ranked results, based on the requested search. You can specify combinations of words with the AND and OR operators, and you can use wildcards. You can search for documents that contain words that share the same stem, words that are located close to each other, or words that revolve around the same theme. The *Oracle Text Reference* manual and the *Oracle Text Application Developer's Guide* contain more information on this functionality.

PRINTED BY: Monica Gonzalez <monicazalez1@hotmail.com>. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

Regular expression functions are part Oracle's SQL language. A *regular expression* is a notation for describing a pattern. Regular expressions significantly expand the functionality of the LIKE operator and the INSTR, SUBSTR, and REPLACE functions to search, replace, and validate patterns. You will learn more about the sophisticated capabilities of regular expressions within the context of the Oracle database in Chapter 16, "Regular Expressions and Hierarchical Queries."

LAB 4.1 EXERCISES

a) Execute the following SQL statement. Based on the result, what is the purpose of the INITCAP function?

```
SELECT description "Description",
   INITCAP(description) "Initcap
Description"
  FROM course
WHERE description LIKE '%SQL%'
```

b) What question does the following SQL statement answer?

```
SELECT last_name
   FROM instructor
  WHERE LENGTH(last_name) >= 6
```

PRINTED BY: Monica Gonzalez <monicazalez1@hotmail.com>. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

c) Describe the result of the following SQL statement. Pay particular attention to the negative number parameter.

```
SELECT SUBSTR('12345', 3),
   SUBSTR('12345', 3, 2),
  SUBSTR('12345', -4, 3)
FROM dual
```

d) Based on the result of the following SQL statement, describe the purpose of the LTRIM and RTRIM functions.

```
SELECT zip, LTRIM(zip, '0'),
RTRIM(ZIP, '4')
  FROM zipcode
 ORDER BY zip
```

e) What do you observe when you execute the next statement? How should the statement be changed to achieve the desired result?

```
SELECT TRIM('01' FROM
'0123014560l')
 FROM dual
```

f) What is the result of the following statement?

```
SELECT TRANSLATE('555-1212',
'0123456789',
```

PRINTED BY: Monica Gonzalez <monicazalez1@hotmail.com>. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

```
'##########')
FROM dual
```

g) Write a SQL statement to retrieve students who have a last name with the lowercase letter o occurring three or more times.

h) The following statement determines how many times the string 'ed' occurs in the phrase 'Fred fed Ted bread, and Ted fed Fred bread'. Explain how this is accomplished.

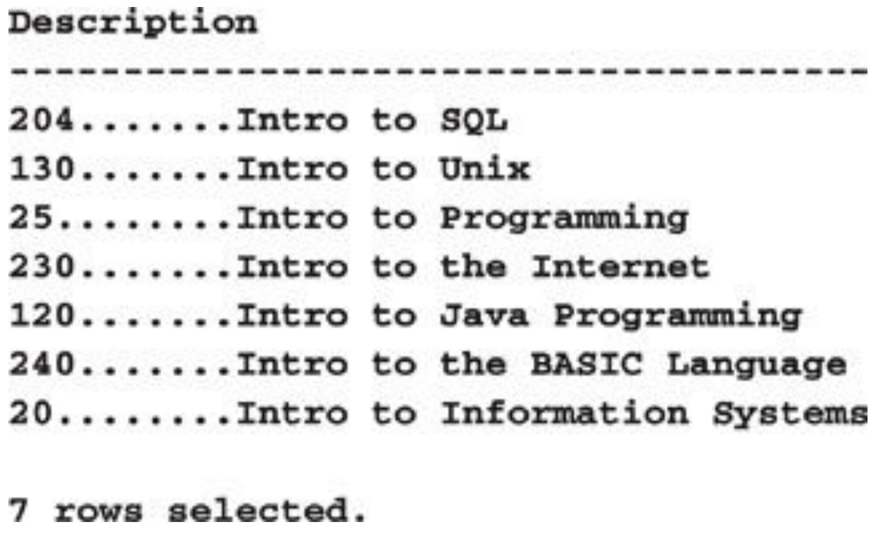
```
SELECT (
  LENGTH('Fred fed Ted bread, and Ted fed Fred bread.') -
  LENGTH(REPLACE(
         'Fred fed Ted bread, and Ted fed Fred bread.', 'ed', NULL))
            ) /2 AS occurr
  FROM dual
    OCCURR
----------
         6

1 row selected.
```

i) Write a SELECT statement that returns each instructor’s last name, followed by a comma and a space, followed by the instructor’s first name, all in a single column in the result set.

j) Using functions in the SELECT list and WHERE and ORDER BY clauses, write a SELECT statement that returns course numbers and course descriptions from the COURSE table and looks like the following result set. Use the SQL

PRINTED BY: Monica Gonzalez <monicazalez1@hotmail.com>. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

Developer Run Script icon to display the result in fixed-width format.



```
Description
----------------------------------------
204.......Intro to SQL
130.......Intro to Unix
25........Intro to Programming
230.......Intro to the Internet
120.......Intro to Java Programming
240.......Intro to the BASIC Language
20........Intro to Information Systems

7 rows selected.
```

Lab 4.1 Exercise Answers

a) Execute the following SQL statement. Based on the result, what is the purpose of the INITCAP function?

```
SELECT description ”Description”,
   INITCAP(description) ”Initcap
Description”
  FROM course
 WHERE description LIKE '%SQL%'
```

ANSWER: The INITCAP function capitalizes the first letter of a word and forces the remaining characters to be lowercase.

PRINTED BY: Monica Gonzalez <monicazalez1@hotmail.com>. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

The result set contains two rows, one displaying a course description as it appears in the database and one displaying each word with only the first letter capitalized. Words are delimited by nonalphanumeric characters or spaces.

```
Description               Initcap Description
------------------------- ------------------
Intro to SQL              Intro To Sql
PL/SQL Programming        Pl/Sql Programming

2 rows selected.
```

b) What question does the following SQL statement answer?

```
SELECT last_name
   FROM instructor
  WHERE LENGTH(last_name) >= 6
```

ANSWER: The question answered by the query could be phrased like this: “Which instructors have last names equal to six characters or more?”

```
LAST_NAME
---------------
Wojick
Schorin
...
Frantzen
Willig

7 rows selected.
```

PRINTED BY: Monica Gonzalez <monicazalez1@hotmail.com>. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

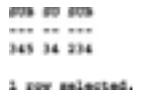
The LENGTH function returns the length of a string, expressed as a number. The LENGTH function takes only a single input parameter, as in the following syntax.

```
LENGTH(char)
```

c) Describe the result of the following SQL statement. Pay particular attention to the negative number parameter.

```
SELECT SUBSTR('12345', 3),
   SUBSTR('12345', 3, 2),
  SUBSTR('12345', -4, 3)
 FROM dual
```

ANSWER: The first column takes the characters starting from the third position until the end, resulting in the string '345'. The second SUBSTR function also starts at the third position but ends after two characters and therefore returns '34'. The third column has a negative number as the first parameter. It counts from the end of the string to the left four characters; thus the substring starts at position 2 and for a length of three characters, resulting in '234'.



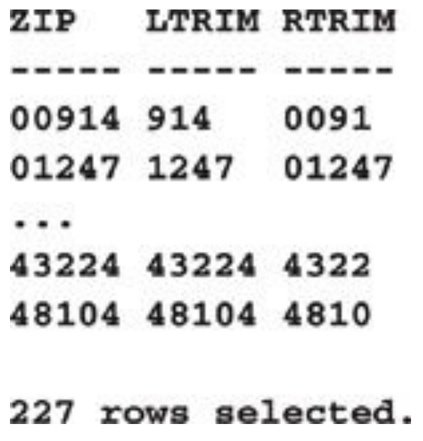
```
SUB SU SUB
--- -- ---
345 34 234

1 row selected.
```

PRINTED BY: Monica Gonzalez <monicazalez1@hotmail.com>. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

d) Based on the result of the following SQL statement, describe the purpose of the LTRIM and RTRIM functions.

```
SELECT zip, LTRIM(zip, '0'),
RTRIM(ZIP, '4')
  FROM zipcode
 ORDER BY zip
```

ANSWER: The LTRIM and RTRIM functions left trim and right trim strings, based on the function's parameters. With the three columns in the result set side by side, you see the differences: The first column shows the ZIP column without modification, the second with ZIP left-trimmed of its 0s, and the third with ZIP right-trimmed of its 4s.



```
ZIP   LTRIM RTRIM
----- ----- -----
00914 914   0091
01247 1247  01247
...
43224 43224 4322
48104 48104 4810

227 rows selected.
```

PRINTED BY: Monica Gonzalez <monicazalez1@hotmail.com>. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

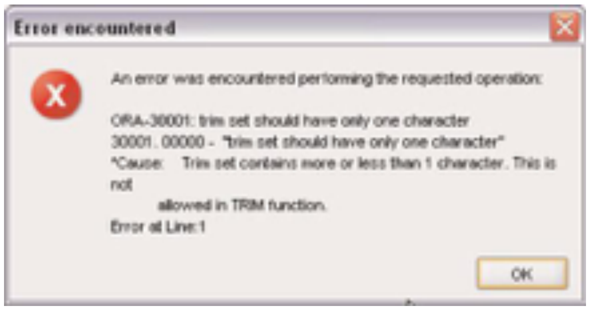
e) What do you observe when you execute the next statement? How should the statement be changed to achieve the desired result?

```
SELECT TRIM('01' FROM
'01230145601')
  FROM dual
```

ANSWER: The query results in an error (see Figure 4.3), indicating that only one character can be trimmed at a time. This query attempts to trim two characters, which are 0 and 1. Nest the LTRIM and RTRIM functions to achieve the desired result.

```
SELECT TRIM('01' FROM
'01230145601')
FROM dual
```

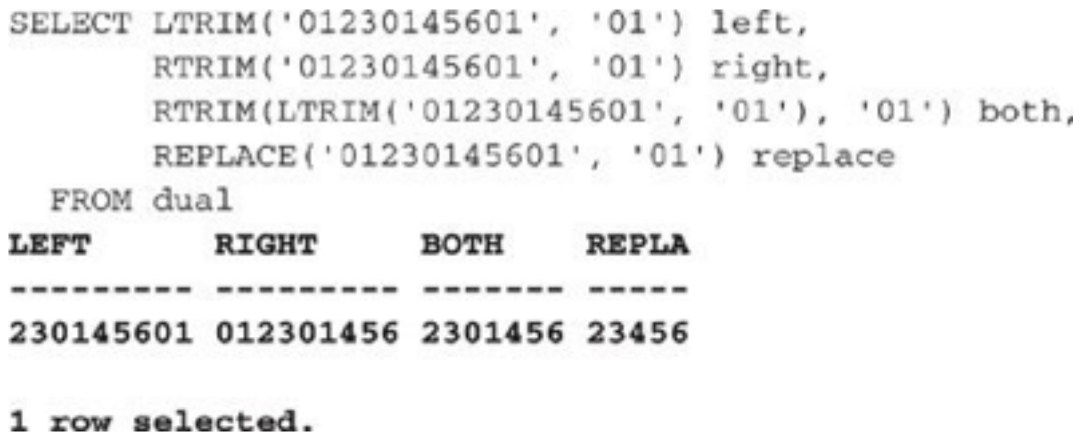
FIGURE 4.3 Error message on an invalid TRIM parameter



```
Error encountered

An error was encountered performing the requested operation:

ORA-30001: trim set should have only one character
30001. 00000 -  "trim set should have only one character"
*Cause:    Trim set contains more or less than 1 character. This is
not
           allowed in TRIM function.
Error at Line:1
```

PRINTED BY: Monica Gonzalez <monicazalez1@hotmail.com>. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

To trim multiple characters, use the LTRIM and RTRIM functions instead of TRIM. If you try the REPLACE function, it replaces all occurrences of the ‘01’ string, not just the first and last.



```
SELECT LTRIM('01230145601', '01') left,
       RTRIM('01230145601', '01') right,
       RTRIM(LTRIM('01230145601', '01'), '01') both,
       REPLACE('01230145601', '01') replace
  FROM dual
LEFT      RIGHT     BOTH    REPLA
--------- --------- ------- -----
230145601 012301456 2301456 23456

1 row selected.
```

f) What is the result of the following statement?

```
SELECT TRANSLATE('555-1212',
'0123456789',
      '##########')
 FROM dual
```

ANSWER: It returns the result ###-####. The TRANSLATE function is a character substitution function. The listed SQL statement uses each of the characters in the string '555-1212' to look up the corresponding character and then returns this character. One of the uses for this function is to determine whether data is entered in the correct format.

PRINTED BY: Monica Gonzalez <monicazalez1@hotmail.com>. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

```
TRANSLAT
--------
###-####

1 row selected.
```

USING TRANSLATE FOR PATTERN SEARCHING

The TRANSLATE function also comes in handy when you need to perform a pattern search using the LIKE operator and you are looking for the actual wildcard characters % or _. Assume that you need to query the STUDENT table, and you want to find any students whose employer spells his or her name similar to the pattern B_B. The underscore has to be taken as a literal underscore, not as a wildcard. Qualifying 151 152 Character Functions employer names are Bayer B_Biller and ABCB_Bellman. Unfortunately, no such employer names exist in the STUDENT database, but there are occasions when data entry errors occur and you need to figure out which are the offending rows. The following query checks whether an employer with the pattern B_B exists in the table.

```
SELECT student_id, employer
   FROM student
```

PRINTED BY: Monica Gonzalez <monicazalez1@hotmail.com>. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

```
    WHERE TRANSLATE(employer, '_',
  '+') LIKE '%B+B%'
```

As you can see, the TRANSLATE function performs this trick. Here the underscore is replaced with the plus sign and then the LIKE function is applied with the replaced plus sign in the character literal.

USING INSTR FOR PATTERN SEARCHING

Another way to solve the problem in the preceding section would be to use the previously discussed INSTR function, which returns the starting position of the string.

```
SELECT student_id, employer
   FROM student
  WHERE INSTR(employer, 'B_B') > 0
```

THE ESCAPE CHARACTER AND THE LIKE OPERATOR

Yet another way to solve the problem is with the escape character functionality in conjunction with the LIKE operator. In the next example, the backslash (\) sign is selected as the escape character to indicate that the underscore character following the character is to be

PRINTED BY: Monica Gonzalez <monicazalez1@hotmail.com>. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

interpreted as a literal underscore and not as the wildcard underscore.

```
SELECT student_id, employer
   FROM student
 WHERE employer LIKE '%B\_B%'
ESCAPE '\'
```

The regular expressions functionality allows sophisticated pattern searches with the REGEXP_LIKE operator. You will learn more about these capabilities in Chapter 16.

g) Write a SQL statement to retrieve students who have a last name with the lowercase letter o occurring three or more times.

ANSWER: The INSTR function determines the third or more occurrence of the lowercase letter o in the LAST_NAME column of the STUDENT table.

The INSTR function has two required parameters; the rest are optional and default to 1. The first parameter is the string or column where the function needs to be applied and where you are looking to find the desired values. The second parameter identifies the search string; here you are looking for the letter o. The third parameter determines at which starting position the search

PRINTED BY: Monica Gonzalez <monicazalez1@hotmail.com>. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

must occur. The last parameter specifies which occurrence of the string is requested.

If the INSTR function finds the desired result, it returns the starting position of the searched value. The WHERE clause condition looks for those rows where the result of the INSTR function is greater than 0.

```
SELECT student_id, last_name
  FROM student
 WHERE INSTR(last_name, 'o', 1, 3) > 0
```

```
STUDENT_ID LAST_NAME
---------- ---------
       280 Engongoro
       251 Frangopoulos
       254 Chamnonkool

3 rows selected.
```

h) The following statement determines how many times the string 'ed' occurs in the phrase 'Fred fed Ted bread, and Ted fed Fred bread'. Explain how this is accomplished.

```
SELECT (
  LENGTH('Fred fed Ted bread, and Ted fed Fred bread') -
  LENGTH(REPLACE(
         'Fred fed Ted bread, and Ted fed Fred bread', 'ed', NULL))
          ) /2 AS occurr
  FROM dual
   OCCURR
----------
         6

1 row selected.
```

PRINTED BY: Monica Gonzalez <monicazalez1@hotmail.com>. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

ANSWER: The nesting of the REPLACE and LENGTH functions determines that there are six occurrences of the string 'ed' in the phrase.

To understand the statement, it's best to break down the individual components of the statement: The first function determines the length of this tongue twister. The next component nests the LENGTH and REPLACE functions. The REPLACE function replaces every occurrence of 'ed' with a null. Effectively, the result string looks as follows.

```
SELECT REPLACE
  ('Fred fed Ted bread, and Ted fed Fred bread',
   'ed', NULL)
  FROM dual
REPLACE('FREDFEDTEDBREAD,ANDTE
------------------------------
Fr f T bread, and T f Fr bread

1 row selected.
```

Then the LENGTH function determines the length of the reduced string. If you deduct the total length of the entire string from the length of the reduced string and divide the result by 2 (the number of letters in the string 'ed'), you determine the total number of occurrences.

PRINTED BY: Monica Gonzalez <monicazalez1@hotmail.com>. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

Starting with Oracle 11*g*, you can now use the REGEXP_COUNT function to perform the same functionality described in Exercise h. This function is discussed in Chapter 16 as part of the Regular Expression functionality. 153

154

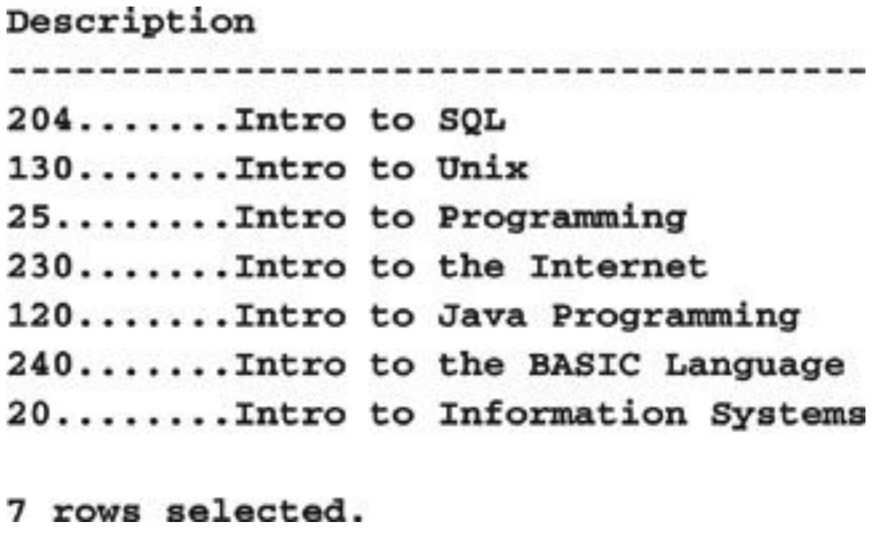
i) Write a SELECT statement that returns each instructor's last name, followed by a comma and a space, followed by the instructor's first name, all in a single column in the result set.

ANSWER: The instructor's last name, a comma and a space, and the instructor's first name are all concatenated using the || symbol.

```
SELECT last_name||', '||first_name
  FROM instructor
LAST_NAME||','||FIRST_NAME
-------------------------
Hanks, Fernand
Wojick, Tom
...
Frantzen, Marilyn
Willig, Irene

10 rows selected.
```

PRINTED BY: Monica Gonzalez <monicazalez1@hotmail.com>. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

j) Using functions in the SELECT list and WHERE and ORDER BY clauses, write a SELECT statement that returns course numbers and course descriptions from the COURSE table and looks like the following result set. Use the SQL Developer Run Script icon to display the result in fixed-width format.



```
Description
----------------------------------------
204.......Intro to SQL
130.......Intro to Unix
25........Intro to Programming
230.......Intro to the Internet
120.......Intro to Java Programming
240.......Intro to the BASIC Language
20........Intro to Information Systems

7 rows selected.
```

ANSWER: The RPAD function right pads the COURSE_NO column with periods, up to 10 characters long; it is then concatenated with the DESCRIPTION column. The INSTR function is used in the WHERE clause to filter on descriptions, with the string ‘Intro’. The LENGTH function in the ORDER BY clause sorts the result set by ascending (shortest to longest) description length.

PRINTED BY: Monica Gonzalez <monicazalez1@hotmail.com>. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

```
SELECT RPAD(course_no, 10, '.')||
description
   AS "Description"
  FROM course
 WHERE INSTR(description, 'Intro')
= 1
 ORDER BY LENGTH(description)
```

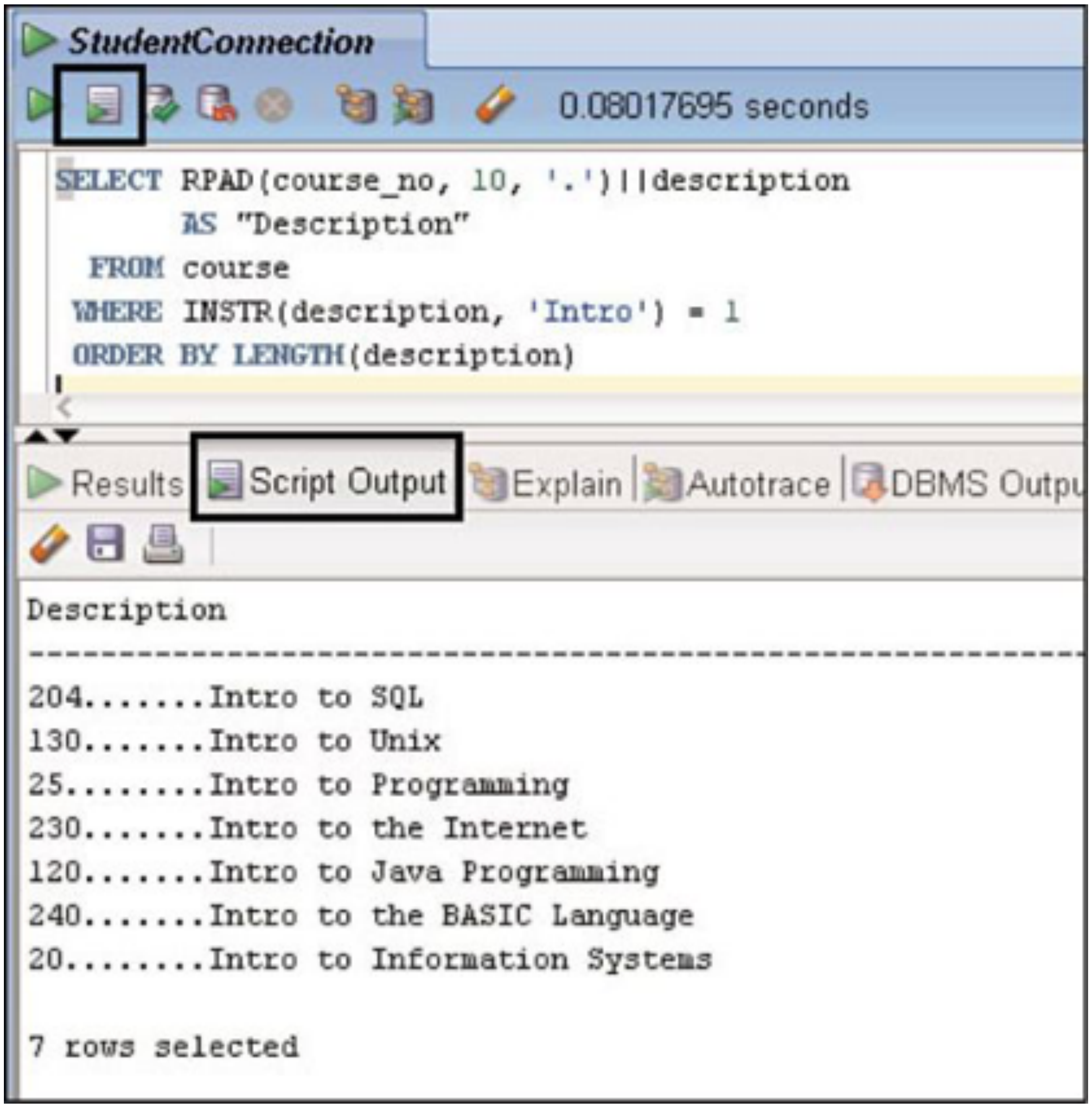
The same result can be obtained without the use of the INSTR function, as in the following WHERE clause.

```
WHERE description LIKE 'Intro%'
```

Figure 4.4 shows the statement executed within SQL Developer, using the Run Script icon, and the resulting output is shown in fixed-width format in the Script Output tab.

PRINTED BY: Monica Gonzalez <monicazalez1@hotmail.com>. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

FIGURE 4.4 Script Output tab result when using SQL Developer's Run Script icon



```
SELECT RPAD(course_no, 10, '.')||description
       AS "Description"
  FROM course
 WHERE INSTR(description, 'Intro') = 1
 ORDER BY LENGTH(description)
```

```
Description
---------------------------------------------------------------
204.......Intro to SQL
130.......Intro to Unix
25........Intro to Programming
230.......Intro to the Internet
120.......Intro to Java Programming
240.......Intro to the BASIC Language
20........Intro to Information Systems

7 rows selected
```

PRINTED BY: Monica Gonzalez <monicazalez1@hotmail.com>. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

Lab 4.1 Quiz

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

1) A function that operates on a single value can have only one input parameter.

   ______**a)** True

   ______**b)** False

2) The DUAL table can be used for testing functions.

   ______**a)** True

   ______**b)** False

3) The same function can be used twice in a SELECT statement.

   ______**a)** True

   ______**b)** False

4) The following SELECT statement contains an error.

```
SELECT UPPER(description)
  FROM LOWER(course)
```

   ______**a)** True

PRINTED BY: Monica Gonzalez <monicazalez1@hotmail.com>. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

______**b)** False

5) The RTRIM function is useful for eliminating extra spaces in a string.

______**a)** True

______**b)** False

6) Which one of the following string functions tells you how many characters are in a string?

______**a)** INSTR

______**b)** SUBSTR

______**c)** LENGTH

______**d)** REPLACE

7) Which result will the following query return? Note there are extra spaces between Mary and Jones.

```
SELECT TRIM(' Mary Jones ')
  FROM dual
```

______**a)** Mary Jones

______**b)** Mary Jones

______**c)** MaryJones

______**d)** The query returns an error.

PRINTED BY: Monica Gonzalez <monicazalez1@hotmail.com>. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

8) The functions INSTR, SUBSTR, and TRIM are all single-row functions.

______**a)** True

______**b)** False

9) Which character function returns a specified portion of a character string?

______**a)** INSTR

______**b)** LENGTH

______**c)** SUBSTR

______**d)** INSTRING

10) Character functions never return results in the data type NUMBER.

______**a)** True

______**b)** False

ANSWERS APPEAR IN APPENDIX A.

LAB 4.2 Number Functions

LAB OBJECTIVES

After this lab, you will be able to:

PRINTED BY: Monica Gonzalez <monicazalez1@hotmail.com>. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

- Use Number Functions
- Perform Mathematical Computations

Number functions are valuable tools for operations such as rounding numbers or computing the absolute value of a number. There are several single-row number functions in Oracle; the most useful ones are discussed here.

The ABS Function

The ABS function computes the *absolute value* of a number, measuring its magnitude.

```
SELECT -14, SIGN(-14), SIGN(14), SIGN(0), ABS(-14)
  FROM dual
      -14 SIGN(-14) SIGN(14)    SIGN(0)  ABS(-14)
--------- --------- ----------- -------- --------
      -14        -1           1        0       14

1 row selected.
```

ABS takes only a single input parameter, and its syntax is as follows.

```
ABS(value)
```

The SIGN Function

The SIGN function tells you the *sign* of a value, returning a number 1 for a positive number, −1 for a

PRINTED BY: Monica Gonzalez <monicazalez1@hotmail.com>. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

negative number, or 0 for zero. The following example compares SIGN with the ABS function.

```
SELECT -14, SIGN(-14), SIGN(14), SIGN(0), ABS(-14)
  FROM dual
      -14 SIGN(-14) SIGN(14)    SIGN(0)  ABS(-14)
--------- --------- ----------- -------- --------
      -14        -1           1        0       14

1 row selected.
```

158

159

SIGN also takes only a single input parameter, and its syntax is as follows.

```
SIGN(value)
```

Most single-row functions return NULL when a NULL is the input parameter.

ROUND and TRUNC Functions

ROUND and TRUNC are two useful functions that *round* and *truncate* (or cut off) values, respectively, based on a given number of digits of precision. The next SELECT statement illustrates the use of ROUND and TRUNC, which both take two input parameters. Observe the differences in the result.

PRINTED BY: Monica Gonzalez <monicazalez1@hotmail.com>. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

```
SELECT 222.34501,
       ROUND(222.34501, -2),
       TRUNC(222.34501, -2)
  FROM dual
222.34501 ROUND(222.34501,-2) TRUNC(222.34501,-2)
--------- ------------------- -------------------
222.34501                 200                 200

1 row selected.
```

Here, ROUND (222.34501, 2) rounds the number 222.34501 to two digits to the right of the decimal, rounding the result up to 222.35, following the normal convention for rounding. In contrast, TRUNC cuts off all digits beyond two digits to the right of the decimal, resulting in 222.34. ROUND and TRUNC can be used to affect the left side of the decimal as well by passing a negative number as a parameter.

```
SELECT 222.34501,
       ROUND(222.34501, -2),
       TRUNC(222.34501, -2)
  FROM dual
222.34501 ROUND(222.34501,-2) TRUNC(222.34501,-2)
--------- ------------------- -------------------
222.34501                 200                 200

1 row selected.
```

The following is the syntax for ROUND and TRUNC.

```
ROUND(value [, precision])
```

PRINTED BY: Monica Gonzalez <monicazalez1@hotmail.com>. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

```
TRUNC(value [, precision])
```

Numbers with decimal places may be rounded to whole numbers by omitting the second parameter, or specifying a precision of 0.

```
SELECT 2.617, ROUND(2.617), TRUNC(2.617)
  FROM dual
2.617 ROUND(2.617) TRUNC(2.617)
----- ------------ ------------
2.617            3            2

1 row selected.
```

You can use the TRUNC and ROUND functions not only on values of the NUMBER data type but also on the DATE data type, discussed in Chapter 5.

The FLOOR and CEIL Functions

The CEIL function returns the smallest integer greater than or equal to a value; the FLOOR function returns the largest integer equal to or less than a value. These functions perform much like the ROUND and TRUNC functions, without the optional precision parameter.

```
SELECT FLOOR(22.5), CEIL(22.5), TRUNC(22.5), ROUND(22.5)
  FROM dual
FLOOR(22.5) CEIL(22.5) TRUNC(22.5) ROUND(22.5)
----------- ---------- ----------- -----------
         22         23          22          23

1 row selected.
```

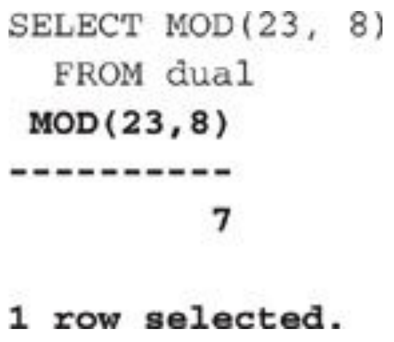
PRINTED BY: Monica Gonzalez <monicazalez1@hotmail.com>. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

The syntax for the FLOOR and CEIL functions is as follows.

```
FLOOR(value)
CEIL(value)
```

The MOD Function

MOD is a function that returns the *modulus*, or the remainder of a value divided by another value. It takes two input parameters, as in the following SELECT statement.



```
SELECT MOD(23, 8)
  FROM dual
 MOD(23,8)
----------
         7

1 row selected.
```

The MOD function divides 23 by 8 and returns a remainder of 7. The following is the syntax for MOD.

```
MOD(value, divisor)
```

The MOD function is particularly useful if you want to determine whether a value is odd or even. If you divide by 2 and the remainder is a zero, this indicates that the value is even; if the remainder is 1, it means that the value is odd.

PRINTED BY: Monica Gonzalez <monicazalez1@hotmail.com>. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

Floating-Point Numbers

Oracle's floating-point numbers support the IEEE standard for binary floating-point arithmetic, just like Java and XML. Computations on floating-point values can sometimes be on the order of 5 to 10 times faster than NUMBER because the floating-point data types use the native instruction set supplied by the hardware vendor. Compared to the NUMBER data type, the floating-point data types use up less space for values stored with significant precision.

Oracle offers the BINARY_FLOAT and BINARY_DOUBLE data types. A floating-point number consists of three components: a *sign*, the signed *exponent*, and a *significand*. The BINARY_ DOUBLE data type supports a wider range of values than does BINARY_FLOAT. The special operators IS [NOT] NAN and IS [NOT] INFINITE check for is "not a number" (NAN) and infinity, respectively.

If an operation involves a mix of different numeric data types, the operation is performed in the data type with the highest precedence. The order of precedence is BINARY_DOUBLE, BINARY_FLOAT, and then NUMBER. For example, if the operation includes a

PRINTED BY: Monica Gonzalez <monicazalez1@hotmail.com>. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

NUMBER and a BINARY_DOUBLE, the value of the NUMBER data type is implicitly converted to the BINARY_DOUBLE data type. Oracle offers various functions that allow conversions to and from different data types; they are discussed in Lab 5.5.

The ROUND function takes on a slightly different behavior if the data type of the input parameter is not NUMBER but is either BINARY_FLOAT or BINARY_DOUBLE. In this case, the ROUND function rounds toward the nearest even value.

```
SELECT ROUND(3.5), ROUND(3.5f), ROUND(4.5), ROUND(4.5f)
  FROM dual
ROUND(3.5) ROUND(3.5F) ROUND(4.5) ROUND(4.5F)
---------- ----------- ---------- -----------
         4    4.0E+000          5    4.0E+000

1 row selected.
```

The REMAINDER Function

The REMAINDER function calculates the remainder, according to the IEEE specification. The syntax is as follows.

```
REMAINDER(value, divisor)
```

The difference between REMAINDER and the MOD function is that MOD uses FLOOR in its computations, whereas REMAINDER uses ROUND. The next example

PRINTED BY: Monica Gonzalez <monicazalez1@hotmail.com>. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

shows that the results between the MOD and REMAINDER functions can be different.

```
SELECT MOD(23,8), REMAINDER(23,8)
  FROM DUAL
 MOD(23,8) REMAINDER(23,8)
---------- ---------------
         7              -1

1 row selected.
```

Effectively, the computation of the MOD function is (23-(8*FLOOR(23/8))), and the computation of REMAINDER is (23-(8*ROUND(23/8))), as illustrated by the next statement.

```
SELECT (23-(8*FLOOR(23/8))) AS mod,
       (23-(8*ROUND(23/8))) AS remainder
  FROM DUAL
       MOD REMAINDER
---------- ---------
         7        -1

1 row selected.
```

Which Number Function Should You Use?

Table 4.3 lists the functions discussed in this lab. Sometimes, you may nest these functions within other functions. As you progress through the following

PRINTED BY: Monica Gonzalez <monicazalez1@hotmail.com>. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

chapters, you will see the usefulness of some of these functions in writing sophisticated SQL statements.

TABLE 4.3 Number Functions

FUNCTION	**PURPOSE**
ABS(value)	Returns the absolute value
SIGN(value)	Returns the sign of a value, such as 1, −1, and 0
MOD(value, divisor)	Returns the modulus
REMAINDER (value, divisor)	Returns the remainder, according to the IEEE specification
ROUND(value [, precision])	Rounds the value
TRUNC(value [, precision])	Truncates the value
FLOOR(value)	Returns the largest integer
CEIL(value)	Returns the smallest integer

In Chapter 17, “Exploring Data Warehousing Features,” you will learn about additional number functions that help you solve analytical and statistical problems. For example, these functions can help you determine rankings, such as the grades of the top 20 students, or compute the median cost of all courses.

PRINTED BY: Monica Gonzalez <monicazalez1@hotmail.com>. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

Arithmetic Operators

The four mathematical operators (addition, subtraction, multiplication, and division) can be used in a SQL statement and can be combined.

In the following example, each of the four operators is used with course costs. One of the distinct course costs is null; here the computation with a null value yields another null.

```
SELECT DISTINCT cost, cost + 10,
       cost - 10, cost * 10, cost / 10
  FROM course
       COST   COST+10   COST-10   COST*10 COST/10
---------- --------- --------- --------- -------
       1095      1105      1085     10950   109.5
       1195      1205      1185     11950   119.5
       1595      1605      1585     15950   159.5


4 rows selected.
```

Parentheses are used to group computations, indicating precedence of the operators. The following SELECT statement returns distinct course costs increased by 10%. The computation within the parentheses is evaluated first, followed by the addition of the value in the COST column, resulting in a single number. NULL values can

PRINTED BY: Monica Gonzalez <monicazalez1@hotmail.com>. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

be replaced with a default value. You will learn about this topic in Lab 4.3.

```
SELECT DISTINCT cost + (cost * .10)
  FROM course
COST+(COST*.10)
---------------
         1204.5
         1314.5
         1754.5


4 rows selected.
```

LAB 4.2 EXERCISES

a) Describe the effect of the negative precision as a parameter of the ROUND function in the following SQL statement.

```
SELECT 10.245, ROUND(10.245, 1),
ROUND(10.245, -1)
FROM dual
```

b) Write a SELECT statement that displays distinct course costs. In a separate column, show the COST increased by 75% and round the decimals to the nearest dollar.

c) Write a SELECT statement that displays distinct numeric grades from the GRADE table and half

PRINTED BY: Monica Gonzalez <monicazalez1@hotmail.com>. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

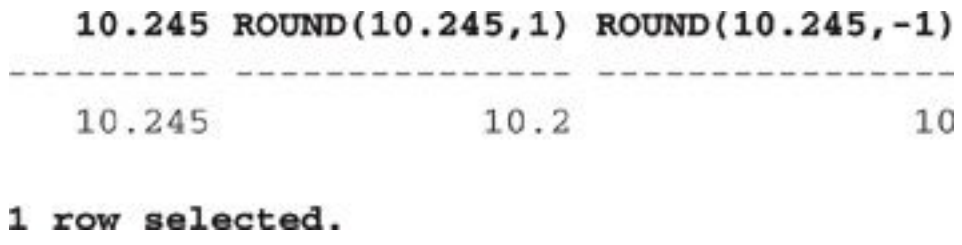
those values expressed as a whole number in a separate column.

LAB 4.2 EXERCISE ANSWERS

a) Describe the effect of the negative precision as a parameter of the ROUND function in the following SQL statement.

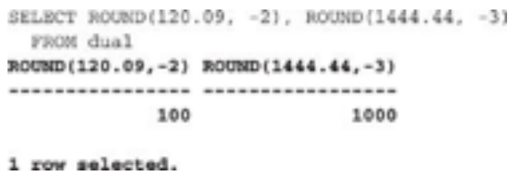
```
SELECT 10.245, ROUND(10.245, 1),
ROUND(10.245, -1)
FROM dual
```

ANSWER: A negative precision rounds digits to the left of the decimal point.



```
   10.245 ROUND(10.245,1) ROUND(10.245,-1)
--------- --------------- ----------------
   10.245            10.2               10

1 row selected.
```

For example, to round to the nearest hundreds, you can use the precision parameter -2, to round to the nearest thousands, use the precision parameter -3, and so on. The next example illustrates the result of these negative parameters.



```
SELECT ROUND(120.09, -2), ROUND(1444.44, -3)
  FROM dual
ROUND(120.09,-2) ROUND(1444.44,-3)
---------------- -----------------
             100              1000

1 row selected.
```

PRINTED BY: Monica Gonzalez <monicazalez1@hotmail.com>. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

b) Write a SELECT statement that displays distinct course costs. In a separate column, show the COST increased by 75% and round the decimals to the nearest dollar.

ANSWER: The SELECT statement uses multiplication and the ROUND function.

```
SELECT DISTINCT cost, cost*1.75, ROUND(cost*1.75)
  FROM course
     COST COST*1.75 ROUND(COST*1.75)
--------- --------- ----------------
     1095   1916.25             1916
     1195   2091.25             2091
     1595   2791.25             2791


4 rows selected.
```

c) Write a SELECT statement that displays distinct numeric grades from the GRADE table and half those values expressed as a whole number in a separate column.

ANSWER: The SELECT statement uses division to derive the value that is half the original value. The resulting value becomes the input parameter for the ROUND function. The displayed output shows a whole number because the ROUND

PRINTED BY: Monica Gonzalez <monicazalez1@hotmail.com>. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

function does not have any precision parameter specified.

```
SELECT DISTINCT numeric_grade, ROUND(numeric_grade / 2)
  FROM grade
NUMERIC_GRADE ROUND(NUMERIC_GRADE/2)
------------- ----------------------
           70                     35
           71                     36
...
           98                     49
           99                     50

30 rows selected.
```

Here, a mathematical computation is combined with a function. Be sure to place computations correctly, either inside or outside the parentheses of a function, depending on the desired result. In this case, if the /2 were on the outside of the ROUND function, a very different result would occur—not the correct answer to the problem posed.

Lab 4.2 Quiz

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

1) Number functions can be nested.

______**a)** True

PRINTED BY: Monica Gonzalez <monicazalez1@hotmail.com>. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

______**b)** False

2) The ROUND function can take only the NUMBER data type as a parameter.

______**a)** True

______**b)** False

3) The following SELECT statement is incorrect.

```
SELECT capacity - capacity
 FROM section
```

______**a)** True

______**b)** False

4) What does the following function return?

```
SELECT LENGTH(NULL)
 FROM dual
```

______**a)** 4

______**b)** 0

______**c)** Null

ANSWERS APPEAR IN APPENDIX A.

PRINTED BY: Monica Gonzalez <monicazalez1@hotmail.com>. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

LAB 4.3 Miscellaneous Single-Row Functions

LAB OBJECTIVES

After this lab, you will be able to:

- Apply Substitution Functions and Other Miscellaneous Functions
- Utilize the Power of DECODE Function and the CASE Expression

In this lab, you will learn about substitution functions to replace nulls with default values. Another important topic is the DECODE function and the CASE expression. They are destined to become your favorites because they allow you to perform powerful *if-then-else* comparisons.

The NVL Function

The NVL function replaces a NULL value with a default value. NULLs represent a special challenge when used in calculations. A computation with an unknown value yields another unknown value, as shown in the following example.

PRINTED BY: Monica Gonzalez <monicazalez1@hotmail.com>. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

```
SELECT 60+60+NULL
  FROM dual
60+60+NULL
---------------


1 row selected.
```

To avoid this problem, you can use the NVL function to substitute the NULL for another value.

```
NVL(input_expression,
substitution_expression)
```

The NVL function requires two parameters: an input expression (for example, a column, literal, or computation) and a substitution expression. If the input expression does *not* contain a NULL value, the input parameter is returned. If the input parameter does contain a NULL value, the substitution parameter is returned.

In the following example, the substitution value is the number literal 1000. The NULL is substituted with 1000, resulting in the output 1120. 167

168

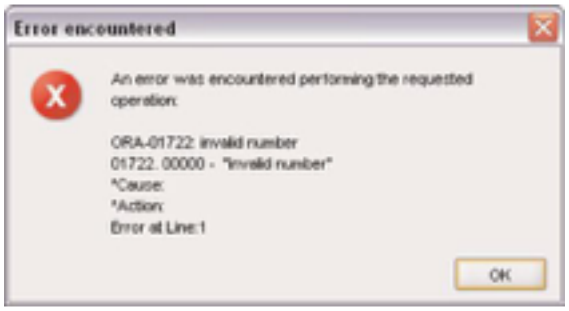
```
SELECT 60+60+NVL(NULL, 1000)
  FROM dual
60+60+NVL(NULL,1000)
--------------------
                1120

1 row selected.
```

PRINTED BY: Monica Gonzalez <monicazalez1@hotmail.com>. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

When you substitute a value, the data type of the substituted value must agree with the data type of the input parameter. The next example uses the NVL function to substitute any NULL values with ‘Not Applicable’ in the PREREQUISITE column. An error is encountered when the statement is executed because the data types of the two parameters are different. The substitution parameter is a text literal, and the column PREREQUISITE is defined as a NUMBER data type. The error shown in Figure 4.5 indicates that Oracle cannot convert the text literal ‘Not Applicable’ into a NUMBER.

```
SELECT course_no, description,
     NVL(prerequisite, 'Not
Applicable') prereq
   FROM course
  WHERE course_no IN (20, 100)
```

FIGURE 4.5 Error message on an invalid number



```
Error encountered

An error was encountered performing the requested
operation:

ORA-01722: invalid number
01722. 00000 - "invalid number"
*Cause:
*Action:
Error at Line: 1
```

PRINTED BY: Monica Gonzalez <monicazalez1@hotmail.com>. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

To overcome this problem, you can transform the output of the PREREQUISITE column into a VARCHAR2 data type, using the TO_CHAR data type conversion function. This function takes a NUMBER or DATE data type and converts it into a string.

```
SELECT course_no, description,
       NVL(TO_CHAR(prerequisite), 'Not Applicable') prereq
  FROM course
 WHERE course_no IN (20, 100)
COURSE_NO DESCRIPTION                 PREREQ
--------- --------------------------- --------------
      100 Hands-On Windows            20
       20 Intro to Information Systems Not Applicable

2 rows selected.
```

The COALESCE Function

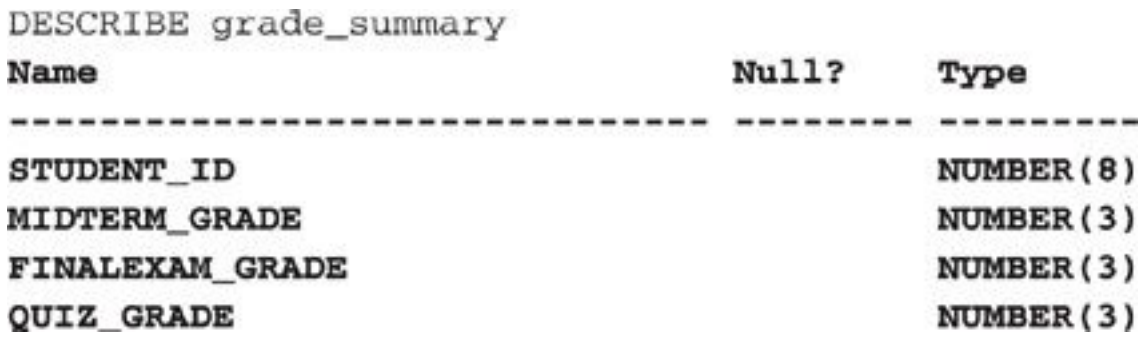
The COALESCE function is similar to the NVL function, but with an additional twist. Instead of specifying one substitution expression for a null value, you can optionally evaluate multiple substitution columns or substitution expressions. The syntax is as follows.

```
COALESCE(input_expression,
substitution_expression_1,
[, substitution_expression_n])
```

PRINTED BY: Monica Gonzalez <monicazalez1@hotmail.com>. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

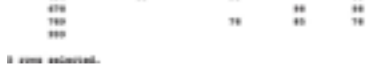
The next SQL query shows a case of two substitution expressions. A table called GRADE_SUMMARY, which is not part of the STUDENT schema, illustrates the idea.

The structure of the GRADE_SUMMARY TABLE is as follows.



```
DESCRIBE grade_summary
Name                           Null?    Type
------------------------------ -------- ---------
STUDENT_ID                              NUMBER(8)
MIDTERM_GRADE                           NUMBER(3)
FINALEXAM_GRADE                         NUMBER(3)
QUIZ_GRADE                              NUMBER(3)
```

In the following example, the resulting output in the Coalesce column shows that if the midterm grade is null, the final exam grade is substituted. If the final exam grade is also null, then the grade for the quiz is substituted. This is the case with student 678, where both the MIDTERM_ GRADE and the FINALEXAM_GRADE column values are null, and therefore the value in the QUIZ_GRADE column is substituted. For student 999, all the column values are null; therefore the COALESCE function returns a null value.



```
SELECT student_id, midterm_grade, finalexam_grade, quiz_grade,
       COALESCE(midterm_grade, finalexam_grade, quiz_grade) "Coalesce"
  FROM grade_summary
STUDENT_ID MIDTERM_GRADE FINALEXAM_GRADE QUIZ_GRADE  Coalesce
---------- ------------- --------------- ---------- --------
       123            90              50        100       90
       456            80              95                  80
       678                                       98       98
       789                            78         85       78
       999

5 rows selected.
```

PRINTED BY: Monica Gonzalez <monicazalez1@hotmail.com>. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

The GRADE_SUMMARY table is a denormalized table, which you typically do not design in such a fashion unless you have a very good reason to denormalize. But the purpose here is to illustrate the functionality of COALESCE. If you wish, you can create this table by downloading an additional script from the companion Web site, located at **www.oraclesqlbyexample.com**.

169

170

The following is an example using the COALESCE function with just one substitution expression, which is equivalent to the NVL function discussed previously. The TO_CHAR function is necessary because the data types of the expressions do not agree. In this case, the PREREQUISITE column is of data type NUMBER. The 'Not Applicable' string is a character constant. You can use the TO_CHAR conversion function to make the two data types equivalent. The TO_CHAR function and conversion functions in general are covered in greater detail in Chapter 5.

```
SELECT course_no, description,
       COALESCE(TO_CHAR(prerequisite), 'Not Applicable') prereq
  FROM course
 WHERE course_no IN (20, 100)
COURSE_NO DESCRIPTION                  PREREQ
--------- ---------------------------- --------------
      100 Hands-On Windows             20
       20 Intro to Information Systems Not Applicable

2 rows selected.
```

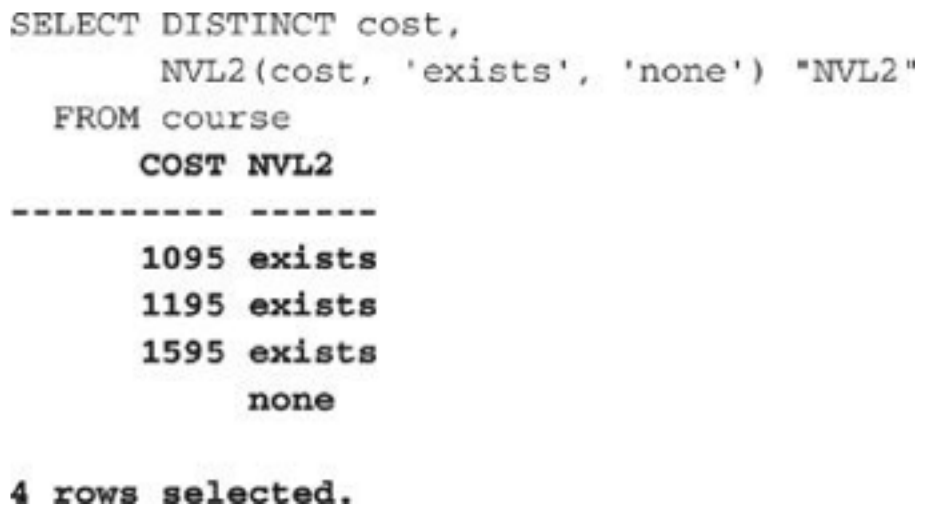
PRINTED BY: Monica Gonzalez <monicazalez1@hotmail.com>. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

The NVL2 Function

The NVL2 function is yet another extension of the NVL function. It checks for both not null and null values and has three parameters versus NVL's two parameters. The syntax for the function is as follows:

```
NVL2(input_expr,
not_null_substitution_expr,
null_substitution_expr)
```

If the input expression is not null, the second parameter of the function, not_null_ substitution_expr, is returned. If the input expression is null, then the last parameter, null_substitution_expr, is returned instead. This query shows how the NVL2 function works. The distinct course costs are displayed; if the value in the COST column is not null, the literal exists is displayed; otherwise the result displays the word none.

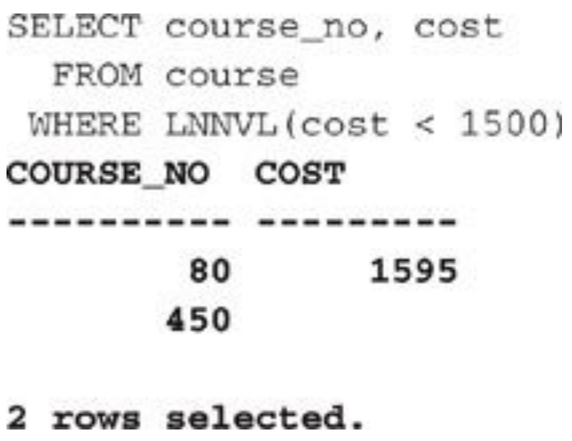


```
SELECT DISTINCT cost,
       NVL2(cost, 'exists', 'none') "NVL2"
  FROM course
      COST NVL2
---------- ------
      1095 exists
      1195 exists
      1595 exists
           none

4 rows selected.
```

PRINTED BY: Monica Gonzalez <monicazalez1@hotmail.com>. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

The LNNVL Function

The LNNVL function can be used only in the WHERE clause of a SELECT statement. It returns either true or false. It returns true and therefore a result if the condition is either false or unknown. The LNNVL function returns the next two rows because the rows do *not* meet the condition COST < 1500. The result displays the courses in the COURSE table where the COST column value is null or greater than 1500.



```
SELECT course_no, cost
  FROM course
 WHERE LNNVL(cost < 1500)
COURSE_NO  COST
---------- ---------
        80      1595
       450

2 rows selected.
```

The syntax for the LNNVL function is as follows.

```
LNNVL(condition)
```

The NULLIF Function

The NULLIF function is unique in that it generates null values. The function compares two expressions; if the values are equal, the function returns a null; otherwise,

PRINTED BY: Monica Gonzalez <monicazalez1@hotmail.com>. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

the function returns the first expression. The following SQL statement returns null for the NULLIF function if the values in the columns CREATED_DATE and MODIFIED_DATE are equal. This is the case for the row with STUDENT_ ID of 150. Both date columns are exactly the same; therefore, the result of the NULLIF function is null. For the row with STUDENT_ID of 340, the columns contain different values; therefore, the first substitution expression is displayed. In this example, you see the use of the TO_CHAR function together with a DATE data type as the input parameter. This allows the display of dates as formatted character strings. This functionality is explained in greater detail in Chapter 5.

```
SELECT student_id,
       TO_CHAR(created_date, 'DD-MON-YY HH24:MI:SS') "Created",
       TO_CHAR(modified_date, 'DD-MON-YY HH24:MI:SS') "Modified",
       NULLIF(created_date, modified_date) "Null if equal"
  FROM student
 WHERE student_id IN (150, 340)
STUDENT_ID Created            Modified           Null if e
---------- ------------------ ------------------ ---------
       150 30-JAN-07 00:00:00 30-JAN-07 00:00:00
       340 19-FEB-07 00:00:00 22-FEB-07 00:00:00 19-FEB-07

2 rows selected.
```

171

172

The syntax for the NULLIF function is as follows.

```
NULLIF(expression1,
equal_expression2)
```

PRINTED BY: Monica Gonzalez <monicazalez1@hotmail.com>. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

The NANVL Function

The NANVL function is used only for the BINARY_FLOAT and BINARY_DOUBLE floating-point data types. The function returns a substitution value in case the input value is NAN ("not a number"). In the following query output, the last row contains such a value. In this instance, the NANVL function's second parameter substitutes the value zero.

The FLOAT_TEST table is part of an additional script that contains sample tables, and it is available for download from the companion Web site, located at **www.oraclesqlbyexample.com.**

```
SELECT test_col, NANVL(test_col, 0)
  FROM float_test
TEST_COL   NANVL(TEST_COL,0)
---------- -----------------
       2.5               2.5
       NaN                 0

2 rows selected.
```

The NANVL function's input and substitution values are numeric, and the syntax is as follows.

PRINTED BY: Monica Gonzalez <monicazalez1@hotmail.com>. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

```
NANVL(input_value,
substitution_value)
```

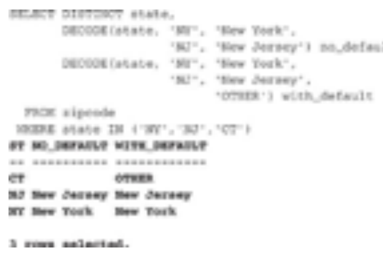
The DECODE Function

The DECODE function substitutes values based on a condition, using *if-then-else* logic. If a value is equal to another value, the substitution value is returned. If the value compared is not equal to any of the listed expressions, an optional default value can be returned. The syntax code for the DECODE function is as follows.

```
DECODE (if_expr, equals_search,
then_result, [else_default])
```

The search and result values can be repeated.

In the following query, the text literals 'New York' and 'New Jersey' are returned when the state is equal to 'NY' or 'NJ', respectively. If the value in the STATE column is other than 'NY' or 'NJ', a null value is displayed. The second DECODE function shows the use of the *else* condition. In the case of 'CT', the function returns the value 'Other'.



```
SELECT DISTINCT state,
       DECODE(state, 'NY', 'New York',
                     'NJ', 'New Jersey') no_default,
       DECODE(state, 'NY', 'New York',
                     'NJ', 'New Jersey',
                           'OTHER') with_default
  FROM zipcode
 WHERE state IN ('NY','NJ','CT')
ST NO_DEFAULT WITH_DEFAULT
-- ---------- ------------
CT            OTHER
NJ New Jersey New Jersey
NY New York   New York

3 rows selected.
```

PRINTED BY: Monica Gonzalez <monicazalez1@hotmail.com>. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

THE DECODE FUNCTION AND NULLS

If you want to specifically test for the null value, you can use the keyword NULL. The following SQL statement shows, for instructors with a null value in the ZIP column, the text NO zipcode! Although one null does not equal another null, for the purpose of the DECODE function, null values are treated as equals.

```
SELECT instructor_id, zip,
       DECODE(zip, NULL, 'NO zipcode!', zip) "Decode Use"
  FROM instructor
 WHERE instructor_id IN (102, 110)
INSTRUCTOR_ID ZIP   Decode Use
------------- ----- -----------
          110       NO zipcode!
          102 10025 10025

2 rows selected.
```

THE DECODE FUNCTION AND COMPARISONS

The DECODE function does not allow greater than or less than comparisons; however, combining the DECODE function with the SIGN function overcomes this shortcoming.

The following SELECT statement combines the DECODE and SIGN functions to display the course

PRINTED BY: Monica Gonzalez <monicazalez1@hotmail.com>. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

cost as 500 for courses that cost less than 1195. If the course cost is greater than or equal to 1195, the actual cost is displayed. The calculation of the value in the COST column minus 1195 results in a negative number, a positive number, a zero, or null. The SIGN function determines the sign of the calculation and returns, respectively, –1, +1, 0, or null. The DECODE function checks whether the result equals –1. If it does, this indicates that the cost is less than 1195, and the DECODE function returns 500; otherwise, the regular cost is shown.

```
SELECT course_no, cost,
       DECODE(SIGN(cost-1195),-1, 500, cost) newcost
  FROM course
 WHERE course_no IN (80, 20, 135, 450)
 ORDER BY 2
COURSE_NO   COST    NEWCOST
----------- ------- -------
        135    1095     500
         20    1195    1195
         80    1595    1595
        450

4 rows selected.
```

The Searched CASE Expression

A searched CASE expression is extremely powerful and can be utilized in many ways. You will see examples in

PRINTED BY: Monica Gonzalez <monicazalez1@hotmail.com>. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

the SELECT list, the WHERE clause, the ORDER BY clause, as a parameter of a function, and anywhere else an expression is allowed. Using a CASE expression is, in many cases, easier to understand, less restrictive, and more versatile than applying the DECODE function. For example, the following query accomplishes the same result as the previous query.

```
SELECT course_no, cost,
       CASE WHEN cost <1195 THEN 500
            ELSE cost
       END "Test CASE"
  FROM course
 WHERE course_no IN (80, 20, 135, 450)
 ORDER BY 2
COURSE_NO       COST  Test CASE
---------- -------- ----------
       135     1095        500
        20     1195       1195
        80     1595       1595
       450

4 rows selected.
```

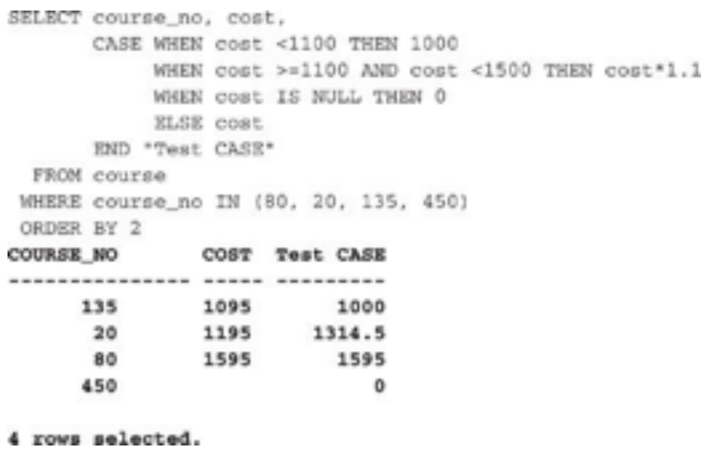
Each CASE expression starts with the keyword CASE and ends with the keyword END; the ELSE clause is optional. A condition is tested with the WHEN keyword; if the condition is true, the THEN clause is executed. The result of the query shows 500 in the column labeled Test CASE when the value in the COST column is less than 1195; otherwise, it just displays the value of the COST

PRINTED BY: Monica Gonzalez <monicazalez1@hotmail.com>. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

column. Following is the syntax of the searched CASE expression.

```
CASE {WHEN condition THEN
return_expr
  [WHEN condition THEN
return_expr]... }
  [ELSE else_expr]
END
```

The next example expands the WHEN condition of the CASE expression, with multiple conditions being tested. The first condition checks whether the value in the COST column is less than 1100; when true, the result evaluates to 1000. If the value in the COST column is equal to or greater than 1100, but less than 1500, the value in the COST column is multiplied by 1.1, increasing the cost by 10%. If the value in the COST column is null, then the value zero is the result. If none of the conditions are true, the ELSE clause is returned.



```
SELECT course_no, cost,
       CASE WHEN cost <1100 THEN 1000
            WHEN cost >=1100 AND cost <1500 THEN cost*1.1
            WHEN cost IS NULL THEN 0
            ELSE cost
       END "Test CASE"
  FROM course
 WHERE course_no IN (80, 20, 135, 450)
 ORDER BY 2
COURSE_NO       COST  Test CASE
--------------- ----- ---------
      135        1095      1000
       20        1195    1314.5
       80        1595      1595
      450                     0

4 rows selected.
```

PRINTED BY: Monica Gonzalez <monicazalez1@hotmail.com>. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

The CASE expression lets you evaluate if-then-else conditions more simply than the DECODE function.

NESTING CASE EXPRESSIONS

A CASE expression can be nested further with additional CASE expressions, as shown in the next example. An additional row with the COURSE_NO 230 is included in this query to demonstrate the result of the nested expression. This nested expression is evaluated only if COST is less than 1100. If this expression is true, the value of the PREREQUISITE column is checked; if it is either 10 or 50, the cost is cut in half. If the PREREQUISITE column does not have the value 10 or 50, just the value in the COST is displayed.

```
SELECT course_no, cost, prerequisite,
       CASE WHEN cost <1100 THEN
                 CASE WHEN prerequisite IN (10, 50) THEN cost/2
                      ELSE cost
                 END
            WHEN cost >=1100 AND cost <1500 THEN cost*1.1
            WHEN cost IS NULL THEN 0
            ELSE cost
       END "Test CASE"
  FROM course
```

PRINTED BY: Monica Gonzalez <monicazalez1@hotmail.com>. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

```
 WHERE course_no IN (80, 20, 135, 450, 230)
 ORDER BY 2
COURSE_NO       COST PREREQUISITE  Test CASE
-------------- ---- ------------- ---------
      230       1095           10     547.5
      135       1095          134      1095
       20       1195                 1314.5
       80       1595          204      1595
      450                     350         0


5 rows selected.
```

CASE EXPRESSION IN THE WHERE CLAUSE

Case expressions are allowed anywhere expressions are allowed; the following example shows a CASE expression in the WHERE clause. It multiplies the CAPACITY column by the result of the CASE expression that returns either 2, 1.5, or null, depending on the starting letter of the value in the LOCATION column. Only if the result of the CASE expression is greater than 30 is the row chosen for output.

```
SELECT DISTINCT capacity, location
  FROM section
WHERE capacity*CASE
        WHEN SUBSTR(location, 1,1)='L' THEN 2
        WHEN SUBSTR(location, 1,1)='M' THEN 1.5
        ELSE NULL
      END  > 30
CAPACITY  LOCATION
--------- --------
       25 L210
...
       25 M500

8 rows selected.
```

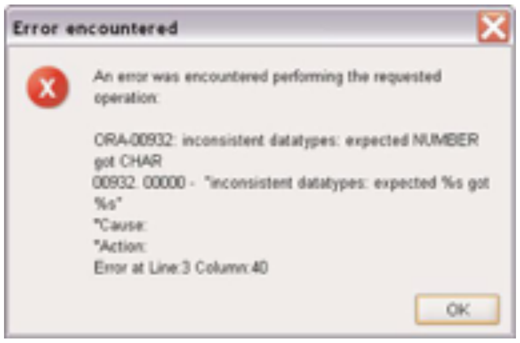
PRINTED BY: Monica Gonzalez <monicazalez1@hotmail.com>. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

DATA TYPE INCONSISTENCIES

You may come across the error message shown in Figure 4.6 when executing the CASE expression or the DECODE function. It indicates that the return data type of the first condition does not agree with the data type of the subsequent conditions. The first CASE condition returns a NUMBER data type, and the second condition returns the character string ‘Room too small’.

```
SELECT section_id, capacity,
  CASE WHEN capacity >=15 THEN
capacity
   WHEN capacity < 15 THEN 'Room
too small'
  END AS ”Capacity”
 FROM section
WHERE section_id IN (101, 146, 147)
```

FIGURE 4.6 Error message on inconsistent data types



```
An error was encountered performing the requested
operation:

ORA-00932: inconsistent datatypes: expected NUMBER
got CHAR
00932. 00000 -  "inconsistent datatypes: expected %s got
%s"
*Cause:
*Action:
Error at Line:3 Column:40
```

PRINTED BY: Monica Gonzalez <monicazalez1@hotmail.com>. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

You match the two data types with a conversion function. The next example shows the use of the TO_CHAR conversion function to convert the values of the CAPACITY column to a character data type.

```
SELECT section_id, capacity,
       CASE WHEN capacity >=15 THEN TO_CHAR(capacity)
            WHEN capacity < 15 THEN 'Room too small'
       END AS "Capacity"
  FROM section
 WHERE section_id IN (101, 146, 147)
SECTION_ID   CAPACITY Capacity
------------ -------- -------------
       147         15 15
       146         25 25
       101         10 Room too small

3 rows selected.
```

Simple CASE Expression

If your conditions are testing for equality only, you can use a simple CASE expression, which has the following syntax.

```
CASE {expr WHEN comparison_expr
THEN return_expr
   [WHEN comparison_expr THEN
return_expr]...}
   [ELSE else_expr]
END
```

PRINTED BY: Monica Gonzalez <monicazalez1@hotmail.com>. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

The next query example checks the value in the COST column to see if it equals the different amounts, and, if true, the appropriate THEN expression is executed.

```
SELECT course_no, cost,
       CASE cost WHEN 1095 THEN cost/2
                 WHEN 1195 THEN cost*1.1
                 WHEN 1595 THEN cost
                 ELSE cost*0.5
       END "Simple CASE"
  FROM course
 WHERE course_no IN (80, 20, 135, 450)
 ORDER BY 2
 COURSE_NO       COST Simple CASE
---------------- ---- -----------
       135       1095       547.5
        20       1195      1314.5
        80       1595        1595
       450


4 rows selected.
```

Rather than hard-coding literals in CASE expressions, you can use subqueries to read dynamic values from tables instead. You will learn more about this in Chapter 8, “Subqueries.”

PRINTED BY: Monica Gonzalez <monicazalez1@hotmail.com>. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

Which Functions and CASE Expressions Should You Use?

Table 4.4 lists the miscellaneous functions and CASE expressions discussed in this lab.

PRINTED BY: Monica Gonzalez <monicazalez1@hotmail.com>. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

TABLE 4.4 Miscellaneous Functions and CASE expressions

FUNCTION/EXPRESSION	**PURPOSE**
NVL(input_expression, substitution_expression)	Null value replacement.
COALESCE(input_expression, substitution substitution_expression_1, [, expressions. substitution_expression_*n*])	Null value replacement with multiple substitution expressions.
NVL2(input_expr, not_null_substitution_expr, null_substitution_expr)	Null and not null substitution replacement.
LNNVL(condition)	Returns true if the condition is false or unknown. Returns false if the condition is true.
NULLIF(expression1, equal_expression2)	Returns null if the value of two expressions are identical; otherwise, returns first expression.
NANVL(input_value, substitution_value)	Returns a substitution value in the case of NAN (not a number) value.
DECODE (if_expr, equals_search, then_result, [else_default])	Substitution function based on if-then-else logic.

PRINTED BY: Monica Gonzalez <monicazalez1@hotmail.com>. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

CASE {WHEN cond THEN return_expr [WHEN cond THEN return_ expr]…} [ELSE else_expr] END	Searched CASE expression. It allows for testing of null values and other comparisons.
CASE {expr WHEN expr THEN return_expr [WHEN expr THEN return_expr]…} [ELSE else_expr] END	The simple CASE expression tests for equality only. No greater than, less than, or IS NULL comparisons are allowed.

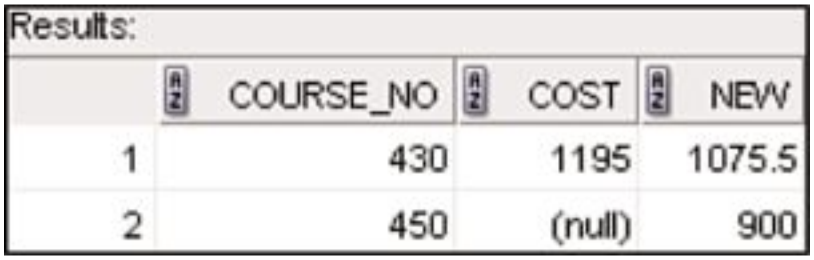
LAB 4.3 EXERCISES

a) List the last names, first names, and phone numbers of students who do not have phone numbers. Display 212-555-1212 for the phone number.

b) For course numbers 430 and greater, show the course cost. Add another column, reflecting a discount of 10% off the cost, and substitute any NULL values in the COST column with the number 1000. The result should look similar to the output shown in Figure 4.7.

PRINTED BY: Monica Gonzalez <monicazalez1@hotmail.com>. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

FIGURE 4.7 Output reflecting the course cost with a 10% discount



Results:

	COURSE_NO	COST	NEW
1	430	1195	1075.5
2	450	(null)	900

c) Write the query to accomplish the following output, using the NVL2 function in the column Get this result.

```
ID  NAME           PHONE        Get this result
--- -------------- ------------ -----------------
112 Thomas Thomas  201-555-5555 Phone# exists.
111 Peggy Noviello              No phone# exists.

2 rows selected.
```

d) Rewrite the query from Exercise c, using the DECODE function instead.

e) For course numbers 20, 120, 122, and 132, display the description, course number, and prerequisite course number. If the prerequisite is course number 120, display 200; if the prerequisite is 130, display N/A. For courses with no prerequisites, display None. Otherwise, list

PRINTED BY: Monica Gonzalez <monicazalez1@hotmail.com>. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

the current prerequisite. The result should look as follows.

```
COURSE_NO DESCRIPTION                ORIGINAL NEW
--------- -------------------------- -------- ----
      132 Basics of Unix Admin            130 N/A
      122 Intermediate Java Programming   120 200
      120 Intro to Java Programming        80 80
       20 Intro to Information Systems        None

4 rows selected.
```

f) Display the student IDs, zip codes, and phone numbers for students with student IDs 145, 150, or 325. For students living in the 212 area code and in zip code 10048, display North Campus. List students living in the 212 area code but in a different zip code as West Campus. Display students outside the 212 area code as Off Campus. The result should look like the following output. Hint: The solution to this query requires nested DECODE functions or nested CASE expressions.

```
STUDENT_ID ZIP   PHONE           LOC
---------- ----- --------------- ------------
       145 10048 212-555-5555    North Campus
       150 11787 718-555-5555    Off Campus
       325 10954 212-555-5555    West Campus

3 rows selected.
```

PRINTED BY: Monica Gonzalez <monicazalez1@hotmail.com>. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

g) Display all the distinct salutations used in the INSTRUCTOR table. Order them alphabetically, except for female salutations, which should be listed first. Hint: Use the DECODE function or CASE expression in the ORDER BY clause.

LAB 4.3 EXERCISE ANSWERS

a) List the last names, first names, and phone numbers of students who do not have phone numbers. Display 212-555-1212 for the phone number.

ANSWER: There are various solutions to obtain the desired result. First, you can determine the rows with a NULL phone number, using the IS NULL operator. Then you apply the NVL function to the column with the substitution string '212-555-1212'. The second solution uses the NVL function in both the SELECT and WHERE clauses. Another way to achieve the result is to use the COALESCE function.

```
SELECT first_name||' '|| last_name name,
       phone oldphone,
       NVL(phone, '212-555-1212') newphone
  FROM student
 WHERE phone IS NULL
NAME                      OLDPHONE       NEWPHONE
------------------------- -------------- ------------
Peggy Noviello                           212-555-1212

1 row selected.
```

PRINTED BY: Monica Gonzalez <monicazalez1@hotmail.com>. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

You can also retrieve the same rows by applying the NVL function in the WHERE clause.

```
SELECT first_name||' '|| last_name name,
       phone oldphone,
       NVL(phone, '212-555-1212') newphone
  FROM student
 WHERE NVL(phone, 'NONE') = 'NONE'
NAME                      OLDPHONE       NEWPHONE
------------------------- -------------- -----------
Peggy Noviello                           212-555-1212

1 row selected.
```

The next query applies the COALESCE function to achieve the same result.

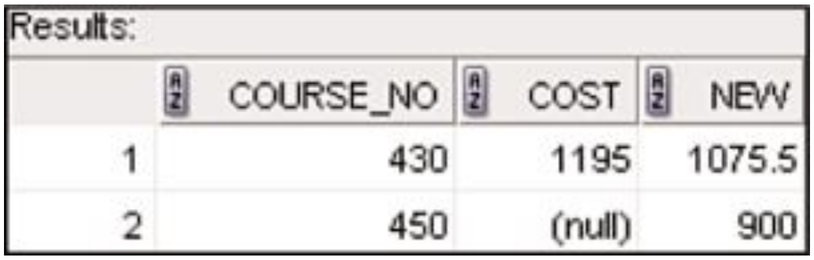
```
SELECT first_name||' '||
last_name name,
  phone oldphone,
  COALESCE(phone, '212-555-1212')
newphone
 FROM student
WHERE COALESCE(phone, 'NONE')
='NONE'
```

b) For course numbers 430 and greater, show the course cost. Add another column, reflecting a discount of 10% off the cost, and substitute any NULL values in the COST column with the

PRINTED BY: Monica Gonzalez <monicazalez1@hotmail.com>. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

number 1000. The result should look similar to the output shown in Figure 4.8.

FIGURE 4.8 Output reflecting the course cost with a 10% discount



Results:

	COURSE_NO	COST	NEW
1	430	1195	1075.5
2	450	(null)	900

ANSWER: Substitute 1000 for the null value, using the NVL function before applying the discount calculation. Otherwise, the calculation yields a NULL.

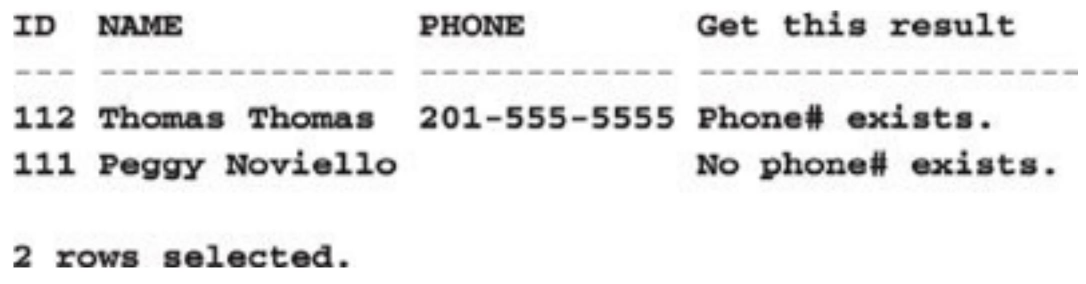
```
SELECT course_no, cost,
  NVL(cost,1000)*0.9 new
 FROM course
WHERE course_no >= 430
```

You can use the COALESCE function instead.

```
SELECT course_no, cost,
  COALESCE(cost,1000)*0.9 new
 FROM course
WHERE course_no >= 430
```

PRINTED BY: Monica Gonzalez <monicazalez1@hotmail.com>. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

c) Write the query to accomplish the following output, using the NVL2 function in the column Get this result.



```
ID  NAME             PHONE        Get this result
--- ---------------- ------------ -----------------
112 Thomas Thomas    201-555-5555 Phone# exists.
111 Peggy Noviello                No phone# exists.

2 rows selected.
```

ANSWER: If the input parameter is not null, the NVL2 function's second parameter is returned. If the input parameter is null, then the third parameter is used.

```
SELECT student_id id,
first_name||' '|| last_name name,
   phone,
   NVL2(phone, 'Phone# exists.',
'No phone# exists.')
   "Get this result"
 FROM student
WHERE student_id IN (111, 112)
ORDER BY 1 DESC
```

d) Rewrite the query from Exercise c, using the DECODE function instead.

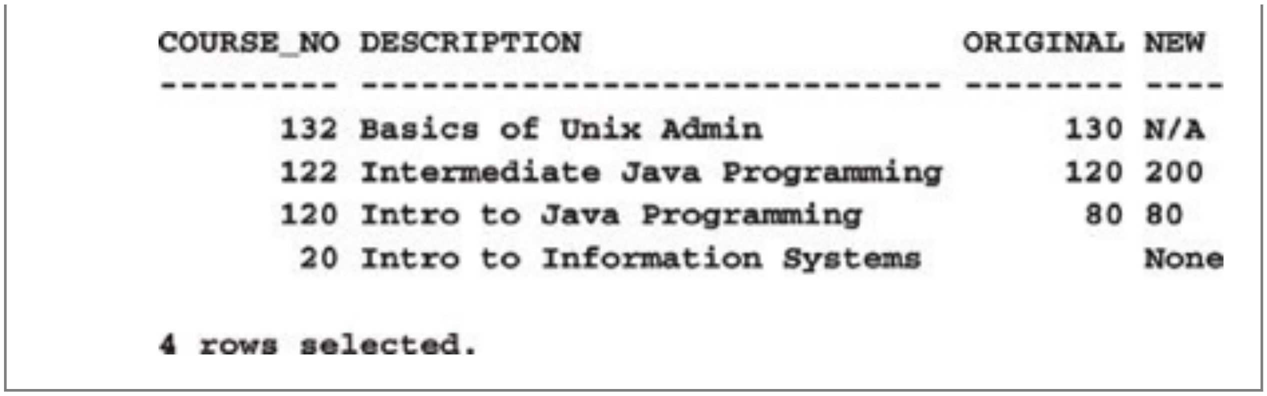
PRINTED BY: Monica Gonzalez <monicazalez1@hotmail.com>. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

ANSWER: The DECODE function can easily be substituted for the NVL2 function or the NVL function, because you can test for a NULL value. In this result, the DECODE function checks whether the value is null. If this is true, the 'No phone# exists.' literal is displayed; otherwise, it shows 'Phone# exists.'

```
SELECT student_id, first_name||'
'|| last_name name,
   phone,
   DECODE(phone, NULL, 'No phone#
exists.', 'Phone# exists.')
   "Get this result"
 FROM student
WHERE student_id IN (111, 112)
ORDER BY 1 DESC
```

e) For course numbers 20, 120, 122, and 132, display the description, course number, and prerequisite course number. If the prerequisite is course number 120, display 200; if the prerequisite is 130, display N/A. For courses with no prerequisites, display None. Otherwise, list the current prerequisite. The result should look as follows.

PRINTED BY: Monica Gonzalez <monicazalez1@hotmail.com>. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.



```
COURSE_NO DESCRIPTION                    ORIGINAL NEW
--------- ---------------------------- -------- ----
      132 Basics of Unix Admin              130 N/A
      122 Intermediate Java Programming     120 200
      120 Intro to Java Programming          80 80
       20 Intro to Information Systems          None

4 rows selected.
```

ANSWER: The solution can be achieved with either the CASE expression or the DECODE function.

SOLUTION USING THE CASE EXPRESSION

The following solution query checks for nulls with the IS NULL condition. The ELSE clause requires you to convert the NUMBER data type into a VARCHAR2, using the TO_CHAR function; otherwise, you receive error ORA-00932 (“inconsistent data types”), indicating that the output data types do not match. Oracle expects the data type to be consistent, with the same data type as the first result expression, which is a string, as indicated by the single quotes around 200.

```
SELECT course_no, description,
prerequisite ”Original”,
  CASE WHEN prerequisite = 120 THEN
'200'
```

PRINTED BY: Monica Gonzalez <monicazalez1@hotmail.com>. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

```
   WHEN prerequisite = 130 THEN 'N/
A'
   WHEN prerequisite IS NULL THEN
'None'
   ELSE TO_CHAR(prerequisite)
  END "NEW"
 FROM course
WHERE course_no IN (20, 120, 122,
132)
ORDER BY course_no DESC
```

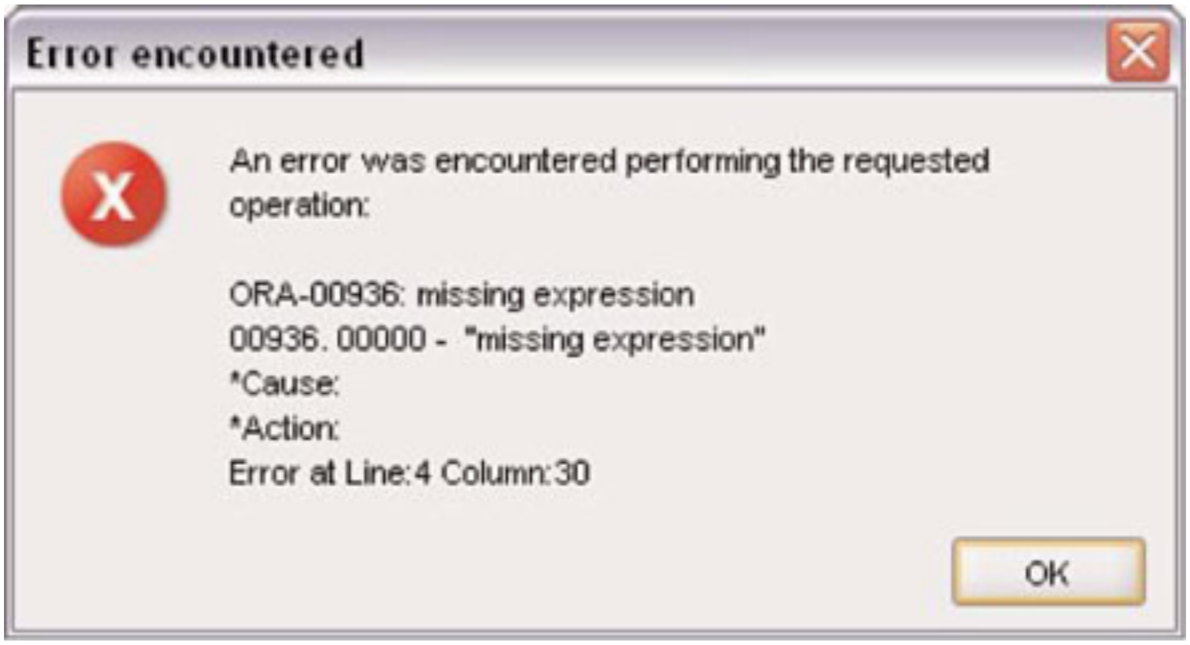
If you attempt to use the simple CASE expression to solve the query, the test for the null value cannot be accomplished because the simple CASE expression only allows testing for equality (=). The IS NULL operator is not permitted and returns the error ORA-00936 (“missing expression”) (see Figure 4.9).

```
SELECT course_no, description,
prerequisite "Original",
  CASE prerequisite WHEN 120 THEN
'200'
   WHEN 130 THEN 'N/A'
   WHEN IS NULL THEN 'None'
   ELSE TO_CHAR(prerequisite)
  END "NEW"
 FROM course
```

PRINTED BY: Monica Gonzalez <monicazalez1@hotmail.com>. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

```
WHERE course_no IN (20, 120, 122,
132)
```

FIGURE 4.9 An ORA-00936 error


Error encountered

An error was encountered performing the requested operation:

ORA-00936: missing expression
00936. 00000 - "missing expression"
*Cause:
*Action:
Error at Line:4 Column:30

SOLUTION USING THE DECODE FUNCTION

The following solution is best approached in several steps. The PREREQUISITE column is of data type NUMBER. If you replace it in the DECODE function with another NUMBER for prerequisite 120, Oracle expects to continue to convert to the same data type for all subsequent replacements. As the other replacements ('N/A' and 'None') are text literals, you need to enclose the number 200 in single quotation marks to

PRINTED BY: Monica Gonzalez <monicazalez1@hotmail.com>. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

predetermine the data type for all subsequent substitutions as a VARCHAR2.

For any records that have a null prerequisite, None is displayed. Although one null does not equal another null, for the purpose of the DECODE function, null values are treated as equals.

```
SELECT course_no, description, prerequisite "ORIGINAL",
       DECODE(prerequisite, 120, '200',
                            130, 'N/A',
                            NULL, 'None',
                            TO_CHAR(prerequisite)) "NEW"
  FROM course
 WHERE course_no IN (20, 120, 122, 132)
 ORDER BY course_no DESC
```

Explicit data type conversion with the TO_CHAR function on PREREQUISITE is good practice, although if you omit it, Oracle implicitly converts the value to a VARCHAR2 data type. The automatic data type conversion works in this example because the data type is predetermined by the data type of the first substitution value.

f) Display the student IDs, zip codes, and phone numbers for students with student IDs 145, 150, or 325. For students living in the 212 area code and in zip code 10048, display North Campus. List students living in the 212 area code but in a

PRINTED BY: Monica Gonzalez <monicazalez1@hotmail.com>. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

different zip code as West Campus. Display students outside the 212 area code as Off Campus. The result should look like the following output. Hint: The solution to this query requires nested DECODE functions or nested CASE expressions.

```
STUDENT_ID ZIP   PHONE          LOC
---------- ----- -------------- -----------
       145 10048 212-555-5555   North Campus
       150 11787 718-555-5555   Off Campus
       325 10954 212-555-5555   West Campus

3 rows selected.
```

ANSWER: The CASE expressions can be nested within each other to allow for the required logic. A more complicated way to obtain the desired result is by using nested DECODE statements; the output from one DECODE is an input parameter in a second DECODE function.

Following is the solution using the CASE expression.

```
SELECT student_id, zip, phone,
       CASE WHEN SUBSTR(phone, 1, 3) = '212' THEN
                 CASE WHEN zip = '10048' THEN 'North Campus'
                      ELSE 'West Campus'
                 END
            ELSE 'Off Campus'
       END loc
  FROM student
 WHERE student_id IN (150, 145, 325)
```

PRINTED BY: Monica Gonzalez <monicazalez1@hotmail.com>. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

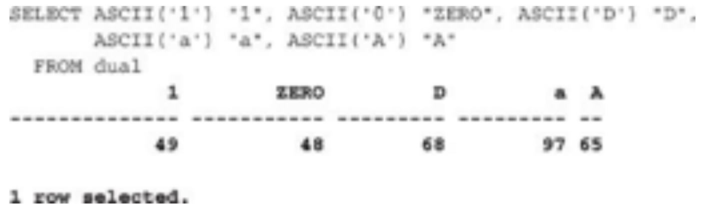
The next solution uses the DECODE function.

```
SELECT student_id, zip, phone,
 DECODE(SUBSTR(phone, 1, 3),
'212',
  DECODE(zip, '10048', 'North
Campus',
    'West Campus'),
```

```
       'Off Campus') loc
   FROM student
WHERE student_id IN (150, 145,
325)
```

g) Display all the distinct salutations used in the INSTRUCTOR table. Order them alphabetically, except for female salutations, which should be listed first. Hint: Use the DECODE function or CASE expression in the ORDER BY clause.

ANSWER: The DECODE function or the CASE expression is used in the ORDER BY clause to substitute a number for all female salutations, thereby listing them first when executing the ORDER BY clause.



```
SELECT ASCII('1') "1", ASCII('0') "ZERO", ASCII('D') "D",
       ASCII('a') "a", ASCII('A') "A"
  FROM dual
             1        ZERO          D          a  A
-------------- ----------- ---------- ---------- --
            49          48         68         97 65

1 row selected.
```

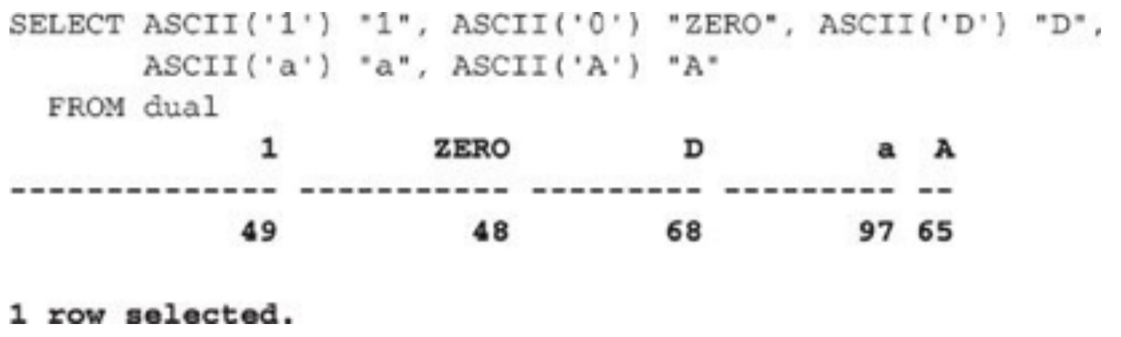
PRINTED BY: Monica Gonzalez <monicazalez1@hotmail.com>. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

The following shows the CASE expression.

```
SELECT DISTINCT salutation
 FROM instructor
ORDER BY CASE salutation WHEN 'Ms'
THEN '1'
      WHEN 'Mrs' THEN '1'
      WHEN 'Miss' THEN '1'
      ELSE salutation
   END
```

The ASCII equivalent number of 1 is less than the ASCII equivalent of Dr, or any other salutation. Therefore, Ms is listed first in the sort order. (ASCII, which stands for American Standard Code for Information Interchange, deals with common formats.)

To display the decimal representation of the first character of a string, use the ASCII function. The following query is an example of how you can determine the ASCII numbers of various values.



```
SELECT ASCII('1') "1", ASCII('0') "ZERO", ASCII('D') "D",
       ASCII('a') "a", ASCII('A') "A"
  FROM dual
             1        ZERO         D         a  A
-------------- ----------- --------- --------- --
            49          48        68        97 65

1 row selected.
```

PRINTED BY: Monica Gonzalez <monicazalez1@hotmail.com>. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

Lab 4.3 Quiz

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

1) A calculation with a null always yields another null.

______**a)** True

______**b)** False

2) The following query is valid.

```
SELECT NVL(cost, 'None')
 FROM course
```

______**a)** True

______**b)** False

3) The NVL2 function updates the data in the database.

______**a)** True

______**b)** False

4) The DECODE function lets you perform if-then-else functionality within the SQL language.

______**a)** True

______**b)** False

PRINTED BY: Monica Gonzalez <monicazalez1@hotmail.com>. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

5) The DECODE function cannot be used in the WHERE clause of a SQL statement.

______**a)** True

______**b)** False

6) CASE expressions can be used in the ORDER BY clause of a SELECT statement.

______**a)** True

______**b)** False

7) The functions discussed in this lab can be used on the VARCHAR2 data type only.

______**a)** True

______**b)** False

ANSWERS APPEAR IN APPENDIX A.

186

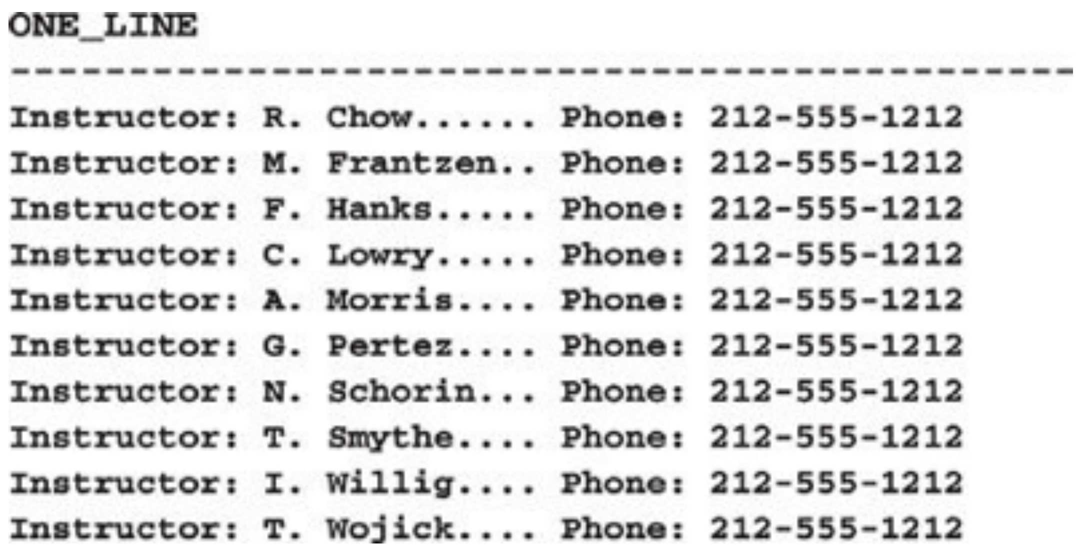
187

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**.

PRINTED BY: Monica Gonzalez <monicazalez1@hotmail.com>. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

1) Write the SELECT statement that returns the following output.



```
ONE_LINE
-----------------------------------------------------------
Instructor: R. Chow...... Phone: 212-555-1212
Instructor: M. Frantzen.. Phone: 212-555-1212
Instructor: F. Hanks..... Phone: 212-555-1212
Instructor: C. Lowry..... Phone: 212-555-1212
Instructor: A. Morris.... Phone: 212-555-1212
Instructor: G. Pertez.... Phone: 212-555-1212
Instructor: N. Schorin... Phone: 212-555-1212
Instructor: T. Smythe.... Phone: 212-555-1212
Instructor: I. Willig.... Phone: 212-555-1212
Instructor: T. Wojick.... Phone: 212-555-1212
```

2) Rewrite the following query to replace all occurrences of the string 'Unix' with 'Linux'.

```
SELECT 'I develop software on the
Unix platform'
 FROM dual
```

3) Determine which student does not have the first letter of her or his last name capitalized. Show the STUDENT_ID and LAST_NAME columns.

4) Check whether any of the phone numbers in the INSTRUCTOR table have been entered in the (###)###-#### format.

5) Explain the functionality of the following query.

PRINTED BY: Monica Gonzalez <monicazalez1@hotmail.com>. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

```
SELECT section_id, capacity,
  CASE WHEN MOD(capacity, 2) <> 0
THEN 'Odd capacity'
   ELSE 'Even capacity'
  END ”Odd or Even”
 FROM section
WHERE section_id IN (101, 146,
147)
```