CHAPTER 5 Date and Conversion 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:

- Applying Oracle's Date Format Models
- Performing Date and Time Math
- Understanding Timestamp and Time Zones Data Types
- Performing Calculations with the Interval Data Types
- Converting from One Data Type to Another

In this chapter, you will gain an understanding of Oracle's two date-related categories of data types: the datetime data types and the interval data types.

The datetime data types keep track of both date and time; they consist of the individual data types DATE, TIMESTAMP, TIMESTAMP WITH TIME ZONE, and

TIMESTAMP WITH LOCAL TIME ZONE. In the first two labs, you will learn about the most popular data type —the DATE data type. Lab 5.3 introduces you to the other three data types, which contain fractional seconds and time zone values. Table 5.1 shows an overview of the datetime data types.

TABLE 5.1 Overview of Datetime Data Types

	FRACTIONAL		
DATA TYPE	**SECONDS**	**TIME ZONE**	**LAB**
DATE	No	No	5.1, 5.2
TIMESTAMP	Yes	No	5.3
TIMESTAMP WITH TIME ZONE	Yes	Yes	5.3
TIMESTAMP WITH LOCAL TIME ZONE	Yes	Yes	5.3

189

190

The interval data types, which are the topic of Lab 5.4, express differences between dates and times. The Oracle-supported interval data types are INTERVAL YEAR TO MONTH and INTERVAL DAY TO SECOND (see Table 5.2).

TABLE 5.2 Overview of Interval Data Types

DATA TYPE	SUPPORTED TIME DIFFERENCES
INTERVAL YEAR TO MONTH	Years and months
INTERVAL DAY TO SECOND	Days, hours, minutes, and seconds

In Lab 5.5, you will become familiar with using data type conversion functions—an important skill for dealing with data.

SQL novices often find date and conversion functions challenging, but the many examples in this chapter's labs will help you master these functions and avoid the common pitfalls.

LAB 5.1 Applying Oracle's Date Format Models

LAB OBJECTIVES

After this lab, you will be able to:

- Compare a Text Literal to a DATE Column

- Apply Various Format Models

When working with an Oracle database, you will inevitably need to query columns that contain dates. Oracle's DATE data type consists of a *date and time* that are stored in an internal format that keeps track of the century, year, month, day, hour, minute, and second.

Changing the Date Display Format

When you query a DATE data type column, Oracle displays it in the default format determined by the database NLS_DATE_FORMAT parameter. The most frequent setup values you will see are DD-MON-YYYY and DD-MON-RR. The RR represents a two-digit year based on the century; if the two-digit year is between 50 and 99, then it's the previous century; if the two-digit year is between 00 and 49, it's the current century.

```
SELECT last_name, registration_date
  FROM student
 WHERE student_id IN (123, 161, 190)
LAST_NAME  REGISTRAT
---------  ---------
Affinito   03-FEB-07
Grant      02-FEB-07
Radicola   27-JAN-07

3 rows selected.
```

To change the display format of the column REGISTRATION_DATE, you use the TO_CHAR function together with a format model, also referred to as a *format mask*. The result shows the registration date in both the default date format and the MM/DD/YYYY format.

```
SELECT last_name, registration_date,
       TO_CHAR(registration_date, 'MM/DD/YYYY')
       AS "Formatted"
  FROM student
 WHERE student_id IN (123, 161, 190)
LAST_NAME     REGISTRAT Formatted
---------     --------- ----------
Affinito      03-FEB-07 02/03/2007
Grant         02-FEB-07 02/02/2007
Radicola      27-JAN-07 01/27/2007

3 rows selected.
```

The TO_CHAR conversion function changes the DATE data type into text and applies a format mask. As you see from the syntax listed in Table 5.3, the function takes a DATE data type as the first parameter; the second optional parameter is for the format mask. Table 5.4 lists commonly used elements of date format masks.

TABLE 5.3 Date-Related Conversion Functions

FUNCTION	**PURPOSE**	**RETURN DATA TYPE**
TO_CHAR(date, [format_mask])	Converts datetime data types into VARCHAR2 to use a different display format than the default date format. (The TO_CHAR function can be used with other data types besides DATE; see Lab 5.5.)	VARCHAR2

TO_DATE(char, [format_mask])	Converts a text literal to a DATE data type. As with all other date-related conversion functions, the format_mask is optional if the literal is in the default format; otherwise, a format mask must be specified.	DATE

TABLE 5.4 Commonly Used Elements of the DATE Format Mask

FORMAT	DESCRIPTION
YYYY	Four-digit year.
YEAR	Year, spelled out.
RR	Two-digit year, based on century. If two-digit year is between 50 and 99, then it's the previous century; if the year is between 00 and 49, it's the current century.
MM	Two-digit month.
MON	Three-letter abbreviation of the month, in uppercase letters.
MONTH	Month, spelled out, in uppercase letters and padded with blanks.
Month	Month, spelled with, first letter uppercase and padded with blanks to a length of nine characters.
DD	Numeric day (1–31).
DAY	Day of the week, spelled out, in uppercase letters and padded with blanks to a length of nine characters.
DY	Three-letter abbreviation of the day of the week, in uppercase letters.

D	Day of the week number (1–7), where Sunday is day 1, Monday is day 2, and so forth.
DDD	Day of the year (1–366).
DL	Day long format; the equivalent format mask is fmDay, Month DD, YYYY.
HH or HH12	Hours (0–12).
HH24	Hours in military format (0–23).
MI	Minutes (0–59).
SS	Seconds (0–59).
SSSSS	Seconds past midnight (0–86399).
AM or PM	Meridian indicator.
TS	Short time format; the equivalent format mask is HH:MI:SS AM.
WW	Week of the year (1–53).
W	Week of the month (1–5).
Q	Quarter of the year.

The TO_DATE function does just the opposite of the TO_CHAR function: It converts a text literal into a DATE data type.

The next SQL statement shows the same student record, with the date and time formatted in various ways. The first format model is Dy, which shows the abbreviated day of the week in mixed format. The next format model is DY, and it returns the uppercase version. The fourth column has the month spelled out, but notice the extra spaces after the month. Oracle pads the month with up to nine spaces, which may be useful when you choose to align the month columns. If you want to eliminate the extra spaces, use the fill mask fm. You will see some examples of this format mask shortly. The last column shows only the time.

```
SELECT last_name,
       TO_CHAR(registration_date, 'Dy') AS "1.Day",
       TO_CHAR(registration_date, 'DY') AS "2.Day",
       TO_CHAR(registration_date, 'Month DD, YYYY')
         AS "Look at the Month",
       TO_CHAR(registration_date, 'HH:MI PM') AS "Time"
        FROM student
       WHERE student_id IN (123, 161, 190)
LAST_NAME 1.Da 2.Da Look at the Month  Time
--------- ---- ---- ------------------ --------
Affinito  Sat  SAT  February  03, 2007 12:00 PM
Grant     Fri  FRI  February  02, 2007 12:00 PM
Radicola  Sat  SAT  January   27, 2007 12:00 PM

3 rows selected.
```

Here is a more elaborate example, which uses the *fm* mask to eliminate the extra spaces between the month and the date in the second column of the following result set. In addition, this format mask uses the *th* suffix on the day (dd) mask, to include the st, nd, rd, and th in lowercase after each number. The third and last column spells out the date using the *sp* format parameter, with the first letter capitalized by using the Dd format. Also, you can add a text literal, as in this case with the "of" text.

```
SELECT last_name,
       TO_CHAR(registration_date, 'fmMonth ddth, YYYY')
       "Eliminating Spaces",
       TO_CHAR(registration_date, 'Ddspth "of" fmMonth')
       "Spelled out"
  FROM student
 WHERE student_id IN (123, 161, 190)
LAST_NAME  Eliminating Spaces   Spelled out
---------  ------------------   --------------------------
Affinito   February 3rd, 2007   Third of February
Grant      February 2nd, 2007   Second of February
Radicola   January 27th, 2007   Twenty-Seventh of January

3 rows selected.
```

Table 5.5 shows additional examples of how the format models can be used.

TABLE 5.5 Date Format Model Examples

FORMAT MASK	EXAMPLE
DD-Mon-YYYY HH24:MI:SS	12-Apr-2009 17:00:00 (The case matters!)
MM/DD/YYYY HH:MI PM	04/12/2009 5:00 PM
Month	April
fmMonth DDth, YYYY	April 12th, 2009
Day	Sunday
DY	SUN
Qth YYYY	2nd 2009 (This shows the 2nd quarter of 2009.)
Ddspth	Twelfth (Spells out the date.)
DD-MON-RR	12-APR-09 (You'll learn ore on the RR format later in this lab.)

Performing a Date Search

You'll find that you often need to query data based on certain date criteria. For example, if you need to look for all students with a registration date of January 22, 2007, you write a SQL statement similar to the following.

```
SELECT last_name, registration_date
  FROM student
 WHERE registration_date = TO_DATE('22-JAN-2007', 'DD-MON-YYYY')
LAST_NAME                 REGISTRAT
------------------------- ---------
Crocitto                  22-JAN-07
Landry                    22-JAN-07
...
Sethi                     22-JAN-07
Walter                    22-JAN-07

8 rows selected.
```

In the WHERE clause, the text literal '22-JAN-2007' is converted to a DATE data type using the TO_DATE function and the format model. The TO_DATE function helps Oracle understand the text literal, based on the supplied format mask. The text literal is converted to the DATE data type, which is then compared to the REGISTRATION_DATE column, also of data type DATE. Now you are comparing identical data types.

The format mask needs to agree with your text literal; otherwise, Oracle will not be able to interpret the text literal correctly and will return the following error message.

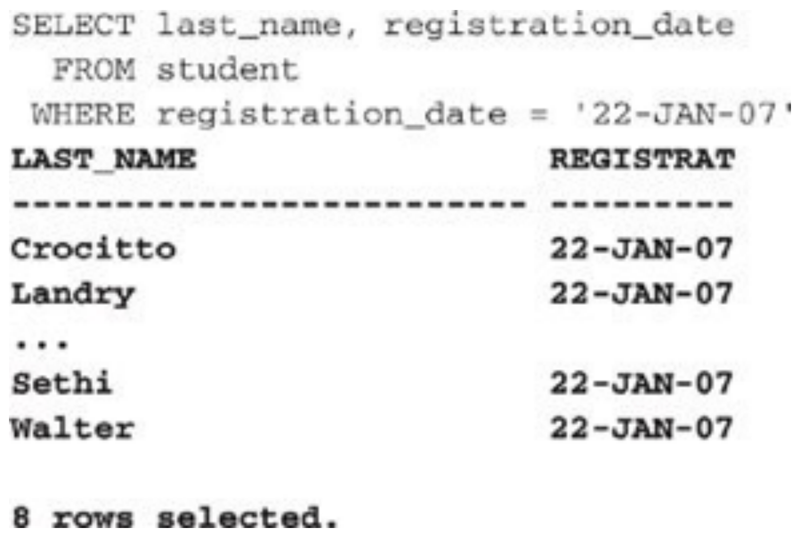
```
SELECT last_name, registration_date
  FROM student
```

```
 WHERE registration_date =
TO_DATE('22/01/2007', 'DD-MON-
YYYY')
```

```
WHERE registration_date =
TO_DATE('22/01/2007', 'DD-MON-
YYYY') *

ERROR at line 3:
ORA-01843: not a valid month
```

Implicit Conversion and Default Date Format

Without a format mask, Oracle can implicitly perform a conversion of the text literal to the DATE data type when the text literal is in the default date format. This default format is determined by the NLS_DATE_FORMAT, an Oracle instance parameter. The next SQL statement shows an example of this.



```
SELECT last_name, registration_date
  FROM student
 WHERE registration_date = '22-JAN-07'
LAST_NAME                 REGISTRAT
------------------------- ---------
Crocitto                  22-JAN-07
Landry                    22-JAN-07
...
Sethi                     22-JAN-07
Walter                    22-JAN-07

8 rows selected.
```

The same result can also be achieved with the following WHERE clause. In this example, it allows either a four-digit or two-digit input of the year. For a two-digit year, Oracle determines the century for you.

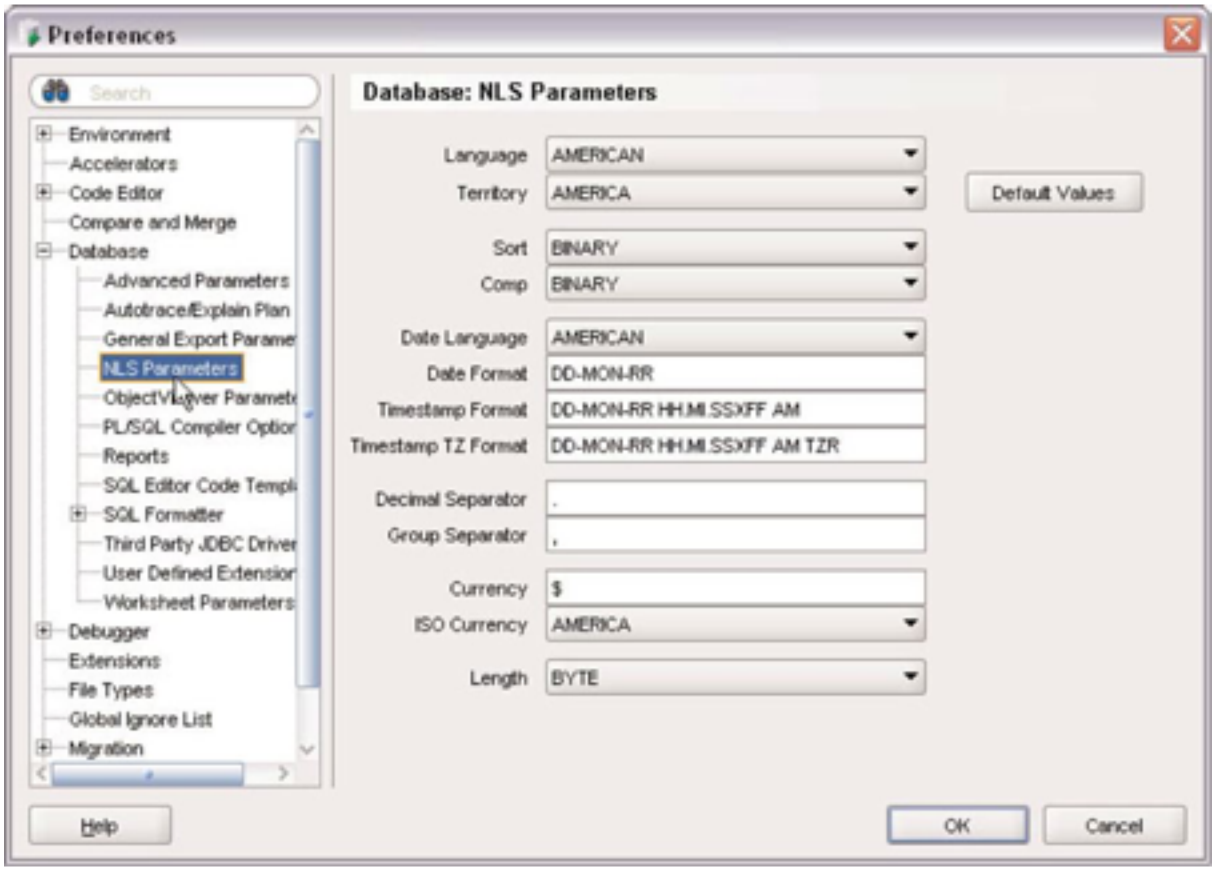
```
WHERE registration_date = '22-
JAN-2007'
```

The NLS_DATE_FORMAT value is defined as Oracle database's initialization parameter, but you can also modify the behavior by changing the Windows registry, or within a tool such as SQL Developer, by selecting Tools, Preferences, Database, NLS Parameters and using the options there (see Figure 5.1). You can also issue the ALTER SESSION command to temporarily set the value.

```
ALTER SESSION SET NLS_DATE_FORMAT =
'DD-MON-RRRR'
```

It is always best to explicitly use the TO_DATE function along with the appropriate format mask when converting a text literal. This makes your statement explicit and independent of any settings. You will see the advantages of doing so as you go through some of the exercises in this chapter.

FIGURE 5.1 SQL Developer's display for NLS parameters



Database: NLS Parameters

Language	AMERICAN
Territory	AMERICA
Sort	BINARY
Comp	BINARY
Date Language	AMERICAN
Date Format	DD-MON-RR
Timestamp Format	DD-MON-RR HH.MI.SSXFF AM
Timestamp TZ Format	DD-MON-RR HH.MI.SSXFF AM TZR
Decimal Separator	.
Group Separator	,
Currency	$
ISO Currency	AMERICA
Length	BYTE

The RR Date Format Mask and the Previous Century

Although the year 2000 is long behind us, you still have to deal with dates in the twentieth century. For example, the next statement retrieves all rows in the

GRADE_TYPE table that were created on December 31, 1998. Notice that the century is missing in the WHERE clause.

```
SELECT grade_type_code, description, created_date
  FROM grade_type
 WHERE created_date = '31-DEC-98'
GR DESCRIPTION          CREATED_D
-------------------- ---------
FI Final                31-DEC-98
HM Homework             31-DEC-98
MT Midterm              31-DEC-98
PA Participation        31-DEC-98
PJ Project              31-DEC-98
QZ Quiz                 31-DEC-98

6 rows selected.
```

The query will return rows only if your Oracle installation includes the DD-MON-RR or DD-MON-
RRRR format mask. This special RR format mask 197
interprets the two-digit year from 50 until 99 as the prior 198
century, which currently is for years from 1950 through 1999. Two-digit year numbers from 00 until 49 are interpreted as the current century—that is, as years 2000 through 2049.

You can also see what your session settings are by issuing the following query, which returns session attributes.

```
SELECT SYS_CONTEXT ('USERENV', 'NLS_DATE_FORMAT')
  FROM dual
SYS_CONTEXT('USERENV','NLS_DATE_FORMAT')
----------------------------------------
DD-MON-RR

1 row selected.
```

If your default format mask is set to DD-MON-YY instead, Oracle interprets '31-DEC-98' as '31-DEC-2098', which is obviously not the desired result. Because databases can have different date configuration settings, it is always best to be specific and to include the four-digit year in your WHERE clause.

The next query illustrates how a two-digit year gets interpreted with the RR format mask. The text literals '17-OCT-67' and '17-OCT-17' are converted to a DATE data type with the format mask DD-MON-RR. Then the TO_CHAR function converts the DATE data type back to text, but this time with a four-digit year. Effectively, the two-digit year 67 is interpreted as 1967, and the two-digit year literal 17 is interpreted as 2017.

```
SELECT TO_CHAR(TO_DATE('17-OCT-67','DD-MON-RR'),'YYYY') "1900",
       TO_CHAR(TO_DATE('17-OCT-17','DD-MON-RR'),'YYYY') "2000"
  FROM dual
1900 2000
---- ----
1967 2017

1 row selected.
```

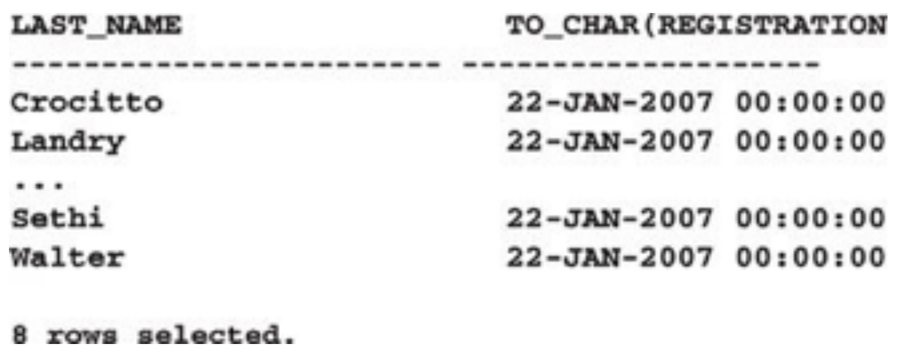
The Time Component

As previously mentioned, the Oracle DATE data type includes the time. You can query records for a specific time or ignore the time altogether. The next SQL statement displays the time as part of the result set. If no time component was included when the data was entered, Oracle assumes that the time is midnight, which is 12:00:00 AM, or 00:00:00 military time (HH24 time format mask). The WHERE clause retrieves only rows where the column has a value of January 22, 2007, midnight; other records with a different time are not returned, if any exist.

```
SELECT last_name,
   TO_CHAR(registration_date, ‘DD-
MON-YYYY HH24:MI:SS’)
  FROM student
 WHERE registration_date =
TO_DATE(‘22-JAN-2007’, ‘DD-MON-
YYYY’)
```

198

199



```
LAST_NAME                 TO_CHAR(REGISTRATION
------------------------- --------------------
Crocitto                  22-JAN-2007 00:00:00
Landry                    22-JAN-2007 00:00:00
...
Sethi                     22-JAN-2007 00:00:00
Walter                    22-JAN-2007 00:00:00

8 rows selected.
```

Time and the TRUNC Function

You already learned about the TRUNC function in connection with the NUMBER data type in Chapter 4, “Character, Number, and Miscellaneous Functions.” The TRUNC function can also take a DATE data type as an input parameter, which interprets the time as midnight (that is, 12:00:00 AM). The next example shows the TRUNC function applied to the ENROLL_DATE column. It has the effect of including the records no matter what the time, as long as the date is February 7, 2007.

```
SELECT student_id, TO_CHAR(enroll_date, 'DD-MON-YYYY HH24:MI:SS')
  FROM enrollment
 WHERE TRUNC(enroll_date) = TO_DATE('07-FEB-2007', 'DD-MON-YYYY')
STUDENT_ID TO_CHAR(ENROLL_DATE,
---------- --------------------
       140 07-FEB-2007 10:19:00
       141 07-FEB-2007 10:19:00
...
       158 07-FEB-2007 10:19:00
       159 07-FEB-2007 10:19:00

20 rows selected.
```

The ANSI DATE and ANSI TIMESTAMP Formats

Instead of using Oracle’s date literals, you can specify a date in the ANSI format listed in the next example. This format contains no time portion and must be listed exactly in the format YYYY-MM-DD, with the DATE keyword prefix.

```
SELECT student_id,
TO_CHAR(enroll_date, 'DD-MON-YYYY
HH24:MI:SS')
  FROM enrollment
 WHERE enroll_date >= DATE
'2007-02-07'
  AND enroll_date < DATE
'2007-02-08'
```

If you want to include the time portion, use the ANSI TIMESTAMP keyword. The literal must be in the ANSI TIMESTAMP format, which is defined as YYYY-MM-DD HH24:MI:SS.

```
SELECT student_id,
TO_CHAR(enroll_date, 'DD-MON-YYYY
HH24:MI:SS')
 FROM enrollment
WHERE enroll_date >= TIMESTAMP
'2007-02-07 00:00:00'
  AND enroll_date < TIMESTAMP
'2007-02-08 00:00:00'
```

LAB 5.1 EXERCISES

a) Display the course number, section ID, and starting date and time for sections taught on May 4, 2007.

b) Show the student records that were modified on or before January 22, 2007. Display the date a record was modified and each student's first and last name, concatenated in one column.

c) Display the course number, section ID, and starting date and time for sections that start on Sundays.

d) List the section ID and starting date and time for all sections that begin and end in July 2007.

e) Determine the day of the week for December 31, 1899.

f) Execute the following statement. Write the questions to obtain the desired result. Pay particular attention to the ORDER BY clause.

```
SELECT ‘Section ’||section_id||‘
       begins on ’||
       TO_CHAR(start_date_time,
       ‘fmDay’)||'.' AS ”Start”
  FROM section
 WHERE section_id IN (146, 127,
       121, 155, 110, 85, 148)
 ORDER BY TO_CHAR(start_date_time,
       ‘D’)
```

LAB 5.1 EXERCISE ANSWERS

a) Display the course number, section ID, and starting date and time for sections taught on May 4, 2007.

ANSWER: To display a DATE column in a nondefault format, use the TO_CHAR function. To compare a text literal to a DATE column, use the TO_DATE function. It is best to always use the four-digit year and the format mask when using the TO_DATE function. This is good practice because it means your queries are not subject to ambiguities if the default date format is different.

```
SELECT course_no, section_id,
       TO_CHAR(start_date_time, 'DD-MON-YYYY HH24:MI')
  FROM section
 WHERE start_date_time >= TO_DATE('04-MAY-2007', 'DD-MON-YYYY')
   AND start_date_time < TO_DATE('05-MAY-2007', 'DD-MON-YYYY')
COURSE_NO SECTION_ID TO_CHAR(START_DAT
--------- ---------- -----------------
       25         88 04-MAY-2007 09:30
      100        144 04-MAY-2007 09:30
      120        149 04-MAY-2007 09:30
      122        155 04-MAY-2007 09:30

4 rows selected.
```

The returned result set displays the starting date and time, using the TO_CHAR function and the specified format mask in the SELECT list. In the WHERE clause, the text literals '04-MAY-2007' and '05-MAY-2007'are transformed into the DATE data type with the TO_DATE function. Because no format mask for the time is specified, Oracle assumes that the time is midnight, which is 12:00:00 AM, or 00:00:00 military time (HH24 time format mask). The WHERE clause retrieves only rows where the START_DATE_TIME column has values on or after '04-MAY-2007 12:00:00 AM'and before '05-MAY-2007 12:00:00 AM'.

You can also include the time in your WHERE clause, as in the following example. It is irrelevant whether you choose AM or PM in the display 'DD-MON-YYYY HH:MI:SS AM'format mask for the display of the result, but obviously not in the WHERE clause with the actual date string listed as '04-MAY-2007 12:00:00 AM' and '04-MAY-2007 11:59:59 PM'.

```
SELECT course_no, section_id,
 TO_CHAR(start_date_time, 'DD-
MON-YYYY HH24:MI')
  FROM section
 WHERE start_date_time >=
TO_DATE('04-MAY-2007 12:00:00
AM',
 'DD-MON-YYYY HH:MI:SS AM')
 AND start_date_time <=
TO_DATE('04-MAY-2007 11:59:59
PM',
'DD-MON-YYYY HH:MI:SS AM')
```

The next SQL query returns the same result when the following WHERE clause is used instead. Here, note that Oracle has to perform the implicit conversion of the text literal into a DATE data type.

```
WHERE start_date_time >= '04-
MAY-2007'
```

```
  AND start_date_time < '05-
MAY-2007'
```

The next WHERE clause returns the same result again, but Oracle has to perform the implicit conversion and choose the correct century.

```
WHERE start_date_time >= '04-
MAY-07'
  AND start_date_time < '05-
MAY-07'
```

You can use the TRUNC function to ignore the timestamp.

```
 SELECT course_no, section_id,
    TO_CHAR(start_date_time, 'DD-
MON-YYYY HH24:MI')
  FROM section
 WHERE TRUNC(start_date_time) =
TO_DATE('04-MAY-2007', 'DD-MON-
YYYY')
```

The next WHERE clause is another valid alternative; however, the previous WHERE clause is preferable because it explicitly specifies the TO_DATE data type conversion, together with the format mask, and includes the four-digit year.

```
WHERE TRUNC(start_date_time) =
‘04-MAY-07’
```

When you modify a database column with a function in the WHERE clause, such as the TRUNC function on the database column START_DATE_TIME, you cannot take advantage of an index if one exists on the column, unless it is a function-based index. Indexes speed up the retrieval of the data; you will learn more about the performance advantages of indexes in Chapter 13, “Indexes, Sequences, and Views.”

The next statement does not return the desired rows. Only rows that have a START_DATE_TIME of midnight on May 4, 2007, qualify, and because there are no such rows, none are selected for output.

```
SELECT course_no, section_id,
   TO_CHAR(start_date_time, ‘DD-
MON-YYYY HH24:MI’)
 FROM section
```

```
WHERE start_date_time = ‘04-
MAY-07’
```

no rows selected

The ANSI format is listed in the next example. The ANSI DATE format must be specified exactly in the format YYYY-MM-DD, with the DATE keyword prefix; note that it does not have a time component.

```
SELECT course_no, section_id,
   TO_CHAR(start_date_time, ‘DD-
MON-YYYY HH24:MI’)
 FROM section
WHERE start_date_time >= DATE
‘2007-05-04’;
  AND start_date_time < DATE
‘2007-05-05’
```

Alternatively, you can apply the TRUNC function on the START_DATE_TIME column, but be aware of the possible performance impact mentioned previously.

```
SELECT course_no, section_id,
   TO_CHAR(start_date_time, ‘DD-
MON-YYYY HH24:MI’)
  FROM section
```

```
 WHERE TRUNC(start_date_time) =
DATE '2007-05-04'
```

If you want to include the time, use the ANSI TIMESTAMP keyword. The literal must be exactly in the ANSI TIMESTAMP format, defined as YYYY-MM-DD HH24:MI:SS.

```
SELECT course_no, section_id,
   TO_CHAR(start_date_time, 'DD-
MON-YYYY HH24:MI')
 FROM section
WHERE start_date_time >=
TIMESTAMP '2007-05-04 00:00:00'
  AND start_date_time < TIMESTAMP
'2007-05-05 00:00:00'
```

Error When Entering the Wrong Format

Any attempt to change the predetermined format or the use of the wrong keyword results in an error, as you see in the next example. For this query to work, the TIMESTAMP keyword must be used instead of the DATE keyword, because the literal is in the ANSI TIMESTAMP format.

```
SELECT course_no, section_id,
   TO_CHAR(start_date_time, ‘DD-
MON-YYYY HH24:MI’)
 FROM section
WHERE start_date_time >= DATE
‘2007-05-04 00:00:00’
 AND start_date_time < DATE
‘2007-05-05 00:00:00’
WHERE start_date_time >= DATE
‘2007-05-04 00:00:00’
                              *

ERROR at line 4:
ORA-01861: literal does not match
format string
```

202

203

b) Show the student records that were modified on or before January 22, 2007. Display the date a record was modified and each student’s first and last name, concatenated in one column.

ANSWER: The query compares the MODIFIED_DATE column to the text literal. The text literal may be in either the Oracle default format or, better yet, formatted with the TO_DATE function and the appropriate four-digit year format model.

```
SELECT first_name||' '||last_name fullname,
       TO_CHAR(modified_date, 'DD-MON-YYYY HH:MI P.M.')
       "Modified Date and Time"
  FROM student
 WHERE modified_date < TO_DATE('01/23/2007','MM/DD/YYYY')
FULLNAME                  Modified Date and Time
------------------------- ---------------------
Fred Crocitto             22-JAN-2007 12:00 A.M.
J. Landry                 22-JAN-2007 12:00 A.M.
...
Judy Sethi                22-JAN-2007 12:00 A.M.
Larry Walter              22-JAN-2007 12:00 A.M.

8 rows selected.
```

As previously mentioned, it is best practice to explicitly use the TO_DATE function to convert the text literal into a DATE data type. It does not really matter which format mask you use (in this case, MM/DD/YYYY was used in the WHERE clause), as long as the date literal agrees with the format mask. This allows Oracle to interpret the passed date correctly. Be sure to include the century to avoid ambiguities.

Another possible solution is the following WHERE clause, which uses the TRUNC function: WHERE TRUNC(modified_date) <= TO_DATE(‘01/22/2007’, ‘MM/DD/YYYY’)

c) Display the course number, section ID, and starting date and time for sections that start on Sundays.

ANSWER: The SQL statement shows all the sections that start on Sunday by using the DY format mask, which displays the abbreviated day of the week, in uppercase letters.

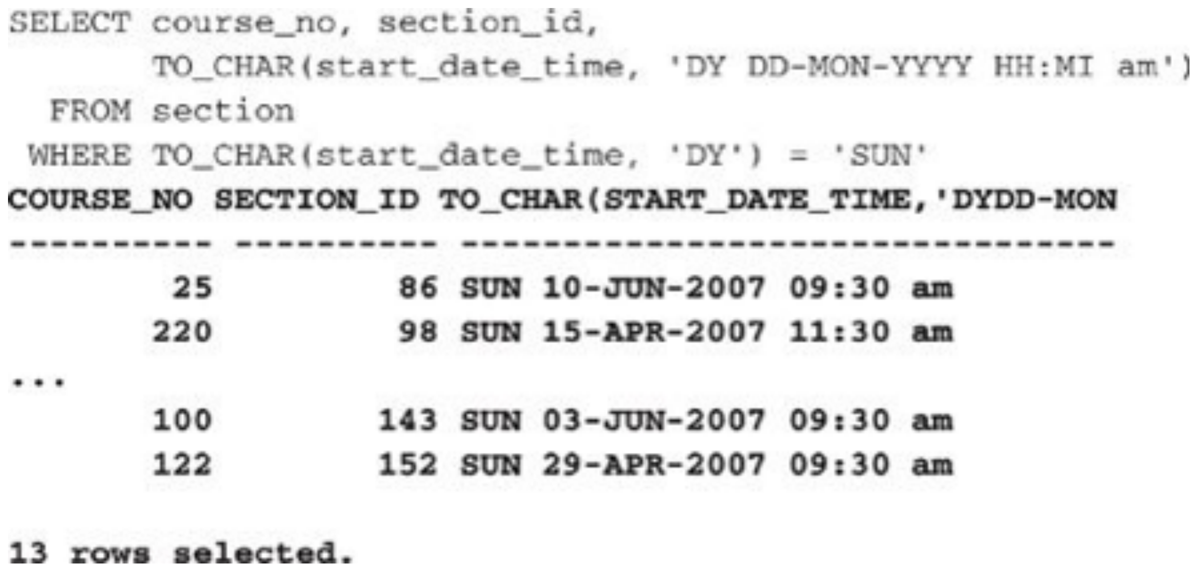
```
SELECT course_no, section_id,
       TO_CHAR(start_date_time, 'DY DD-MON-YYYY HH:MI am')
  FROM section
 WHERE TO_CHAR(start_date_time, 'DY') = 'SUN'
COURSE_NO SECTION_ID TO_CHAR(START_DATE_TIME,'DYDD-MON
---------- ---------- ------------------------------
        25         86 SUN 10-JUN-2007 09:30 am
       220         98 SUN 15-APR-2007 11:30 am
...
       100        143 SUN 03-JUN-2007 09:30 am
       122        152 SUN 29-APR-2007 09:30 am

13 rows selected.
```

THE FILL MODE

Some of the format masks are tricky. For example, if you choose the 'Day' format mask, you must specify the correct case and add the extra blanks to fill it up to a total length of nine characters. The following query returns no rows.

SELECT	**course_no, section_id,**
	TO_CHAR(start_date_time, 'Day DD-Mon-YYYY HH:MI am')
FROM	section
WHERE	TO_CHAR(start_date_time, 'Day') = 'Sunday'
no rows selected	

You can use the *fill mode (fm)* with the format mask to suppress the extra blanks.



```
SELECT course_no, section_id,
       TO_CHAR(start_date_time, 'DY DD-MON-YYYY HH:MI am')
  FROM section
 WHERE TO_CHAR(start_date_time, 'DY') = 'SUN'
COURSE_NO SECTION_ID TO_CHAR(START_DATE_TIME,'DYDD-MON
---------- ---------- ------------------------------
        25         86 SUN 10-JUN-2007 09:30 am
       220         98 SUN 15-APR-2007 11:30 am
...
       100        143 SUN 03-JUN-2007 09:30 am
       122        152 SUN 29-APR-2007 09:30 am

13 rows selected.
```

d) List the section ID and starting date and time for all sections that begin and end in July 2007.

ANSWER: In SQL, there are often different solutions that deliver the same result set. Examine the various correct solutions and avoid the pitfalls.

SOLUTION 1

This first solution takes the time into consideration. It retrieves rows that start on July 1, 2007, at midnight or thereafter (>=); the AND condition identifies the rows that have a START_DATE_TIME prior to August 1, 2007.

```
 SELECT section_id,
        TO_CHAR(start_date_time, 'DD-MON-YYYY HH24:MI:SS')
   FROM section
  WHERE start_date_time >= TO_DATE('07/01/2007', 'MM/DD/YYYY')
    AND start_date_time <  TO_DATE('08/01/2007', 'MM/DD/YYYY')
SECTION_ID TO_CHAR(START_DATE_T
---------- --------------------
        81 24-JUL-2007 09:30:00
        85 14-JUL-2007 10:30:00
...
       147 24-JUL-2007 09:30:00
       153 24-JUL-2007 09:30:00

14 rows selected.
```

The following query will *not* yield the correct result if you have a section that starts on July 31, 2007, any time after midnight. The TO_DATE function converts the string to a DATE data type and sets the timestamp to 12:00:00 A.M. Therefore, a section starting on July 31, 2007, at 18:00 is not considered part of the range.

```
SELECT section_id,
       TO_CHAR(start_date_time, ‘DD-
       MON-YYYY HH24:MI:SS’)
FROM   section
WHERE  start_date_time BETWEEN
       TO_DATE(‘07/01/2007’, ‘MM/DD/
       YYYY’)
AND    TO_DATE(‘07/31/2007’, ‘MM/DD/
       YYYY’)
```

SOLUTION 2

This solution includes the 24-hour time format mask.

```
SELECT section_id,
       TO_CHAR(start_date_time, ‘DD-
       MON-YYYY HH24:MI:SS’)
FROM   section
WHERE  start_date_time BETWEEN
       TO_DATE(‘07/01/2007’, ‘MM/DD/
       YYYY’)
AND    TO_DATE(‘07/31/2007
       23:59:59’, ‘MM/DD/YYYY
       HH24:MI:SS’)
```

This WHERE clause can also be used to obtain the desired output: The query completely ignores the time on the column START_DATE_TIME.

```
WHERE TRUNC(start_date_time) BETWEEN
      TO_DATE('07/01/2007', 'MM/DD/
      YYYY')
AND   TO_DATE('07/31/2007', 'MM/DD/
      YYYY')
```

The following WHERE clause also returns the correct result if your NLS_DATE_FORMAT parameter is set to DD-MON-RR. It is best not to rely on Oracle's implicit conversion but to specify the conversion function together with the four-digit year.

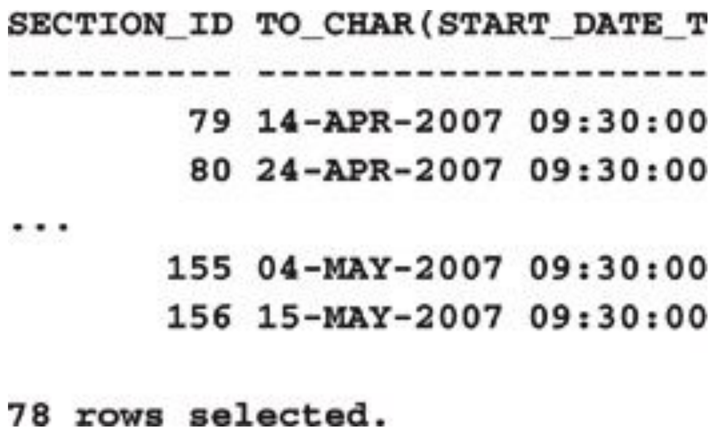
```
WHERE TRUNC(start_date_time)
BETWEEN '1-JUL-07' AND '31-JUL-07'
```

When you compare dates, be sure to think about the time.

APPLYING THE WRONG DATA TYPE CONVERSION FUNCTION

Another common source of errors when using dates is applying the wrong data type conversion function, as illustrated in the following example.

```
SELECT section_id,
       TO_CHAR(start_date_time, 'DD-
       MON-YYYY HH24:MI:SS')
FROM   section
WHERE  TO_CHAR(start_date_time, 'DD-
       MON-YYYY HH24:MI:SS')
       >= '01-JUL-2007 00:00:00'
AND    TO_CHAR(start_date_time, 'DD-
       MON-YYYY HH24:MI:SS')
       <= '31-JUL-2007 23:59:59'
```



```
SECTION_ID TO_CHAR(START_DATE_T
---------- --------------------
        79 14-APR-2007 09:30:00
        80 24-APR-2007 09:30:00
...
       155 04-MAY-2007 09:30:00
       156 15-MAY-2007 09:30:00

78 rows selected.
```

The column START_DATE_TIME is converted to a character column in the WHERE clause and then compared to the text literal. The problem is that the dates are no longer compared. Instead, the character representation of the text literal and the character representation of the contents in column START_DATE_TIME in the format 'DD-MON-YYYY HH24:MI:SS' are compared.

A column value, such as '14-APR-2007 09:30:00' is inclusive of the text literals '01-JUL-2007 00:00:00' and '31-JUL-2007 23:59:59' because the first digit of the column value 1 falls within the range of the characters 0 and 3. Therefore, the condition is true, but we know that April 14, 2007, is not in this date range.

Let's briefly discuss character comparison semantics. The next hypothetical examples further illustrate the effects of character comparisons. The query checks whether the text literal '9' is between the text literals '01' and '31', evaluated by the first digit, 0 and 3, and it returns no row, which indicates that '9'does not fall in this range.

```
SELECT *
  FROM dual
 WHERE '9' BETWEEN '01' AND '31'
no rows selected
```

With this knowledge, you can try the text literals. As you can see, the comparison of text literals used in the query with the wrong data type makes this condition true—but not if you compared the DATE data type values, because April 14, 2007, does not fall in the month of July 2007.

```
SELECT *
  FROM dual
 WHERE '14-APR-2007 09:30:00' BETWEEN '01-JUL-2007 00:00:00'
                                  AND '31-JUL-2007 23:59:59'
D
-
X

1 row selected.
```

Be sure to choose the correct data type conversion function in your WHERE clause.

Remember to make sure your data type conversion does not cause incorrect results! 206

207

TO_CHAR FUNCTION VERSUS TO_DATE FUNCTION

The TO_DATE function converts text to the DATE data type, typically used in the WHERE clause of a SELECT statement. The TO_CHAR function converts a DATE data type to text, typically used in the SELECT clause to format the results. You can also use TO_CHAR to query for specifics in a format mask. For example, to find which sections meet on Sunday, you use the

TO_CHAR function in the WHERE clause, as shown in the answer to Exercise c and listed here once again.

```
SELECT course_no, section_id,
       TO_CHAR(start_date_time, ‘Day DD-
       Mon-YYYY HH:MI am’)
FROM   section
WHERE  TO_CHAR(start_date_time, ‘fmDay’) =
       ‘Sunday’
```

e) Determine the day of the week for December 31, 1899.

ANSWER: The day of the week is Sunday.

You need to nest conversion functions by using the TO_DATE function to convert the text literal to a DATE data type and then use the TO_CHAR function to display the day of the week.

First, you translate the text literal ‘31-DEC-1899’, using the format mask ‘DD-MON-YYYY' into the Oracle DATE data type. Then apply the TO_CHAR formatting function to convert the date into any format you want—in this case, to show the day of the week.

```
SELECT TO_CHAR(TO_DATE(‘31-
DEC-1899’, ‘DD-MON-YYYY’), ‘Dy’)
```

```
FROM dual
TO_
--
Sun
1 row selected.
```

f) Execute the following statement. Write the question to obtain the desired result. Pay particular attention to the ORDER BY clause.

```
SELECT ‘Section ’||section_id||
       ‘begins on ’||
       TO_CHAR(start_date_time,
       ‘fmDay’)||‘.’ AS “Start”
FROM   section
WHERE  section_id IN (146, 127,
       121, 155, 110, 85, 148)
ORDER  BY
       TO_CHAR(start_date_time,
       ‘D’)
```

ANSWER: Your answer may be phrased similar to the following:“Display the day of the week when the sections 146, 127, 121, 155, 110, 85, and 148 start. Order the result by the day of the week starting with Sunday.”

The result of the query will look similar to the following output. The statement uses the D format mask to order by the day of the week. This format assigns the number 1 for Sunday, 2 for Monday, and so on.

```
Start
------------------------------
Section 110 begins on Sunday.
Section 121 begins on Monday.
Section 127 begins on Tuesday.
Section 146 begins on Wednesday.
Section 148 begins on Thursday.
Section 155 begins on Friday.
Section 85 begins on Saturday.

7 rows selected.
```

Lab 5.1 Quiz

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

1) The TRUNC function on a date without a format model truncates the timestamp to 12:00:00 A.M.

_____ a) True

_____ b) False

2) Converting a text literal to a DATE format requires the use of the TO_CHAR function.

______ a) True

______ b) False

3) Which of the following is the display that results from the format mask ‘Dy' for Monday?

______ a) MON

______ b) Monday

______ c) MONDAY

______ d) Mon

4) Which format mask displays December 31st, 1999?

______ a) DD-MON-YYYY

______ b) MONTH DDth, YYYY

______ c) fmMONTH DD, YYYY

______ d) Month fmDD, YYYY

______ e) fmMonth ddth, yyyy

5) The SQL query displays the distinct hours and minutes from the SECTION table’s START_DATE_TIME column.

```
SELECT DISTINCT
TO_CHAR(start_date_time,
'HH24:MM')
  FROM section
```

_____ a) True

_____ b) False

ANSWERS APPEAR IN APPENDIX A.

LAB 5.2 Performing Date and Time Math

LAB OBJECTIVES

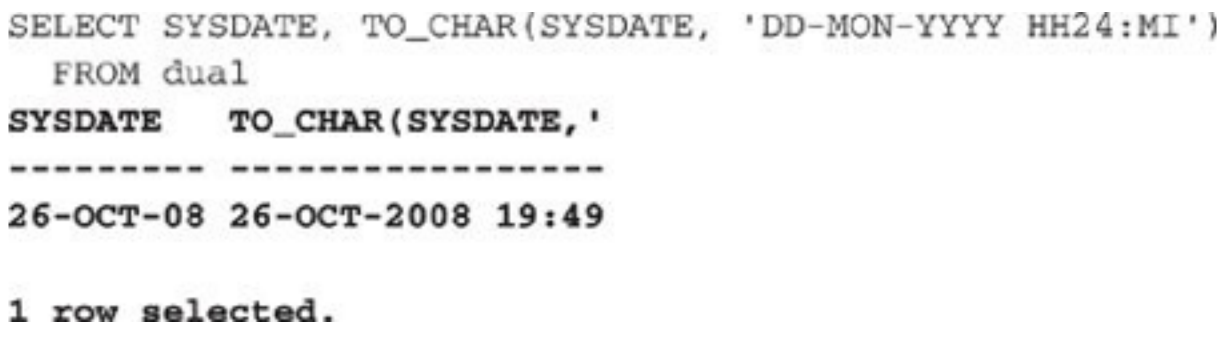
After this lab, you will be able to:

- Understand the SYSDATE Function
- Perform Date Arithmetic

You will frequently need to determine the differences between two date values or calculate dates in the future or in the past. Oracle provides functionality to help you accomplish these tasks.

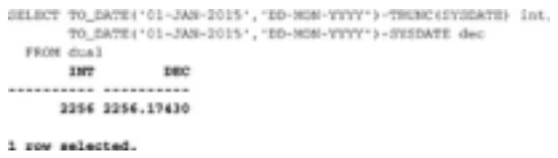
The SYSDATE Function

The SYSDATE function returns the computer operating system's current date and time and does not take any parameters. If you connect to the database server via a client machine, it returns the date and time of the machine hosting the database, not the date and time of your client machine. For example, if your client workstation is located in New York, your local time zone is *Eastern Standard Time* (EST); if you connect to a server in California, you receive the server's *Pacific Standard Time* (PST) date and time. To include the time in the result, you use the TO_CHAR function together with the appropriate format mask.



```
SELECT SYSDATE, TO_CHAR(SYSDATE, 'DD-MON-YYYY HH24:MI')
  FROM dual
SYSDATE   TO_CHAR(SYSDATE,'
--------- -----------------
26-OCT-08 26-OCT-2008 19:49

1 row selected.
```

Using the SYSDATE function, you can determine the number of days until the year 2015. The following query subtracts today's date from January 1, 2015.



```
SELECT TO_DATE('01-JAN-2015','DD-MON-YYYY')-TRUNC(SYSDATE) int,
       TO_DATE('01-JAN-2015','DD-MON-YYYY')-SYSDATE dec
  FROM dual
       INT        DEC
---------- ----------
      2256 2256.17430

1 row selected.
```

To perform any date calculation, the column or text literal must be converted to the Oracle DATE data type. For the first column, the text literal ‘01-JAN-2015’ is converted into a DATE data type, using the TO_DATE function and the corresponding format mask. Because a time is not specified, the text literal ‘01-JAN-2015’ is set to 00:00:00 military time (the equivalent of 12:00:00 A.M.). From this date, the operating system’s date (the result of the SYSDATE function) is subtracted. SYSDATE is nested inside the TRUNC function, which truncates the timestamp to 00:00:00. As a result, the column shows 2256 days.

The second column of the returned result performs the identical operation; however, this expression does not use the TRUNC function on the SYSDATE function, and therefore the time is factored into the calculation. The difference is now expressed in days, with the time in decimal format. To display the decimal in hours or minutes, you can use the NUMTODSINTERVAL function, discussed in Lab 5.4.

Performing Arithmetic on Dates

In the following example, three hours are added to the current date and time. To determine tomorrow’s date and time, simply add the number 1 to the SYSDATE function.

```
SELECT TO_CHAR(SYSDATE, 'MM/DD HH24:MI:SS') now,
       TO_CHAR(SYSDATE+3/24, 'MM/DD HH24:MI:SS')
       AS now_plus_3hrs,
       TO_CHAR(SYSDATE+1, 'MM/DD HH24:MI:SS') tomorrow,
       TO_CHAR(SYSDATE+1.5, 'MM/DD HH24:MI:SS') AS
       "36Hrs from now"
  FROM dual
NOW            NOW_PLUS_3HRS  TOMORROW       36Hrs from now
-------------- -------------- -------------- --------------
10/26 10:34:17 10/26 13:34:17 10/27 10:34:17 10/27 22:34:17

1 row selected.
```

The fraction 3/24 represents three hours; you can also express minutes as a fraction of 1440 (60 minutes × 24 hours = 1440, which is the total number of minutes in a day). For example, 15 minutes is 15/1440, or 1/96, or any equivalent fraction or decimal number.

Oracle has a number of functions to perform specific date calculations. To determine the date of the first Sunday of the year 2000, use the NEXT_DAY function, as in the following SELECT statement.

211

212

```
SELECT TO_CHAR(TO_DATE('12/31/1999','MM/DD/YYYY'),
                 'MM/DD/YYYY DY') "New Year's Eve",
       TO_CHAR(NEXT_DAY(TO_DATE('12/31/1999',
                                'MM/DD/YYYY'),
               'SUNDAY'),'MM/DD/YYYY DY')
       "First Sunday"
  FROM dual
New Year's Eve First Sunday
-------------- --------------
12/31/1999 FRI 01/02/2000 SUN

1 row selected.
```

The text string ‘12/31/1999' is first converted to a date. To determine the date of the next Sunday, the NEXT_DAY function is applied. Finally, format the output with a TO_CHAR format mask to display the result in the ‘MM/DD/YYYY DY’ format.

The ROUND Function

The ROUND function allows you to round days, months, or years. The following SQL statement lists the current date and time in the first column, using the TO_CHAR function and a format mask. The next column shows the current date and time, rounded to the next day. If the timestamp is at or past 12:00 noon and no format mask is supplied, the ROUND function rounds to the next day. The last column displays the date rounded to the nearest month, using the MM format mask.

```
SELECT TO_CHAR(SYSDATE,'DD-MON-YYYY HH24:MI') now,
       TO_CHAR(ROUND(SYSDATE),'DD-MON-YYYY HH24:MI') day,
       TO_CHAR(ROUND(SYSDATE,'MM'),'DD-MON-YYYY HH24:MI')
        mon
  FROM dual
NOW               DAY               MON
----------------- ----------------- -----------------
26-OCT-2008 10:33 26-OCT-2008 00:00 01-NOV-2008 00:00

1 row selected.
```

The EXTRACT Function

The EXTRACT function extracts the year, month, or day from a column of the DATE data type column. The next example shows rows with April values in the START_DATE_TIME column and how the various elements of the DATE data type can be extracted. Valid keyword choices are YEAR, MONTH, and DAY. You cannot extract hours, minutes, or seconds from the DATE data type. These options are available only on the other datetime-related data types you will learn about in Lab 5.3.

```
SELECT TO_CHAR(start_date_time, 'DD-MON-YYYY') "Start Date",
       EXTRACT(MONTH FROM start_date_time) "Month",
       EXTRACT(YEAR FROM start_date_time) "Year",
       EXTRACT(DAY FROM start_date_time) "Day"
  FROM section
 WHERE EXTRACT(MONTH FROM start_date_time) = 4
 ORDER BY start_date_time

Start Date      Month      Year       Day
--------------- ---------- ---------- ---
08-APR-2007              4       2007   8
09-APR-2007              4       2007   9
09-APR-2007              4       2007   9
...
29-APR-2007              4       2007  29

21 rows selected.
```

The following example of the EXTRACT function passes a text literal as the parameter, which is in ANSI DATE format.

```
SELECT EXTRACT(YEAR FROM DATE '2010-03-11') year,
       EXTRACT(MONTH FROM DATE '2010-03-11') month,
       EXTRACT(DAY FROM DATE '2010-03-11') day
  FROM dual
     YEAR     MONTH       DAY
--------- --------- ---------
     2010         3        11

1 row selected.
```

Table 5.6 summarizes some of the most frequently used DATE calculation functions and describes their purposes and respective syntax.

TABLE 5.6 Commonly Used Oracle Datetime-Related Calculation Functions

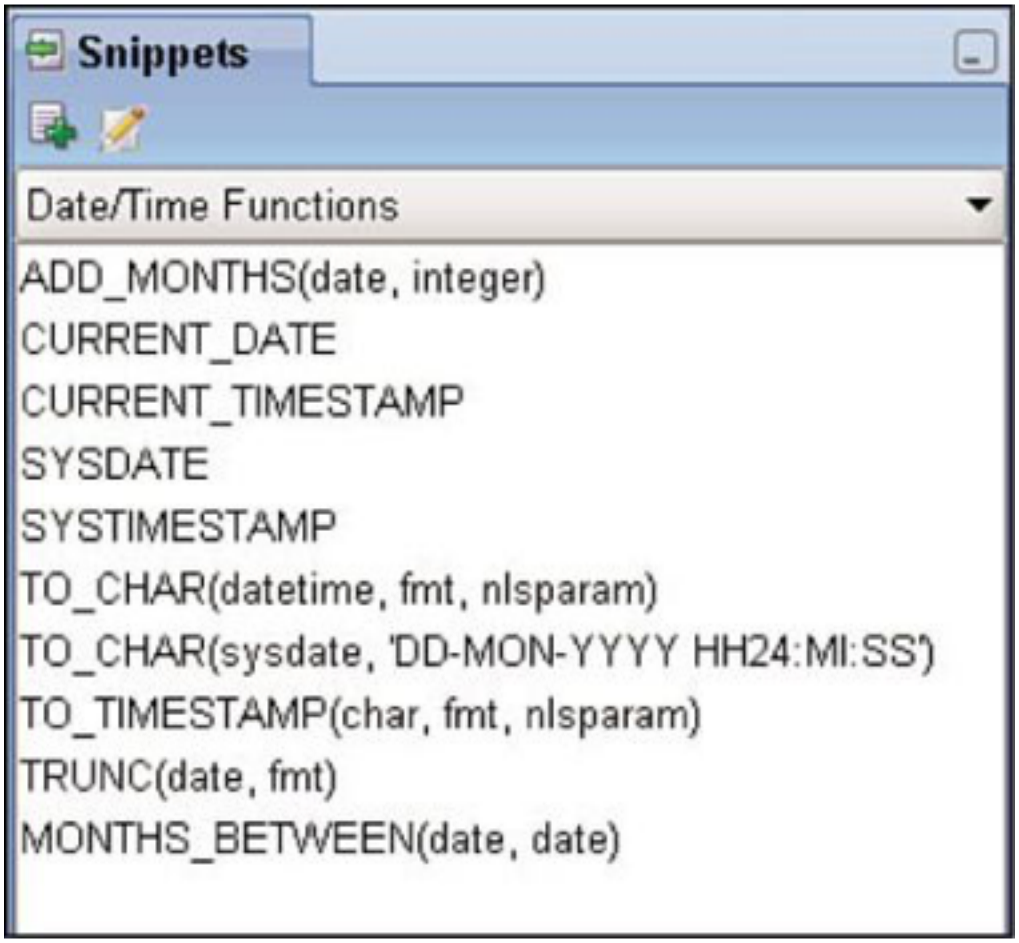
FUNCTION	PURPOSE	RETURN DATA TYPE
ADD_MONTHS(date, integer)	Adds or subtracts the number of months from a certain date.	DATE
MONTHS_BETWEEN (date2, date1)	Determines the number of months between two dates.	NUMBER
LAST_DAY(date)	Returns the last date of the month.	DATE
NEXT_DAY(date, day_of_the_week)	Returns the first day of the week that is later than the date parameter passed.	DATE
TRUNC(date)	Ignores the hours, minutes, and seconds on the DATE data type.	DATE

ROUND(date, [format_mask])	Rounds to various DATE components, depending on the optional supplied format mask.	DATE
NEW_TIME(date, current_time_zone, new_time_zone)	Returns the date and time in another time zone; for example, EST (Eastern Standard Time), PST (Pacific Standard Time), PDT (Pacific Daylight Time).	DATE
	Oracle's time zone data types, discussed in Lab 5.3, handle conversions and computations related to various time zones with much more ease than NEW_TIME.	

Snippets

As mentioned in Chapter 4, you can use the Snippets window (see Figure 5.2) in SQL Developer to drag the function into the SQL Developer window and replace the placeholder with the appropriate expression. There is another set of snippets for date formatting masks and date/time functions.

FIGURE 5.2 The Snippets window for date/time functions



ADD_MONTHS(date, integer)
CURRENT_DATE
CURRENT_TIMESTAMP
SYSDATE
SYSTIMESTAMP
TO_CHAR(datetime, fmt, nlsparam)
TO_CHAR(sysdate, 'DD-MON-YYYY HH24:MI:SS')
TO_TIMESTAMP(char, fmt, nlsparam)
TRUNC(date, fmt)
MONTHS_BETWEEN(date, date)

The Snippets window lists commonly used functions, expressions, and code fragments. It does not provide a complete list of all available functions and syntax options in Oracle SQL. You can customize the snippets and add frequently used code fragments or functions.

LAB 5.2 EXERCISES

a) Determine the number of days between February 13, 1964, and the last day of the February 1964.

b) Compute the number of months between September 29, 1999, and August 17, 2007.

c) Add three days to the current date and time.

LAB 5.2 EXERCISE ANSWERS

a) Determine the number of days between February 13, 1964, and the last day of February 1964.

ANSWER: First convert the text literal '13-FEB-1964' to a DATE data type and then use the LAST_DAY function. The date returned is

February 29, 1964, because 1964 was a leap year. The difference between the two dates is 16 days.

```
SELECT LAST_DAY(TO_DATE('13-FEB-1964','DD-MON-YYYY')) lastday,
       LAST_DAY(TO_DATE('13-FEB-1964','DD-MON-YYYY'))
       - TO_DATE('13-FEB-1964','DD-MON-YYYY') days
  FROM dual

   LASTDAY        DAYS
   --------- ---------
   29-FEB-64        16

  1 row selected.
```

214

215

The LAST_DAY function takes a single parameter and accepts only parameters of the DATE data type. Either your column must be a DATE data type column or you must convert it with the TO_DATE function.

b) Compute the number of months between September 29, 1999, and August 17, 2007.

ANSWER: The simplest solution is to use the MONTHS_BETWEEN function to determine the result.

```
SELECT MONTHS_BETWEEN(TO_DATE('17-AUG-2007','DD-MON-YYYY'),
          TO_DATE('29-SEP-1999','DD-MON-YYYY')) months
  FROM dual
          MONTHS
   --------------
   94.6129032

1 row selected.
```

The MONTHS_BETWEEN function takes two dates as its parameters and returns a numeric value.

c) Add three days to the current date and time.

ANSWER: The answer will vary, depending on when you execute this query. To add days to the current date and time, you add the number of days to the SYSDATE function.

```
SELECT TO_CHAR(SYSDATE, 'DD-MON-YYYY HH24:MI:SS') "Current",
          TO_CHAR(SYSDATE+3, 'DD-MON-YYYY HH24:MI:SS') "Answer"
  FROM dual
   Current              Answer
   -------------------- --------------------
   26-OCT-2008 23:12:02 29-OCT-2008 23:12:02

1 row selected.
```

If you have to add hours, you can express the hour as a fraction of the day. For example, five hours is SYSDATE+5/24. To find out yesterday's date, you can subtract 1 day; thus, the SELECT clause will read SYSDATE-1. You will see additional examples of computing differences between dates in Lab 5.4, which discusses the interval data types.

Lab 5.2 Quiz

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

1) You can use the ADD_MONTHS function to subtract months from a given date.

   ______ a) True

   ______ b) False

2) Which one of the following solutions adds 15 minutes to a given date?

   ______ a) SELECT SYSDATE+1/96 FROM dual

   ______ b) SELECT SYSDATE+1/128 FROM dual

   ______ c) SELECT TO_DATE(SYSDATE +1/128) FROM dual

   ______ d) SELECT TO_CHAR(SYSDATE +1/128, ‘DD-MON-YYYY 24HH:MI’) FROM dual

3) Choose the date that is calculated by the following query.

```
SELECT
TO_CHAR(NEXT_DAY(TO_DATE('02-
JAN-2000 SUN',
   'DD-MON-YYYY DY'), 'SUN'),
   'fmDay Month DD, YYYY')
 FROM dual
```

______ a) Sunday January 2, 2000

______ b) Monday January 3, 2000

______ c) Sunday January 9, 2000

______ d) None of the above dates

______ e) Invalid query

4) The next query gives you which of the following results?

```
SELECT ROUND(TO_DATE('2000/1/31
11:59', 'YYYY/MM/DD HH24:MI'))
  FROM dual
```

______ a) Returns an Oracle error message

______ b) 30-JAN-00

______ c) 31-JAN-00

______ d) 01-FEB-00

ANSWERS APPEAR IN APPENDIX A.

LAB 5.3 Understanding the TIMESTAMP and TIME ZONE Data Types

LAB OBJECTIVES

After this lab, you will be able to:

- Use Oracle's TIMESTAMP Data Type
- Use TIME ZONE Data Types

Oracle offers datetime-related data types, which include fractional seconds and time zones. These three data types are TIMESTAMP, TIMESTAMP WITH TIME ZONE, and TIMESTAMP WITH LOCAL TIME ZONE.

The TIMESTAMP Data Type

The TIMESTAMP data type allows you to store optional fractional seconds with a precision of up to nine digits; the default is six digits.

An example of a text literal can look like this: '14-FEB-09 08.29.01.000123 AM'. This represents the format mask 'DD-MON-RR HH.MI.SS.FF AM'. The fractional seconds are expressed with the FF format mask. To change the default precision of the fractional seconds, you add a number from 1 to 9 after the FF mask. For example, FF4 displays the fractional seconds with a four-digit precision.

Instead of using the Oracle format model, you can represent the literal with the ANSI TIMESTAMP format as follows: TIMESTAMP ‘2009-02-14 08:29:01.000123'. Again, 000123 is the fractional seconds, showing a six-digit precision.

The TIMESTAMP WITH TIME ZONE Data Type

Besides the date, time, and fractional seconds, the TIMESTAMP WITH TIME ZONE data type includes the *time zone displacement value*. The time zone displacement, also called *time zone offset value*, is expressed as the difference (in hours and minutes) between your local time and *Greenwich Mean Time* (GMT), now called *Coordinated Universal Time* (UTC). The time zone along the Prime Meridian in Greenwich, England, is commonly known as GMT, against which all 217
218 other time zones are compared. At noon Greenwich time, it is midnight at the International Date Line in the Pacific.

The time zone displacement value is shown as a positive or negative number (for example, -5:00), indicating the hours and minutes before or after UTC. Alternatively, the time zone can be expressed as a time zone region name, such as America/New_York instead of -5:00. The TIMESTAMP WITH TIME ZONE data type is useful when storing date and time information across geographic regions. Oracle stores all values of this data type in UTC.

The time zone region of the database is determined at the time of database creation. To find out the time zone value of your database, use the DBTIMEZONE function. The query may return a time zone displacement value such as -05:00, indicating that the time zone is 5 hours before UTC.

Instead of returning the offset number for the time zone displacement, you may see a region name instead. The time zone region equivalent for EST (Eastern Standard Time) and EDT (Eastern Daylight Time) is America/New_York and is listed in the V$TIMEZONE_NAMES data dictionary view, where you can find the list of valid time zone regions.

The data dictionary is a set of tables that provides information about the database. Data dictionary views are discussed in Chapter 14, "The Data Dictionary, Scripting, and Reporting."The server's time zone is determined when the database is created. It can be modified with an ALTER DATABASE statement. For more information on the CREATE and ALTER DATABASE statements, see the *Oracle Administrator's Guide* documentation.

The TIMESTAMP WITH LOCAL TIME ZONE Data Type

The TIMESTAMP WITH LOCAL TIME ZONE stores the date and time values of the database's own local time zone. When the user retrieves the data, the returned values are automatically converted to represent each individual user's time zone. In addition, the database does not store the time zone displacement value as part of the data type.

When performing arithmetic on this data type, Oracle automatically converts all values to UTC before doing the calculation and then converts the value back to the local time. This is in contrast to the TIMESTAMP WITH TIME ZONE data type, where the values are always stored in UTC, and a conversion is unnecessary.

Oracle provides automatic support for daylight saving time and for boundary cases when the time switches. 218

219

Common Format Masks

Table 5.7 shows the datetime-related data types and their individual components, together with commonly used format masks and examples of literals. Throughout this lab, you will use these data types in different exercises.

TABLE 5.7 Overview of Oracle Datetime-Related Data Types

DATA TYPE	COMPONENTS	COMMON FORMAT MASKS

DATE	Century, Year, Month, Day, Hour, Minute, Second	Oracle Format Masks:
		`‘DD-MON-RR’ and ‘DD-MON-YYYY’`
		`‘14-FEB-09’ and ‘14-FEB-2009’`

		ANSI Formats:
		`DATE ‘YYYY-MM-DD’`
		`DATE ‘2009-02-14’`
		`TIMESTAMP ‘YYYY-MM-DD HH24:MI:SS’`
		`TIMESTAMP ‘2009-02-14 16:21:04’`

TIMESTAMP	Same as DATE with additional fractional seconds	Oracle Formats: `‘DD-MON-RR HH.MI.SS.FF AM’` `‘14-FEB-09 04.21.04.000001 PM’` `‘DD-MON-YYYY HH.MI.SS.FF AM’` `‘14-FEB-2009 04.21.04.000001 PM’`
		ANSI Formats: `TIMESTAMP ‘YYYY-MM-DD HH24:MI:SS.FF’` `TIMESTAMP ‘2009-02-14 16:21:04.000001’`

TIMESTAMP WITH TIME ZONE	Same as TIMESTAMP plus Time Zone Hour and Time Zone Minute (TZH:TZM) or Time Zone Region Name (TZR)	Oracle Formats with time offset values in hours and minutes:
		`‘DD-MON-RR HH.MI.SS.FF AM TZH:TZM’`
		`‘14-FEB-09 04.21.04.000001 PM -05:00’`
		`‘DD-MON-YYYY HH.MI.SS.FF AM TZH:TZM’`
		`‘14-FEB-2009 04.21.04.000001 PM -05:00’`
		`Oracle Formats with time zone region:`
		`‘DD-MON-RR HH.MI.SS.FF AM TZR’`
		`‘14-FEB-09 04.21.04.000001 PM America/New_York’`
		`‘DD-MON-YYYY HH.MI.SS.FF AM TZR’`
		`‘14-FEB-2009`

		`14-FEB-2009 04.21.04.000001 PM America/New_York’`
		ANSI Formats with offset value: `TIMESTAMP ‘YYYY-MM-DD HH24:MI:SS.FF TZH:TZM’` `TIMESTAMP ‘2009-02-14 16:21:04.000001 -5:00’`
		ANSI Formats with time zone region: `TIMESTAMP ‘YYYY-MM-DD HH:MI:SS.FF TZR’` `TIMESTAMP ‘2009-02-14 16:21:04.000001 America/New_York’`
TIMESTAMP WITH LOCAL TIMESTAMP	Same components as the TIMESTAMP data type	See TIMESTAMP

Table 5.8 lists the valid range of values for the individual components of the datetime-related data types.

TABLE 5.8 Valid Value Ranges for Date and Time Components

DATE COMPONENT	VALID VALUES
YEAR	–4712 – 9999 (excluding year 0)
MONTH	01 – 12
DAY	01 – 31
HOUR	00 – 23
MINUTE	00 – 59
SECOND	00 – 59 (optional precision up to nine digits on TIMESTAMP, TIMESTAMP WITH TIME ZONE, and TIMESTAMP WITH LOCAL TIME ZONE data types)
TIMEZONE_HOUR	–12 – +13
TIMEZONE_MINUTE	00 – 59

Datetime Functions

Table 5.9 lists the datetime functions, which you use to determine the current date and time.

TABLE 5.9 Session and Server Datetime Functions

FUNCTION	**PURPOSE**	**RETURN DATA TYPE**
SYSDATE	Returns the database server operating system's current date and time.	DATE
CURRENT_DATE	Returns the date and time of the local *session* time zone, in the DATE data type. (The local session time can be different than the server's date and time, if the client session is in a different time zone.)	DATE
CURRENT_TIMESTAMP [(optional_precision)]	Returns the individual's *session* date and time in the data type TIMESTAMP WITH TIME ZONE value.	TIMESTAMP WITH TIME ZONE

SYSTIMESTAMP	Returns the date, time, and fractional seconds and time zone of the server. This is similar to the SYSDATE function but includes the fractional seconds and time zone.	TIMESTAMP WITH TIME ZONE
LOCALTIMESTAMP [(optional_precision)]	Returns in the TIMESTAMP format the current date and time in the local *session* time.	TIMESTAMP
SESSIONTIMEZONE	Returns the time zone offset value of the *session* time zone or the time zone region name, depending on the setup of the database.	VARCHAR2
DBTIMEZONE	Returns the time zone offset value of the database *server* time zone or time zone region name, depending on the setup of the database.	VARCHAR2

THE LOCALTIMESTAMP FUNCTION

The next SQL statement shows the use of the LOCALTIMESTAMP function, which returns the current date and time, including the fractional sections in Oracle’s TIMESTAMP format. This function considers the local user’s *session* time; that is, if the database server is in San Francisco and the user is in New York, the time displayed is the user’s local New York time.

```
SELECT LOCALTIMESTAMP
  FROM dual
LOCALTIMESTAMP
---------------------------
26-FEB-09 04.21.04.000001 PM

1 row selected.
```

THE SYSTIMESTAMP FUNCTION

Unlike the SYSDATE function, the SYSTIMESTAMP function includes fractional seconds with up to nine digits of precision. Like the SYSDATE function, it uses the *database’s* time zone, not that of the client machine executing the function. The time zone displacement, or offset, in the following SQL statement is -05.00, indicating that the time is 5 hours before the UTC, which in this example represents EST. The offset is expressed in the format mask [+|-] TZH:TZM and includes either positive or negative time zone hour and time zone minute offset numbers. 221

222

```
SELECT SYSTIMESTAMP
  FROM dual
SYSTIMESTAMP
-----------------------------------
26-FEB-09 04.21.04.000001 PM -05:00

1 row selected.
```

THE CURRENT_TIMESTAMP FUNCTION

The CURRENT_TIMESTAMP function returns the current *session's* time in the data type TIMESTAMP WITH TIME ZONE value. It differs from the LOCALTIMESTAMP function in that the data type is not TIMESTAMP but TIMESTAMP WITH TIME ZONE and therefore includes the time zone displacement value (or the actual name of the time zone, depending on the client session settings).

```
SELECT CURRENT_TIMESTAMP, LOCALTIMESTAMP
  FROM dual
CURRENT_TIMESTAMP                   LOCALTIMESTAMP
----------------------------------- ----------------------------
26-FEB-09 07.59.49.000000 PM -05:00 26-FEB-09 07.59.49.000000 PM

1 row selected.
```

THE CURRENT_DATE FUNCTION

The CURRENT_DATE function returns the date and time in the *session's* time zone. The returned values can be different than the values returned by the SYSDATE function. For example, if you execute a query on your machine located on the East Coast against a database server located on the West Coast, the SYSDATE function returns the date and time of the server in the Pacific time zone, and the CURRENT_DATE function returns your local East Coast date and time. The return data type of the CURRENT_DATE function is a DATE data type.

```
SELECT TO_CHAR(CURRENT_DATE, 'DD-MON-YYYY HH:MI:SS PM')
  FROM dual
TO_CHAR(CURRENT_DATE,'D
-----------------------
01-FEB-2009 02:37:11 AM

1 row selected.
```

You might wonder how the CURRENT_DATE function compares to the previously mentioned LOCALTIMESTAMP function. The difference is in the return data type of the function. CURRENT_DATE returns a DATE data type, and LOCALTIMESTAMP returns the TIMESTAMP data type, which includes the fractional seconds.

THE SESSIONTIMEZONE FUNCTION

Because an individual user may be in a different time zone than the database server, you can execute different functions, depending on what you want to accomplish. The SESSIONTIME ZONE function returns the session's time zone displacement value; the DBTIMEZONE function returns the server's time zone displacement value.

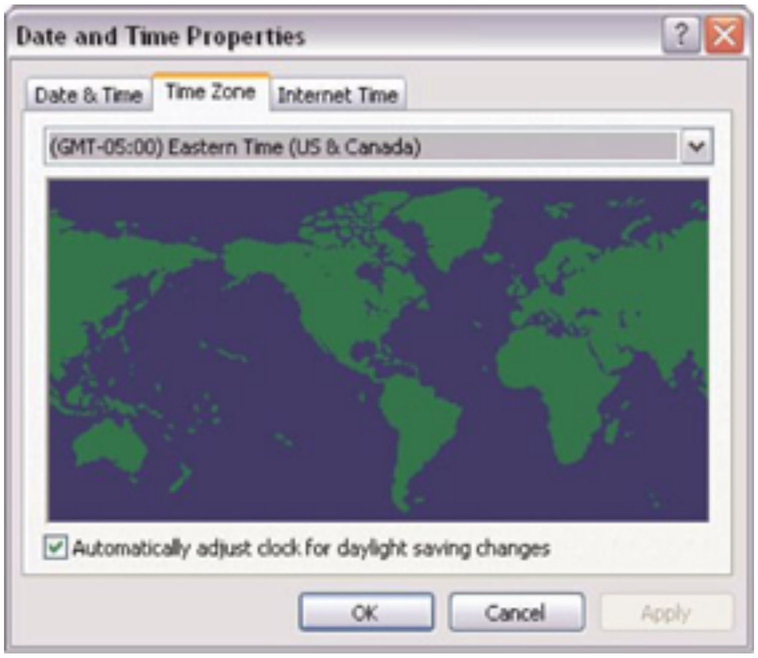
The next statement shows the execution of the SESSIONTIMEZONE function. It includes the time zone displacement value, indicating the difference in hours and minutes between UTC and your local time. (Depending on the client environment setup, you may see the time zone name instead.) The user's local time zone is determined by either the most recent ALTER SESSION statement setting indicating the local time zone or by your operating system's time zone. If none of them are valid, the default is UTC.

```
SELECT SESSIONTIMEZONE
  FROM dual
SESSIONTIMEZONE
---------------
-05:00

1 row selected.
```

CHANGING THE LOCAL TIME ZONE

You can experiment with changing the time zone of your local machine and the effect on the discussed functions. For example, on the Windows operating system, you can change the time zone in the Control Panel by choosing the Date and Time Properties (see Figure 5.3). If you change your default time zone to another time zone with a different time zone displacement value, the results of the SESSIONTIMEZONE function are different.

FIGURE 5.3 Changing the time zone on the Windows operating system


Date and Time Properties

(GMT-05:00) Eastern Time (US & Canada)

☑ Automatically adjust clock for daylight saving changes

Make sure to exit SQL Developer or log out of the current SQL*Plus session before you do this, so the effects of the time zone change are visible.

OVERRIDING THE INDIVIDUAL SESSION TIME ZONE

You can change the time zone for an individual session by using the ALTER SESSION command. The setting remains until you exit the session. The following three statements illustrate different ways you can change the time zone offset value. The first changes the value to a time zone region name, the second makes it equivalent to the database server’s time zone, and the last resets it to the session’s local time zone.

```
ALTER SESSION SET TIME_ZONE =
‘America/New_York’
ALTER SESSION SET TIME_ZONE =
dbtimezone
ALTER SESSION SET TIME_ZONE = local
```

THE DBTIMEZONE FUNCTION

The DBTIMEZONE function displays the database server's time zone displacement value; if none has been set, it displays UTC (+00:00) as the default value.

```
SELECT DBTIMEZONE
  FROM dual
DBTIME
------
-05:00

1 row selected.
```

Extract Functions

Different functions allow you to pull out various components of the datetime data types, such as YEAR, MONTH, and so on (see Table 5.10). Similar results can also be accomplished with the TO_CHAR function and the format masks discussed in Lab 5.2.

THE SYS_EXTRACT_UTC FUNCTION

The purpose of the SYS_EXTRACT_UTC function is to extract the UTC from a passed date and time value. The next example shows two equivalent date and time values when translated to UTC. Both are ANSI literals of the data type TIMESTAMP WITH TIME ZONE.

```
TIMESTAMP '2009-02-11 7:00:00
-8:00'
TIMESTAMP '2009-02-11 10:00:00
-5:00'
```

The first timestamp shows February 11, 2009, at 7:00 A.M. PST, which is 8 hours before UTC. This value is identical to the next timestamp; it shows the same date with 10:00 A.M. EST local time, which is 5 hours before UTC. The 7:00 A.M. time on the West Coast is identical to 10:00 A.M. on the East Coast as there is a three-hour time difference. When calculating the time in UTC, you see that the two timestamps are identical in 224
225 UTC. In fact, Oracle calculates the TIMESTAMP WITH TIME ZONE data type always in UTC and then displays the local time with the time zone displacement.

```
SELECT SYS_EXTRACT_UTC(TIMESTAMP '2009-02-11 7:00:00 -8:00')
          "West coast to UTC",
       SYS_EXTRACT_UTC(TIMESTAMP '2009-02-11 10:00:00 -5:00')
          "East coast to UTC"
FROM dual
West coast to UTC                East coast to UTC
-------------------------------- --------------------------------
11-FEB-09 03.00.00.000000000 PM  11-FEB-09 03.00.00.000000000 PM

1 row selected.
```

TABLE 5.10 Extract Functions

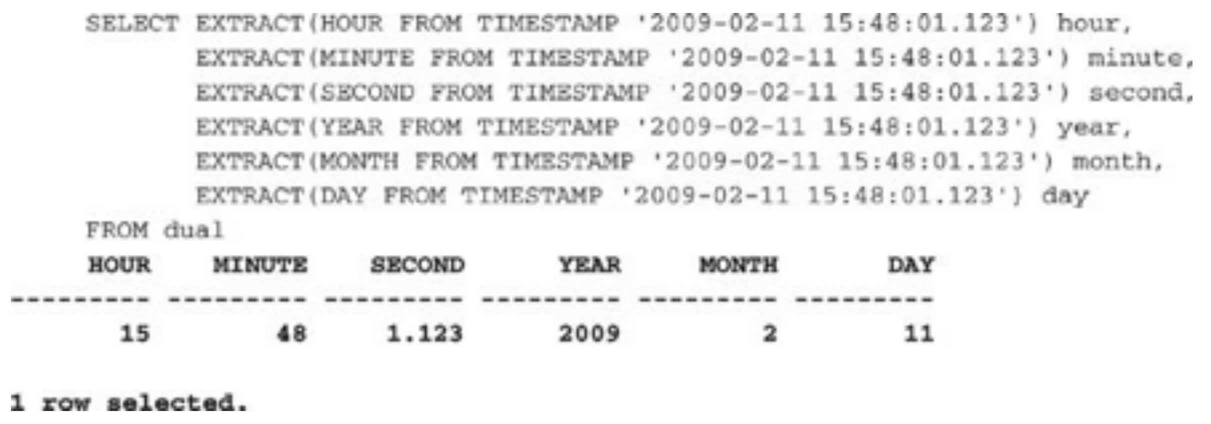
FUNCTION	PURPOSE	RETURN DATA TYPE
EXTRACT(YEAR FROM date)	Extracts year from a DATE data type. Valid keyword choices are YEAR, MONTH, and DAY to extract the year, month, and day, respectively.	NUMBER
EXTRACT(YEAR FROM timestamp)	Extracts the year from a TIMESTAMP data type. Valid keyword choices are YEAR, MONTH, DAY, HOUR, MINUTE, and SECOND to extract the year, month, day, hour, minute, and seconds, including fractional seconds, respectively.	NUMBER
EXTRACT(YEAR FROM timestamp_with_time_zone)	Valid keywords are YEAR, MONTH, DAY, HOUR, MINUTE, SECOND, TIMEZONE_HOUR, TIMEZONE_ MINUTE, TIMEZONE_REGION, TIMEZONE_ABBR. The values are returned in UTC.	NUMBER for TIMEZONE_REGION (If TIMEZONE_ABBR is passed, the EXTRACT function returns VARCHAR2.)
SYS_EXTRACT_UTC (timestamp_with_time zone)	Returns the date and time in UTC	TIMESTAMP WITH TIME ZONE

TZ_OFFSET(time_zone)	Returns the time difference between UTC and passed time zone value	VARCHAR2

THE EXTRACT FUNCTION AND THE TIMESTAMP DATA TYPE

The following SQL statement extracts the various components of the TIMESTAMP data type, including the seconds. You cannot extract the fractional seconds only; they are included as part of the SECOND keyword specification. The passed TIMESTAMP literal is in the ANSI TIMESTAMP default format. 225

226



```
SELECT EXTRACT(HOUR FROM TIMESTAMP '2009-02-11 15:48:01.123') hour,
       EXTRACT(MINUTE FROM TIMESTAMP '2009-02-11 15:48:01.123') minute,
       EXTRACT(SECOND FROM TIMESTAMP '2009-02-11 15:48:01.123') second,
       EXTRACT(YEAR FROM TIMESTAMP '2009-02-11 15:48:01.123') year,
       EXTRACT(MONTH FROM TIMESTAMP '2009-02-11 15:48:01.123') month,
       EXTRACT(DAY FROM TIMESTAMP '2009-02-11 15:48:01.123') day
  FROM dual
     HOUR    MINUTE    SECOND      YEAR     MONTH       DAY
--------- --------- --------- --------- --------- ---------
       15        48     1.123      2009         2        11

1 row selected.
```

THE EXTRACT FUNCTION AND THE TIMESTAMP WITH TIME ZONE DATA TYPE

Following are examples of the EXTRACT function that illustrate how to pull out the various components of the

TIMESTAMP WITH TIME ZONE data type. Note here that when using EXTRACT on this data type, all date and time values are returned in UTC, not the time displayed by default in the column.

The next example shows just a few of the components. When examining the result, you see that the column labeled HOUR displays the time as 21, which is 9 P.M., but the actual local time is stored as 4 P.M. in the column COL_TIMESTAMP_W_TZ. This is a clear indication that the EXTRACT function uses UTC.

```
SELECT col_timestamp_w_tz,
     EXTRACT(YEAR FROM col_timestamp_w_tz) year,
     EXTRACT(MONTH FROM col_timestamp_w_tz) month,
     EXTRACT(DAY FROM col_timestamp_w_tz) day,
     EXTRACT(HOUR FROM col_timestamp_w_tz) hour,
     EXTRACT(MINUTE FROM col_timestamp_w_tz) min,
     EXTRACT(SECOND FROM col_timestamp_w_tz) sec
  FROM date_example
COL_TIMESTAMP_W_TZ                  YEAR MONTH  DAY HOUR MIN SEC
----------------------------------- ----- ------ --- ---- --- ---
24-FEB-09 04.25.32.000000 PM -05:00  2009     2   24   21  25  32

1 row selected.
```

The keywords TIMEZONE_HOUR and TIMEZONE_MINUTE allow you to display the time zone displacement value expressed in hours and minutes. The TIMEZONE_REGION and TIMEZONE_ABBR keywords indicate the time zone region information spelled out or in abbreviated format.

If a region has not been set up for your database or results in ambiguity, you see the value UNKNOWN, as in this example.

```
SELECT col_timestamp_w_tz,
       EXTRACT(TIMEZONE_HOUR FROM col_timestamp_w_tz) tz_hour,
       EXTRACT(TIMEZONE_MINUTE FROM col_timestamp_w_tz) tz_min,
       EXTRACT(TIMEZONE_REGION FROM col_timestamp_w_tz) tz_region,
       EXTRACT(TIMEZONE_ABBR FROM col_timestamp_w_tz) tz_abbr
  FROM date_example
COL_TIMESTAMP_W_TZ                   TZ_HOUR TZ_MIN TZ_REGION TZ_ABBR
------------------------------------ ------- ------ --------- -------
24-FEB-09 04.25.32.000000 PM -05:00       -5      0 UNKNOWN   UNK

1 row selected.
```

THE DATE_EXAMPLE TABLE

In the two previous SQL statements, you may have noticed the use of a table called DATE_ EXAMPLE to illustrate the different date variants. This table is not part of the STUDENT schema but can be created based on the additional script, available for download from the companion Web site, located at www.oraclesqlbyexample.com. Listed here are the columns of the DATE_EXAMPLE table and their respective data types.

```
 SQL> DESCRIBE date_example
Name                          Null?    Type
----------------------------- -------- ------------------------------
COL_DATE                               DATE
COL_TIMESTAMP                          TIMESTAMP(6)
COL_TIMESTAMP_W_TZ                     TIMESTAMP(6) WITH TIME ZONE
COL_TIMESTAMP_W_LOCAL_TZ               TIMESTAMP(6) WITH LOCAL TIME
```

The first column, named COL_DATE, is of the familiar DATE data type. The second column, called COL_TIMESTAMP, includes fractional seconds, with a six-digit default precision. The third column, called COL_TIMESTAMP_W_TZ, additionally contains the time zone offset. Finally, the fourth column is defined as the TIMESTAMP WITH LOCAL TIME ZONE data type.

Conversion Functions

To query against a datetime column using a text literal, you need to list the literal with the appropriate format mask (refer to Table 5.7 for frequently used format masks). Table 5.11 lists the various functions to convert to the desired data type. In the previous labs, you became familiar with the TO_CHAR and the TO_DATE functions. TO_TIMESTAMP and TO_TIMESTAMP_TZ work in a similar way. The TO_CHAR and TO_DATE functions are listed for completeness.

TABLE 5.11 Datetime-Related Conversion Functions

FUNCTION	PURPOSE	RETURN DATA TYPE
TO_TIMESTAMP(char [,format_mask])	Converts text to the TIMESTAMP data type, based on format_mask. (This works similarly to the TO_DATE function.)	TIMESTAMP
TO_TIMESTAMP_TZ(char [,format_mask])	Converts text or a database column of VARCHAR2 or CHAR data type to a TIMESTAMP WITH TIME ZONE data type, based on a format_mask.	TIMESTAMP WITH TIME ZONE

TO_DATE(char [,format_mask])	Converts text to a DATE data type. As with all other datetime-related conversion functions, format_mask is optional if the value conforms to the NLS DATE_FORMAT; otherwise, format_mask must be specified.	DATE
TO_CHAR(date [,format_mask])	Converts all datetime-related data types into VARCHAR2 to display it in a different format than the default date format. (The TO_CHAR function can be used with other data types; see Lab 5.5.)	VARCHAR2

FROM_TZ(timestamp, hour_min_offset)	Converts a TIMESTAMP value into a TIMESTAMP WITH TIME ZONE data type. An example of the hour_min_offset value (time zone displacement value) is '-5:00', or it can be a time zone region name, such as 'America/ New_York'.	TIMESTAMP WITH TIME
CAST	Converts TIMESTAMP, TIMESTAMP WITH TIME ZONE, and TIMESTAMP WITH LOCAL TIME ZONE (see Lab 5.5 regarding the CAST function).	Various

The next statement queries the DATE_EXAMPLE table and converts the text literal into a TIMESTAMP WITH TIME ZONE data type to be able to compare the value against the column COL_TIMESTAMP_W_TZ of the same data type.

```
SELECT col_timestamp_w_tz
  FROM date_example
 WHERE col_timestamp_w_tz = TO_TIMESTAMP_TZ
                ('24-FEB-09 04.25.32.000000 PM -05:00',
                 'DD-MON-RR HH.MI.SS.FF AM TZH:TZM')
COL_TIMESTAMP_W_TZ
-----------------------------------
24-FEB-09 04.25.32.000000 PM -05:00

1 row selected.
```

Datetime Expression

A datetime expression can be a column of data type TIMESTAMP WITH TIME ZONE, TIMESTAMP WITH LOCAL TIME ZONE, or TIMESTAMP or an expression that results in any of the three data types.

The expression can be shown in various time zones with the keywords AT TIME ZONE. The next example illustrates the value 24-FEB-07 04.25.32.000000 P.M. –05:00 in the COL_TIMESTAMP_W_TZ column, displayed in the Los Angeles local time instead. The expression uses the time zone region name ‘America/ Los_Angeles' after the keywords AT TIME ZONE.

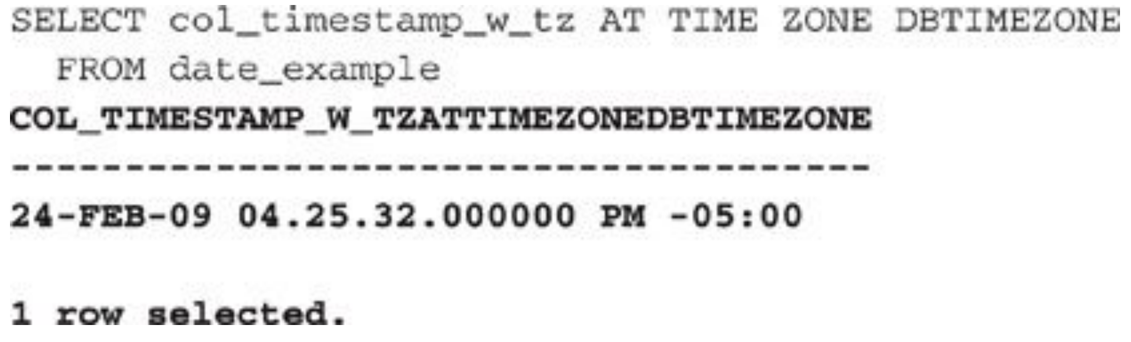
```
SELECT col_timestamp_w_tz AT TIME ZONE 'America/Los_Angeles'
  FROM date_example
COL_TIMESTAMP_W_TZATTIMEZONE'AMERICA/LOS_ANGELES
-----------------------------------------------
24-FEB-09 01.25.32.000000 PM AMERICA/LOS_ANGELES

1 row selected.
```

The syntax of the datetime expression is as follows.

```
datetime_value_expr AT {
 LOCAL |
 TIME ZONE{'[+|-]hh:mm' |
   DBTIMEZONE |
   SESSIONTIMEZONE |
   'time_zone_name'}}
```

Besides showing the time in the local time zone, you can also choose a specific time zone displacement in the TZH:TZM format. Other syntax alternatives are DBTIMEZONE, which returns the value in the database server's time zone, and SESSIONTIMEZONE, which shows the session's time zone and the time zone name for a time zone region name.

The next example displays the same column expressed in the database server's time zone with the DBTIMEZONE keyword.



```
SELECT col_timestamp_w_tz AT TIME ZONE DBTIMEZONE
  FROM date_example
COL_TIMESTAMP_W_TZATTIMEZONEDBTIMEZONE
-------------------------------------
24-FEB-09 04.25.32.000000 PM -05:00

1 row selected.
```

Compared to the NEW_TIME function mentioned in Lab 5.2, the datetime expression is more versatile because it allows a greater number of time zone values.

LAB 5.3 EXERCISES

a) Describe the default display formats of the result returned by the following SQL query.

```
SELECT col_date, col_timestamp, col_timestamp_w_tz
  FROM date_example
COL_DATE  COL_TIMESTAMP                COL_TIMESTAMP_W_TZ
--------- ---------------------------- ------------------
24-FEB-09 24-FEB-09 04.25.32.000000 PM 24-FEB-09 04.25.32.000000 PM -
➥05:00

1 row selected.
```

b) Explain the result of the following SELECT statement. Are there alternate ways to rewrite the query's WHERE clause?

```
SELECT col_timestamp
  FROM date_example
 WHERE col_timestamp = '24-FEB-09 04.25.32.000000 PM'
COL_TIMESTAMP
----------------------------
24-FEB-09 04.25.32.000000 PM

1 row selected.
```

c) What function can you utilize to display the seconds component of a TIMESTAMP data type column?

d) What do you observe about the text literal of the following query's WHERE clause?

```
SELECT col_timestamp_w_tz
  FROM date_example
 WHERE col_timestamp_w_tz = '24-FEB-09 04.25.32.000000 PM -05:00'
COL_TIMESTAMP_W_TZ
-----------------------------------
24-FEB-09 04.25.32.000000 PM -05:00

1 row selected.
```

e) The following SQL statements are issued against the database server. Explain the results.

```
SELECT SESSIONTIMEZONE
  FROM dual
SESSIONTIMEZONE
---------------
-05:00

1 row selected.

SELECT col_timestamp_w_tz, col_timestamp_w_local_tz
  FROM date_example
  COL_TIMESTAMP_W_TZ                  COL_TIMESTAMP_W_LOCAL_TZ
  ----------------------------------- ---------------------------
  24-FEB-09 04.25.32.000000 PM -05:00 24-FEB-09 04.25.32.000000 PM

  1 row selected.

  ALTER SESSION SET TIME_ZONE = '-8:00'
  Session altered.

  SELECT col_timestamp_w_tz, col_timestamp_w_local_tz
    FROM date_example
  COL_TIMESTAMP_W_TZ                  COL_TIMESTAMP_W_LOCAL_TZ
  ----------------------------------- ---------------------------
  24-FEB-09 04.25.32.000000 PM -05:00 24-FEB-09 01.25.32.000000 PM

  1 row selected.

  ALTER SESSION SET TIME_ZONE = '-5:00'
  Session altered.
```

LAB 5.3 EXERCISE ANSWERS

a) Describe the default display formats of the result returned by the following SQL query.

```
SELECT col_date, col_timestamp, col_timestamp_w_tz
  FROM date_example
COL_DATE  COL_TIMESTAMP                COL_TIMESTAMP_W_TZ
--------- ---------------------------- ----------------------------
24-FEB-09 24-FEB-09 04.25.32.000000 PM 24-FEB-09 04.25.32.000000 PM -
➥05:00

1 row selected.
```

ANSWER: This query returns the values of three columns: COL_DATE, COL_TIMESTAMP, and COL_TIMESTAMP_W_TZ.

You are already familiar with the DD-MON-RR DATE default format listed in the right column. The display format for the Oracle TIMESTAMP data type is DD-MON-RR HH.MI.SS.FF AM, as shown in the second column. The third column, named COL_TIMESTAMP_W_TZ, also shows the time zone displacement value, in the format +/- TZH:TZM. (All the default display formats can be changed with the NLS_TIMESTAMP_FORMAT and NLS_TIMESTAMP_TZ_FORMAT parameters in the Oracle database initialization file. You can also modify within SQL Developer or using an

ALTER SESSION statement. An ALTER SESSION statement changes certain values for the user's current session. These temporary settings remain until the user disconnects the session—that is, exits the program that created the session.)

b) Explain the result of the following SELECT statement. Are there alternate ways to rewrite the query's WHERE clause?

```
SELECT col_timestamp
  FROM date_example
 WHERE col_timestamp = '24-FEB-09 04.25.32.000000 PM'
COL_TIMESTAMP
---------------------------
24-FEB-09 04.25.32.000000 PM

1 row selected.
```

ANSWER: The query shows the use of the TIMESTAMP data type. There are alternative ways to achieve the same result. As you learned previously, with the DATE data type, it is always preferable to explicitly perform the data type conversion instead of use the default text literal. The following query uses the TO_TIMESTAMP function to convert the text literal into an Oracle TIMESTAMP data type, and it uses the matching format masks. The FF format mask represents the fractional seconds; the AM format mask indicates the time listed in the AM/PM format, not the 24-hour military time format.

```
SELECT col_timestamp
FROM   date_example
WHERE  col_timestamp =
       TO_TIMESTAMP('24-FEB-2009
       04:25:32.000000 PM', 'DD-
       MON-YYYY HH:MI:SS.FF AM')
```

If you exclude the fractional seconds together with the FF format mask, the fractional seconds are implied to be zero, as you can see in the next example.

```
SELECT col_timestamp
FROM   date_example
WHERE  col_timestamp =
       TO_TIMESTAMP('24-FEB-2009
       04:25:32 PM', 'DD-MON-YYYY
       HH:MI:SS AM')
```

The following query, using the ANSI TIMESTAMP format, also returns the correct result.

```
SELECT col_timestamp
FROM   date_example
WHERE  col_timestamp =
       TIMESTAMP
       '2009-02-24
       16:25:32.000000'
```

CONVERSION BETWEEN ORACLE DATE DATA TYPES

You might wonder whether you can apply any of the previously used TO_DATE format models to query the COL_TIMESTAMP column. The next SQL statement converts the text literal to a DATE data type with the TO_DATE function. The DATE data type is implicitly converted to the TIMESTAMP data type. Because the fractional seconds in this example are equal to 000000, the result is considered equivalent, and the row is returned.

```
SELECT col_timestamp
FROM   date_example
WHERE  col_timestamp =
       TO_DATE('24-FEB-2009
       04:25:32 PM', 'DD-
       MON-YYYY HH:MI:SS
       AM')
```

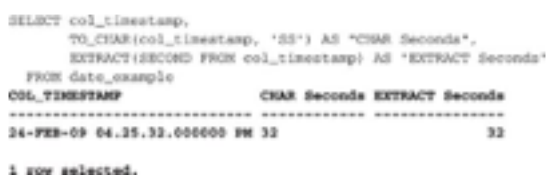
The following SQL statement shows what happens when you apply a TO_TIMESTAMP function to a DATE data type column. The TO_TIMESTAMP function sets the time portion of the DATE column to midnight.

```
SELECT col_timestamp,
       TO_CHAR(col_timestamp, 'SS') AS "CHAR Seconds",
       EXTRACT(SECOND FROM col_timestamp) AS "EXTRACT Seconds"
  FROM date_example
COL_TIMESTAMP                CHAR Seconds EXTRACT Seconds
---------------------------- ------------ ---------------
24-FEB-09 04.25.32.000000 PM 32                        32

1 row selected.
```

c) What function can you utilize to display the seconds component of a TIMESTAMP data type column?

ANSWER: You can use either the TO_CHAR function or the EXTRACT function to display components such as the year, month, date, hour, minute, and seconds from the TIMESTAMP data type columns.

The next SQL statement shows how they are used and their respective differences.



```
SELECT col_timestamp,
       TO_CHAR(col_timestamp, 'SS') AS "CHAR Seconds",
       EXTRACT(SECOND FROM col_timestamp) AS "EXTRACT Seconds"
  FROM date_example
COL_TIMESTAMP                CHAR Seconds EXTRACT Seconds
---------------------------- ------------ ---------------
24-FEB-09 04.25.32.000000 PM 32                        32

1 row selected.
```

The first column displays the column's value, in the default TIMESTAMP format; the second column utilizes the TO_CHAR function to display the seconds. If you want to include the fractional seconds, you need to add the FF format mask, which is omitted in this example. The third column shows the use of the EXTRACT function to return the seconds. You might notice the difference in the alignment of the result between the second and third columns. The TO_CHAR function returns the seconds as a string; the EXTRACT function returns the seconds as a NUMBER data type. The fractional seconds are always included when you use the SECOND keyword with this data type, but because they are zero, they are not shown in the result.

d) What do you observe about the text literal of the following query's WHERE clause?

```
SELECT col_timestamp_w_tz
  FROM date_example
 WHERE col_timestamp_w_tz = '24-FEB-09 04.25.32.000000 PM -05:00'
COL_TIMESTAMP_W_TZ
-----------------------------------
24-FEB-09 04.25.32.000000 PM -05:00

1 row selected.
```

ANSWER: This SQL statement queries the column called COL_TIMESTAMP_W_TZ, using a TIMESTAMP WITH TIMEZONE display format literal.

You may use other formats in the WHERE clause to accomplish the same query result. For example, you can use the TO_TIMESTAMP_TZ function to explicitly convert the text literal already in default format to a TIMESTAMP WITH TIME ZONE data type.

```
SELECT col_timestamp_w_tz
FROM   date_example
WHERE  col_timestamp_w_tz =
       TO_TIMESTAMP_TZ('24-FEB-09
       04.25.32.000000 PM -05:00')
```

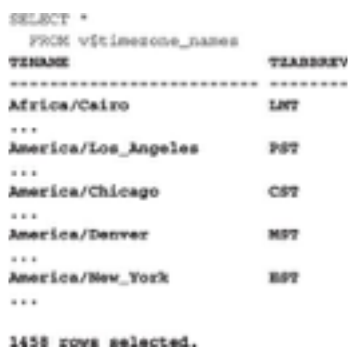
If you choose a text literal in a different format, you must supply the format mask, as illustrated in the next example. The TZH and TZM indicate the time zone displacement values in hours and minutes from UTC. In this example, the fractional seconds (FF) are not included because they are zero.

```
SELECT col_timestamp_w_tz
FROM   date_example
WHERE  col_timestamp_w_tz =
       TO_TIMESTAMP_TZ('24-FEB-2009
       16:25:32 -05:00', 'DD-MON-
       YYYY HH24:MI:SS TZH:TZM')
```

The next WHERE clause uses the region name instead of the time zone offset number value. Region names are expressed in the TZR format mask.

```
SELECT col_timestamp_w_tz
FROM   date_example
WHERE  col_timestamp_w_tz =
       TO_TIMESTAMP_TZ('24-FEB-2009
       16:25:32 EST', 'DD-MON-YYYY
       HH24:MI:SS TZR')
```

You can retrieve valid time zone region names from the data dictionary view V$TIMEZONE_ NAMES.



```
SELECT *
  FROM v$timezone_names
TZNAME                    TZABBREV
------------------------- --------
Africa/Cairo              LMT
...
America/Los_Angeles       PST
...
America/Chicago           CST
...
America/Denver            MST
...
America/New_York          EST
...

1458 rows selected.
```

Alternatively, if you want to express the WHERE clause in PST, you can use the America/Los_Angeles region name and change the actual hour literal from 16 to 13 to result in the same UTC.

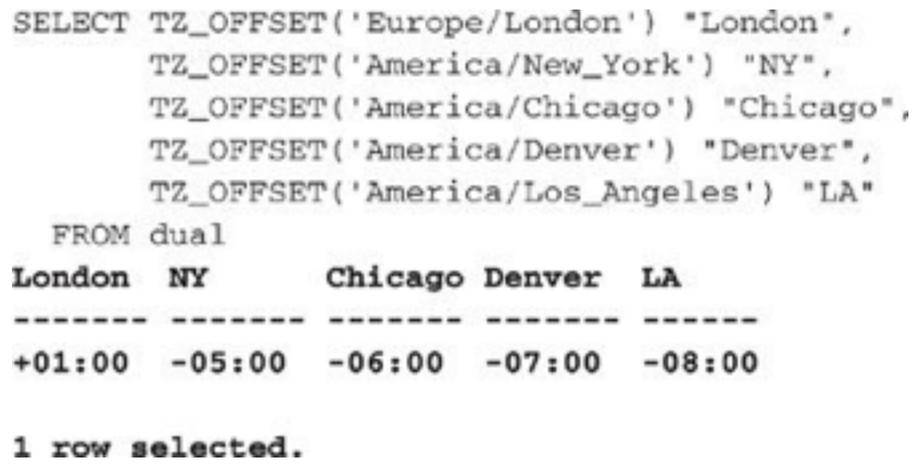
234

235

```
WHERE  col_timestamp_w_tz
       = TO_TIMESTAMP_TZ(
       ‘ 24-FEB-2009
       13:25:32 PST’,
       ‘ DD-MON-YYYY
       HH24:MI:SS TZR’)
```

THE TZ_OFFSET FUNCTION

You can find out the time differences between the UTC and the individual time zones with the TZ_OFFSET function. Following is a query that illustrates the appropriate offset values. Note that the query result is different when daylight saving time is in effect.



```
SELECT TZ_OFFSET('Europe/London') "London",
       TZ_OFFSET('America/New_York') "NY",
       TZ_OFFSET('America/Chicago') "Chicago",
       TZ_OFFSET('America/Denver') "Denver",
       TZ_OFFSET('America/Los_Angeles') "LA"
  FROM dual
London  NY      Chicago Denver  LA
------- ------- ------- ------- ------
+01:00  -05:00  -06:00  -07:00  -08:00

1 row selected.
```

A COMMON TIME FORMAT MASK ERROR

Here's one common mistake you can avoid in conjunction with the datetime-related data types. In the next query, the HH24 mask is used simultaneously with the A.M./P.M. format mask. The Oracle error indicates that you must choose either HH24 or use the HH (or HH12) together with the A.M./P.M.. mask to adjust the time to either 24-hour or 12-hour format.

```
SQL> SELECT col_timestamp_w_tz
  2    FROM date_example
  3   WHERE col_timestamp_w_tz =
  4         TO_TIMESTAMP_TZ('24-FEB-2009 16:25:32 PM -05:00',
  5                         'DD-MON-YYYY HH24:MI:SS PM TZH:TZM')
  6  /
                        'DD-MON-YYYY HH24:MI:SS PM TZH:TZM')
                        *
ERROR at line 5:
ORA-01818: 'HH24' precludes use of meridian indicator
```

e) The following SQL statements are issued against the database server. Explain the results.

```
SELECT SESSIONTIMEZONE
  FROM dual
SESSIONTIMEZONE
---------------
-05:00

1 row selected.
```

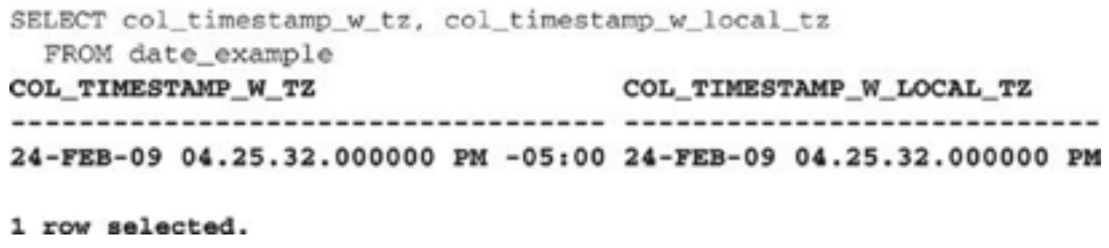
```
SELECT col_timestamp_w_tz, col_timestamp_w_local_tz
  FROM date_example
COL_TIMESTAMP_W_TZ                  COL_TIMESTAMP_W_LOCAL_TZ
----------------------------------- ---------------------------
24-FEB-09 04.25.32.000000 PM -05:00 24-FEB-09 04.25.32.000000 PM

1 row selected.

ALTER SESSION SET TIME_ZONE = '-8:00'
Session altered.

SELECT col_timestamp_w_tz, col_timestamp_w_local_tz
  FROM date_example
COL_TIMESTAMP_W_TZ                  COL_TIMESTAMP_W_LOCAL_TZ
----------------------------------- ---------------------------
24-FEB-09 04.25.32.000000 PM -05:00 24-FEB-09 01.25.32.000000 PM

1 row selected.

ALTER SESSION SET TIME_ZONE = '-5:00'
Session altered.
```

ANSWER: The following query determines the session’s current time zone offset value, which is –5 hours before UTC. (When daylight saving time is in effect, the time zone offset value changes.)

```
SELECT SESSIONTIMEZONE
  FROM dual
SESSIONTIMEZONE
---------------
-05:00

1 row selected.
```

The next query returns the currently stored values in the columns of the data types TIMESTAMP WITH TIME ZONE and TIMESTAMP WITH

LOCAL TIME ZONE. The dates and timestamps are identical in both columns. They represent the same date and time.



```
SELECT col_timestamp_w_tz, col_timestamp_w_local_tz
  FROM date_example
COL_TIMESTAMP_W_TZ                  COL_TIMESTAMP_W_LOCAL_TZ
----------------------------------- ----------------------------
24-FEB-09 04.25.32.000000 PM -05:00 24-FEB-09 04.25.32.000000 PM

1 row selected.
```

Now the session's time zone is changed to be equivalent to the West Coast time zone, which is 8 hours before UTC. This statement helps simulate a user's query result on the West Coast.

```
ALTER SESSION SET TIME_ZONE =
'-8:00'
```
`Session altered.`

The individual database user's local session time zone can be changed for the duration of the session with the ALTER SESSION command. Alternatively, this could also be achieved by changing the user's operating system time zone value, but the ALTER SESSION commands will always override the operating system settings.

The query is reissued, and when you compare the two column values, the second column with the data type TIMESTAMP WITH LOCAL TIME ZONE shows a different value. The local timestamp is adjusted to the local West Coast time.

```
SELECT col_timestamp_w_tz, col_timestamp_w_local_tz
  FROM date_example
COL_TIMESTAMP_W_TZ                 COL_TIMESTAMP_W_LOCAL_TZ
---------------------------------- ---------------------------
24-FEB-09 04.25.32.000000 PM -05:00 24-FEB-09 01.25.32.000000 PM

1 row selected.
```

The next statement resets the time zone to its initial time zone offset value, -5:00, as determined by the SESSIONTIMEZONE function issued previously.

```
SELECT col_timestamp_w_tz, col_timestamp_w_local_tz
  FROM date_example
COL_TIMESTAMP_W_TZ                 COL_TIMESTAMP_W_LOCAL_TZ
---------------------------------- ---------------------------
24-FEB-09 04.25.32.000000 PM -05:00 24-FEB-09 04.25.32.000000 PM

1 row selected.
```

When you reissue the query against the DATE_EXAMPLE table, the local time is back to its original value.

```
SELECT student_id, registration_date,
       registration_date+TO_YMINTERVAL('01-06') "Grad. Date"
  FROM student
 WHERE student_id = 123
    STUDENT_ID REGISTRAT Grad. Dat
    ---------- --------- ---------
           123 27-JAN-07 27-JUL-08

1 row selected.
```

When you exit the SQL Developer or SQL*Plus session, these ALTER SESSION settings are no longer in effect. The ALTER SESSION settings persist only throughout the duration of the session.

Lab 5.3 Quiz

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

1) What data type will the following function return?

```
SELECT EXTRACT(MINUTE FROM INTERVAL '12:51' HOUR TO MINUTE)
  FROM dual
EXTRACT(MINUTEFROMINTERVAL'12:51'HOURTOMINUTE)
----------------------------------------------
                                            51

1 row selected.
```

______ a) DATE

______ b) TIMESTAMP

______ c) TIMESTAMP WITH TIME ZONE

______ d) An Oracle error message

2) The ALTER SESSION statement can change the session's time zone.

______ a) True

______ b) False

3) The TIMESTAMP WITH LOCAL TIME ZONE data type displays the local date and time.

______ a) True

______ b) False

4) The time zone displacement value indicates the time difference from UTC.

a) _____True

b) _____False

5) The TIMESTAMP WITH LOCAL TIME ZONE data type allows fractional seconds.

______ a) True

______ b) False

6) The following query displays five fractional seconds.

```
SELECT TO_CHAR(SYSTIMESTAMP,
'HH:MI:SS.FF5')
  FROM dual
```

______ a) True

______ b) False

ANSWERS APPEAR IN APPENDIX A.

LAB 5.4 Performing Calculations with the Interval Data Types

LAB OBJECTIVES

After this lab, you will be able to:

- Understand the Functionality of the Interval Data Types
- Perform Date-Related Calculations Using Intervals

Oracle has two interval data types: INTERVAL YEAR TO MONTH and INTERVAL DAY TO SECOND. These data types store the difference between two date values. Table 5.12 provides an overview of the two data types and respective example literals.

TABLE 5.12 Interval Data Types

DATA TYPE	PURPOSE AND EXAMPLES OF LITERALS

INTERVAL YEAR [(year_precision)] TO MONTH	Values are expressed in years and months. The default year precision is two digits. Literal examples: INTERVAL ‘3-2’ YEAR TO MONTH (This translates into 3 years and 2 months.) INTERVAL ‘2’ YEAR (2 years) INTERVAL ‘4’ MONTH (4 months) INTERVAL ‘36’ MONTH (36 months, or 3 years)

INTERVAL DAY [(day_precision)] TO SECOND [(fractional_seconds_precision)]	Values are expressed in days, hours, minutes, and seconds. The default precision for the DAY is 2; the fractional seconds precision has a six-digit default value. Literal examples: INTERVAL ‘30’ DAY (30 days) INTERVAL ‘200’ DAY (3) (This translates to 200 days; because the literal exceeds the default DAY precision of 2, you need to explicitly specify the precision.)

INTERVAL ‘12:51’ HOUR TO MINUTE (12 hours and 51 minutes)

INTERVAL ‘15’ MINUTE (15 minutes)

INTERVAL ‘3 5:10:15.10’ DAY TO SECOND (3 days, 5 hours, 10 minutes, 15 seconds, and 10 fractional seconds)

Note that the components must be contiguous; for example, you cannot skip the MINUTE component between the HOUR and SECOND components.

Using Intervals

You can use intervals for calculations as shown in the next example, where an interval of one year and six months is added to a student's registration date. The interval is represented as the literal '01-06'. The TO_YMINTERVAL function converts this text literal to the INTERVAL YEAR TO MONTH data type. The result of the query displays the graduation date as one year and six months after the REGISTRATION_DATE.

```
SELECT DISTINCT TO_CHAR(created_date, 'DD-MON-YY HH24:MI')
        "CREATED_DATE",
       TO_CHAR(start_date_time, 'DD-MON-YY HH24:MI')
        "START_DATE_TIME",
       start_date_time-created_date "Decimal",
       NUMTODSINTERVAL(start_date_time-created_date,'DAY')
        "Interval"
         FROM section
        ORDER BY 3
       CREATED_DATE    START_DATE_TIME Decimal    Interval
       --------------- --------------- ---------- ----------------------
       02-JAN-07 00:00 08-APR-07 09:30 96.3958333 96 9:29:59.999999999
       02-JAN-07 00:00 09-APR-07 09:30 97.3958333 97 9:29:59.999999999
       02-JAN-07 00:00 14-APR-07 09:30 102.395833 102 09:29:59.999999999
       ...
       02-JAN-07 00:00 24-JUL-07 09:30 203.395833 203 09:29:59.999999999

       29 rows selected.
```

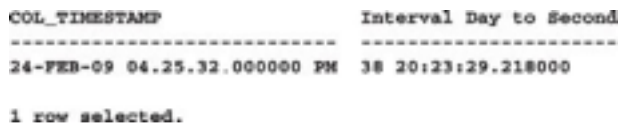
The individual components of the interval data types are listed in Table 5.13.

TABLE 5.13 Valid Value Ranges for Interval Components

INTERVAL COMPONENT	VALID VALUES
YEAR	Positive or negative integer; default precision is 2.
MONTH	00–11 (Note that the 12th month will be converted to a year.)
DAY	Positive or negative integer; default precision is 2.
HOUR	00–23
MINUTE	00–59
SECOND	00–59 (Plus optional precision up to nine-digit fractional seconds)

EXTRACT and Intervals

Just as with other datetime data types, you can use the EXTRACT function to extract specific components. This query retrieves the minutes.



```
COL_TIMESTAMP                Interval Day to Second
---------------------------- ----------------------
24-FEB-09 04.25.32.000000 PM 38 20:23:29.218000

1 row selected.
```

The interval data types allow a number of useful functions, as listed in Table 5.14.

Table 5.14 Useful Interval Functions

FUNCTION	PURPOSE	RETURN DATA TYPE
TO_YMINTERVAL(char)	Converts a text literal to an INTERVAL YEAR TO MONTH data type.	INTERVAL YEAR TO MONTH
TO_DSINTERVAL(char)	Converts a text literal to an INTERVAL DAY TO SECOND data type.	INTERVAL DAY TO SECOND
NUMTOYMINTERVAL (number,'YEAR') NUMTOYMINTERVAL (number,'MONTH')	Converts a number to an INTERVAL YEAR TO MONTH interval. You can use the 'YEAR' or 'MONTH' parameters.	INTERVAL YEAR TO MONTH

NUMTODSINTERVAL (number,'DAY')	Converts a number to an INTERVAL DAY TO SECOND literal. Instead of the DAY parameter, you can pass HOUR, MINUTE, or SECOND instead.	INTERVAL DAY TO SECOND
EXTRACT(MINUTE FROM interval data type)	Extracts specific components (for example, YEAR, MONTH, DAY, HOUR, MINUTE, SECOND).	NUMBER

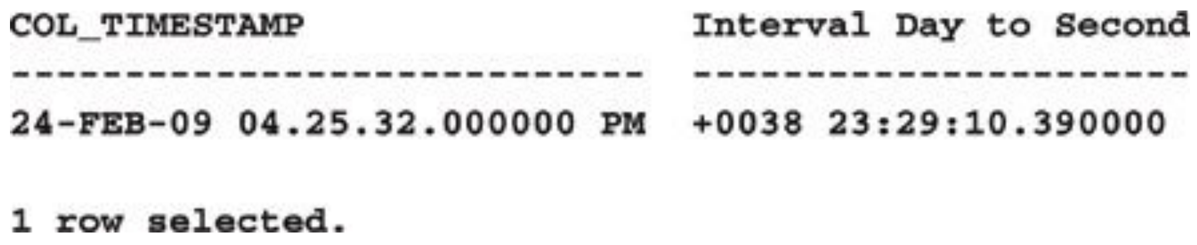
The next example expresses the time difference between the columns START_DATE_TIME and CREATED_DATE of the SECTION table. The first row of the output indicates that the difference between the two dates is the decimal result 96.3958333 days;

according to the fourth column, where the NUMTODSINTERVAL function is applied, this translates into 96 days, 9 hours, 29 minutes, and 59. 999999999 seconds.

```
SELECT  DISTINCT TO_CHAR(created_date, ‘DD-MON-YY
        HH24:MI’)
        “CREATED_DATE”,
        TO_CHAR(start_date_time, ‘DD-MON-YY HH24:MI’)
        “START_DATE_TIME”,
        start_date_time-created_date “Decimal”,
        NUMTODSINTERVAL(start_date_time-created_date,
        ‘DAY’)
        “Interval”
 FROM   ORDER BY 3
section
```

241

242



```
COL_TIMESTAMP                 Interval Day to Second
----------------------------  ----------------------
24-FEB-09 04.25.32.000000 PM  +0038 23:29:10.390000

1 row selected.
```

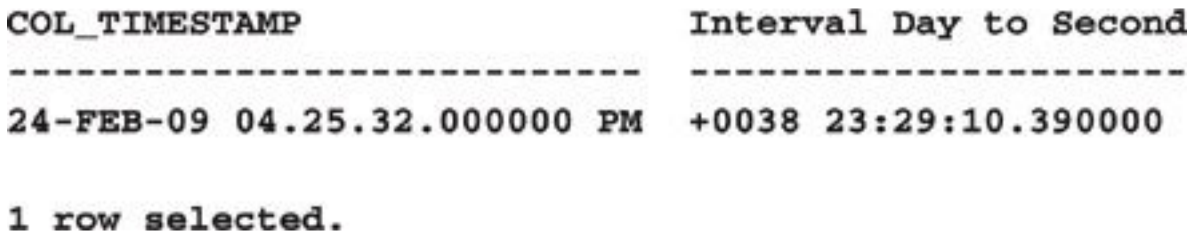
Interval Expressions

As an alternative to the NUMTODSINTERVAL or the NUMTOYMINTERVAL function, you can use an *interval expression*, which can be either DAY TO SECOND or YEAR TO MONTH. The next example shows as the first column the value of data type TIMESTAMP in the COL_ TIMESTAMP column of the

DATE_EXAMPLE table. The second column subtracts from the SYSTIMESTAMP function the COL_TIMESTAMP column and displays the difference as an interval of DAY TO SECOND, resulting in a difference of 38 days, 20 hours, 23 minutes, 29 seconds, and 218000 fractional seconds. The value of SYSTIMESTAMP at the time of the query was 04-APR-09 01.49.01.218000000 PM -04:00. The precision of the DAY interval is set to 4 to allow for larger numbers.

```
SELECT col_timestamp,
       (SYSTIMESTAMP -
       col_timestamp) DAY(4)
       TO SECOND
       "Interval Day to
       Second"
FROM date_example
```

Depending on whether you are executing the query against SQL Developer or SQL*Plus, the interval value is displayed in a slightly different format. Following is the format in SQL Developer.



```
COL_TIMESTAMP                  Interval Day to Second
----------------------------   ----------------------
24-FEB-09 04.25.32.000000 PM   +0038 23:29:10.390000

1 row selected.
```

The SQL*Plus command-line version displays the interval prefixed with a + or - symbol, similar to the following result.

```
COL_TIMESTAMP                 Interval Day to Second
----------------------------  ----------------------
24-FEB-09 04.25.32.000000 PM  +0038 23:29:10.390000

1 row selected.
```

Instead of using the DAY TO SECOND interval, the next query uses a YEAR TO MONTH interval. This result displays as zero years and 1 month.

```
SELECT col_timestamp,
       (SYSTIMESTAMP - col_timestamp) YEAR TO MONTH
       "Interval Year to Month"
  FROM date_example
COL_TIMESTAMP          Interval Year to Month
---------------------  ----------------------
24-FEB-09 04.25.32 PM  0-1

1 row selected.
```

Determining Overlaps

The overlaps functionality is implemented in Oracle but not documented. The OVERLAPS operator is useful to determine whether two time periods overlap. For example, you can use this operator to determine whether a planned meeting conflicts with other scheduled meetings.

The next table, called MEETING, contains three columns: a MEETING_ID column and two columns, MEETING_START and MEETING_END, that determine the start and end dates and times of a meeting.

```
SQL> DESCRIBE meeting
Name                  Null?    Type
--------------------- -------- ----------
MEETING_ID                     NUMBER(10)
MEETING_START                  DATE
MEETING_END                    DATE
```

The table has two rows, as you see from the following SELECT statement.

```
SELECT meeting_id,
       TO_CHAR(meeting_start, 'DD-MON-YYYY HH:MI PM') "Start",
       TO_CHAR(meeting_end, 'DD-MON-YYYY HH:MI PM') "End"
  FROM meeting
MEETING_ID Start               End
---------- ------------------- -------------------
         1 01-JUL-2009 09:30 AM 01-JUL-2009 10:30 AM
         2 01-JUL-2009 03:00 PM 01-JUL-2009 04:30 PM

2 rows selected.
```

If you want to find out whether a particular date and time conflict with any of the already scheduled meetings, you can issue this SQL query with the OVERLAPS operator.

This operator is used just like any of the other comparison operators in the WHERE clause of a SQL statement. Here it compares the column pair MEETING_START and MEETING_END with the date and time 01-JUL-2009 3:30 PM and a two-hour interval. The row that overlaps is returned in the output.

```
SELECT meeting_id,
       TO_CHAR(meeting_start, 'dd-mon-yyyy hh:mi pm') "Start",
       TO_CHAR(meeting_end, 'dd-mon-yyyy hh:mi pm') "End"
  FROM meeting
 WHERE (meeting_start, meeting_end)
       OVERLAPS
       (to_date('01-JUL-2009 3:30 PM', 'DD-MON-YYYY HH:MI PM'),
          INTERVAL '2' HOUR)
MEETING_ID Start                End
---------- -------------------- --------------------
         2 01-JUL-2009 03:00 PM 01-JUL-2009 04:30 PM

1 row selected.
```

Alternatively, if you want to find out which meetings do *not* conflict, you can negate the predicate with the NOT logical operator, as shown in the next example.

```
SELECT meeting_id,
       TO_CHAR(meeting_start, 'DD-MON-YYYY HH:MI PM') "Start",
       TO_CHAR(meeting_end, 'DD-MON-YYYY HH:MI PM') "End"
  FROM meeting
 WHERE NOT (meeting_start, meeting_end)
       OVERLAPS
       (TO_DATE('01-JUL-2009 3:30PM', 'DD-MON-YYYY HH:MI PM'),
          INTERVAL '2' HOUR)
MEETING_ID Start                End
---------- -------------------- --------------------
         1 01-JUL-2009 09:30 AM 01-JUL-2009 10:30 AM

1 row selected.
```

The syntax for OVERLAPS is as follows.

```
event OVERLAPS event
```

In this syntax, event is either of the following.

```
(start_event_date_time,
end_event_start_time)
(start_event_date_time,
interval_duration)
```

While the OVERLAP operator provides useful functionality, you need to understand Oracle's interpretation of an overlap. If the start date/time of one record and the end date/time of the other record are identical, this is not considered a conflict. For example, if one meeting starts at 1:00 P.M. and ends at 3:00 P.M., it would not be considered a conflict with another meeting that starts at 3:00 P.M. and ends at 4:30 P.M. However, if the second meeting started at 2:59 P.M., it would be an overlap.

LAB 5.4 EXERCISES

a) Explain the result of the following SQL statement.

```
SELECT section_id "ID",
       TO_CHAR(created_date, 'MM/DD/YY HH24:MI')
         "CREATED_DATE",
       TO_CHAR(start_date_time, 'MM/DD/YY HH24:MI')
         "START_DATE_TIME",
       NUMTODSINTERVAL(start_date_time-created_date, 'DAY')
         "Interval"
  FROM section
 WHERE NUMTODSINTERVAL(start_date_time-created_date, 'DAY')
       BETWEEN INTERVAL '100' DAY(3) AND INTERVAL '120' DAY(3)
 ORDER BY 3
 ID CREATED_DATE   START_DATE_TIM Interval
--- -------------- -------------- -----------------------
 79 01/02/07 00:00 04/14/07 09:30 102 09:29:59.999999999
 87 01/02/07 00:00 04/14/07 09:30 102 09:29:59.999999999
...
152 01/02/07 00:00 04/29/07 09:30 117 09:29:59.999999999
125 01/02/07 00:00 04/29/07 09:30 117 09:29:59.999999999

17 rows selected.
```

b) Explain the result of the following SQL statements. What do you observe?

```
SELECT NUMTODSINTERVAL(360, 'SECOND'),
       NUMTODSINTERVAL(360, 'MINUTE')
  FROM dual
NUMTODSINTERVAL(360,'SECOND') NUMTODSINTERVAL(360,'MINUTE')
----------------------------- -----------------------------
0 0:6:0.0                     0 6:0:0.0

1 row selected.
SELECT NUMTODSINTERVAL(360, 'HOUR'),
       NUMTODSINTERVAL(360, 'DAY')
  FROM dual
NUMTODSINTERVAL(360,'HOUR')   NUMTODSINTERVAL(360,'DAY')
----------------------------- -----------------------------
15 0:0:0.0                    360 0:0:0.0

1 row selected.
```

LAB 5.4 EXERCISE ANSWERS

a) Explain the result of the following SQL statement.

```
SELECT section_id "ID",
       TO_CHAR(created_date, 'MM/DD/YY HH24:MI')
         "CREATED_DATE",
       TO_CHAR(start_date_time, 'MM/DD/YY HH24:MI')
         "START_DATE_TIME",
       NUMTODSINTERVAL(start_date_time-created_date, 'DAY')
         "Interval"
  FROM section
 WHERE NUMTODSINTERVAL(start_date_time-created_date, 'DAY')
       BETWEEN INTERVAL '100' DAY(3) AND INTERVAL '120' DAY(3)
 ORDER BY 3
 ID CREATED_DATE   START_DATE_TIM Interval
--- -------------- -------------- ----------------------
 79 01/02/07 00:00 04/14/07 09:30 102 09:29:59.999999999
 87 01/02/07 00:00 04/14/07 09:30 102 09:29:59.999999999
...
152 01/02/07 00:00 04/29/07 09:30 117 09:29:59.999999999
125 01/02/07 00:00 04/29/07 09:30 117 09:29:59.999999999

17 rows selected.
```

ANSWER: The query shows four columns: SECTION_ID, CREATED_DATE, START_DATE_TIME (the date and time a section starts), and Interval. The Interval column expresses the difference between the START_DATE_TIME column and the CREATED_DATE column, in days, hours, minutes, and seconds, using the NUMTODSINTERVAL function. Without this function, the calculation returns the time portion as a decimal. The WHERE clause of the query retrieves rows with a time difference value between 100 and 120 days. The ORDER BY clause sorts the result by the START_DATE_TIME column values.

The WHERE clause uses both the NUMTODSINTERVAL function and the INTERVAL expression and checks whether the result falls between the INTERVAL literals 100 and 120 days. Because INTERVAL DAY has a default precision of 2, you must include the three-digit precision as DAY(3).

b) Explain the result of the following SQL statements. What do you observe?

```
SELECT NUMTODSINTERVAL(360, 'SECOND'),
       NUMTODSINTERVAL(360, 'MINUTE')
  FROM dual
NUMTODSINTERVAL(360,'SECOND') NUMTODSINTERVAL(360,'MINUTE')
----------------------------- -----------------------------
0 0:6:0.0                     0 6:0:0.0

1 row selected.

SELECT NUMTODSINTERVAL(360, 'HOUR'),
       NUMTODSINTERVAL(360, 'DAY')
  FROM dual
NUMTODSINTERVAL(360,'HOUR')   NUMTODSINTERVAL(360,'DAY')
----------------------------- -----------------------------
15 0:0:0.0                    360 0:0:0.0

1 row selected.
```

```
NUMTODSINTERVAL(360,'SECOND')  NUMTODSINTERVAL(360,'MINUTE')
-----------------------------  -----------------------------
+000000000 00:06:00.000000000  +000000000 06:00:00.000000000

1 row selected.
```

ANSWER: These SQL statements illustrate how a number literal is translated to an interval, using the NUMSTODSINTERVAL function with various parameter options.

The first SQL statement shows the number literal 360 converted into seconds in the first column. The second column translates it into minutes. From the result, 360 seconds are now 6 minutes, as indicated with the 0 0:6.0 interval. The second column shows the 360 minutes converted into 6 hours, as displayed with the interval 0 6:0:0.0.

The second statement performs the same conversion; this time, the number literal represents hours, which are translated into 15 days. The second column displays 360 days in the interval format.

If you use the SQL*Plus command-line version, your query result may display differently, and you will receive a result similar to the following.

```
NUMTODSINTERVAL(360,'SECOND')  NUMTODSINTERVAL(360,'MINUTE')
-----------------------------  -----------------------------
+000000000 00:06:00.000000000  +000000000 06:00:00.000000000

1 row selected.
```

Lab 5.4 Quiz

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

1) The TO_YMINTERVAL function converts a text literal to an INTERVAL DAY TO SECOND data type.

   ______ a) True

   ______ b) False

2) The NUMTODSINTERVAL function converts a number to an INTERVAL YEAR TO MONTH data type.

   ______ a) True

   ______ b) False

3) The EXTRACT function is not valid for the INTERVAL YEAR TO MONTH data type.

   ______ a) True

   ______ b) False

4) The following interval literal is invalid.

```
INTERVAL '5 10:30:10.00' DAY TO
SECOND
```

______ a) True

______ b) False

ANSWERS APPEAR IN APPENDIX A.

LAB 5.5 Converting from One Data Type to Another

LAB OBJECTIVES

After this lab, you will be able to:

- Convert Between Different Data Types
- Format Data

SQL agrees with the saying "You can't compare apples to oranges." When you compare two values or columns, they must be of the same data type or of compatible data types. Sometimes Oracle can implicitly convert from one data type to another. It is preferable to explicitly specify the conversion with a function to avoid any ambiguities or errors when your SQL statement is executed.

Data Type Conversion

You have already learned about implicit conversion in the context of the datetime functions discussed in the previous labs. In the following SQL statement, the WHERE clause compares a text literal against the numeric COURSE_NO column. When Oracle compares a character data type, in this case the text literal ‘350’, against the NUMBER data type, which is the COURSE_ NO column, Oracle implicitly converts the character data to a NUMBER. This works perfectly, as you see from the query result.

```
SQL> DESCRIBE conversion_example
Name                 Null?    Type
-------------------- -------- -----------
COURSE_NO                     VARCHAR2(9)
```

Clearly, in this example, you have control over the literal and can simply change the text literal ‘350' to a NUMBER to avoid the implicit conversion. Such a change becomes more difficult or impossible when you are working in a programming language where you may not have influence over the data type of a supplied value. 249 250 Inevitably, things can go wrong, as you see illustrated in the following table, called CONVERSION_EXAMPLE. (This table is not part of the STUDENT schema, but you can download it from the companion Web site.)

```
SELECT *
  FROM conversion_example
COURSE_NO
---------
123
xyz

2 rows selected.
```

The following SELECT statement retrieves all the rows from the table. The COURSE_NO column in this table is of data type VARCHAR2, and therefore it accepts both numeric and alphanumeric entries. The table contains two rows: one with the value 123 and another with the value xyz.

```
SELECT *
  FROM conversion_example
 WHERE course_no = '123'
COURSE_NO
---------
123

1 row selected.
```

To illustrate the effects of the implicit data conversion, first query the row with the value 123 in the COURSE_NO column. As you can see, this statement executes flawlessly because the COURSE_NO column is a VARCHAR2 column, and the text literal 123 is enclosed in single quotes.

```
SELECT *
  FROM conversion_example
 WHERE course_no = 123
ERROR:
ORA-01722: invalid number

no rows selected
```

The next query does not enclose the literal in single quotes; in fact, it now represents a number literal. Oracle implicitly converts the COURSE_NO column to a NUMBER data type, resulting in an ORA-01722 ("invalid number") error. This error occurs because all the values in the COURSE_NO column are now implicitly converted into the NUMBER data type. But this conversion cannot be completed because one of the rows, the row with the value xyz, obviously cannot be converted into a NUMBER. Therefore, the query does not return any rows.

```
SELECT *
   FROM conversion_example
 WHERE course_no = 123
ERROR:ORA-01722: invalid number
no rows selected
```

To avoid this error, it is always best to explicitly specify the conversion function to make sure the data types agree. You accomplish the conversion with the TO_CHAR function, which converts the passed parameter into a character data type, as you see in the following SQL statement.

```
SELECT *
   FROM conversion_example
```

```
WHERE course_no = TO_CHAR(123)
```

You might wonder why you would bother even adding the TO_CHAR function if you can just enclose the values in quotation marks. Clearly, the easiest solution is to simply enclose the value in single quotes, but, as previously mentioned, you may encounter cases in which you do not have control over the literal or in which you are comparing one table's column against another table's column.

The CAST Function

The CAST function converts from one data type to another. It can be applied to Oracle's most commonly used built-in data types (that is, VARCHAR2, CHAR, NUMBER, and the datetime variants) or with a user-defined data type or subquery. (The creation of user-defined data types is beyond the scope of this book. You will learn about subqueries in Chapter 8, "Subqueries.")

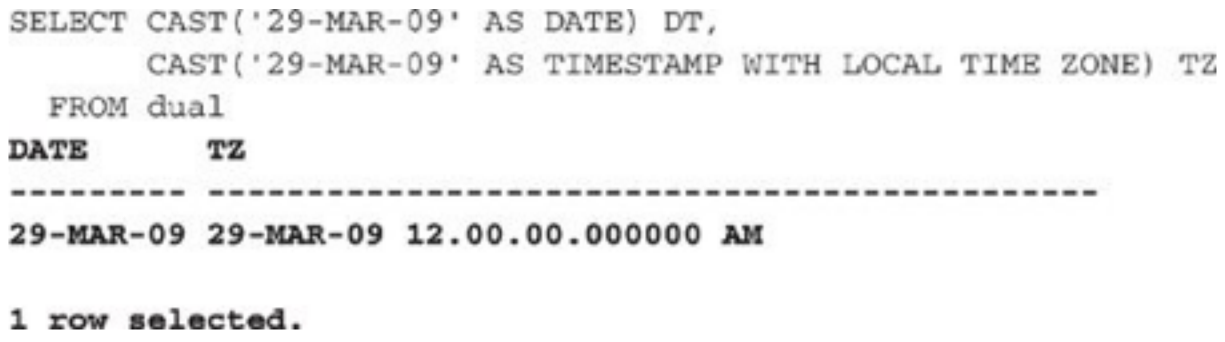
The syntax for the CAST function is as follows.

```
CAST(expression AS data type)
```

Following are examples of how CAST is used with Oracle's familiar data types. The SELECT statement contains CAST instead of the TO_CHAR function. When converting to a VARCHAR2 or CHAR data type, also referred to as *casting*, you need to specify the length. In this case, it is three characters long.

```
SELECT  *
FROM    conversion_example
WHERE   course_no = CAST(123
        AS VARCHAR2(3))
```

The next query casts the text literal '29-MAR-09' into a DATE data type in the first column and as a TIMESTAMP WITH LOCAL TIME ZONE data type in the second column.



```
SELECT CAST('29-MAR-09' AS DATE) DT,
       CAST('29-MAR-09' AS TIMESTAMP WITH LOCAL TIME ZONE) TZ
  FROM dual
DATE      TZ
--------- ---------------------------------------------
29-MAR-09 29-MAR-09 12.00.00.000000 AM

1 row selected.
```

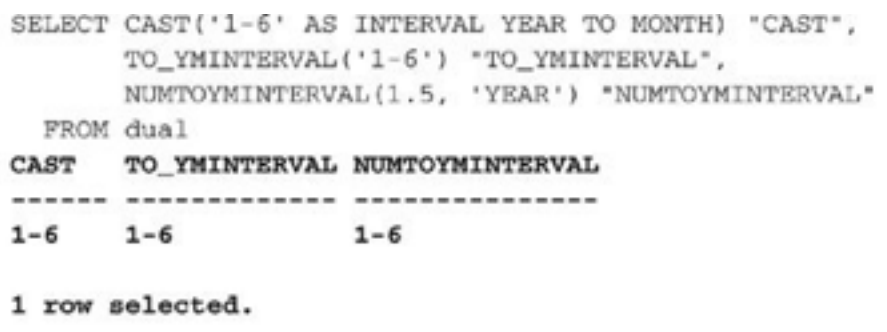
251

252

You can use CAST not only in the SELECT list but also in the WHERE clause.

```
SELECT  section_id,
        TO_CHAR(start_date_time,
        'DD-MON-YYYY HH24:MI:SS')
FROM    section
WHERE   start_date_time >=
        CAST('01-JUL-2007' AS
        DATE)
AND     start_date_time <
        CAST('01-AUG-2007' AS
        DATE)
```

The following statement casts the literal ‘04-JUL-2007 10:00:00 AM’ into the TIMESTAMP data type, because the FROM_TZ function requires this data type as the first parameter. The FROM_TZ function (discussed in Lab 5.3) converts the TIMESTAMP value into a TIMESTAMP WITH TIME ZONE data type. The chosen time zone for date literal is the time zone region name ‘America/New_York’. The AT TIME ZONE keywords of the resulting expression display the value in the local Los Angeles time.



```
SELECT CAST('1-6' AS INTERVAL YEAR TO MONTH) "CAST",
       TO_YMINTERVAL('1-6') "TO_YMINTERVAL",
       NUMTOYMINTERVAL(1.5, 'YEAR') "NUMTOYMINTERVAL"
  FROM dual
CAST   TO_YMINTERVAL NUMTOYMINTERVAL
------ ------------- ---------------
1-6    1-6           1-6

1 row selected.
```

The next example illustrates the use of CAST on intervals. The text literal '1-6' is converted into the INTERVAL YEAR TO MONTH data type. As always, there are multiple ways to accomplish the same functionality in the SQL language; here the TO_YMINTERVAL function performs the identical function. The NUMTOYMINTERVAL function requires a NUMBER data type as an input parameter and translates the number 1.5 into one year and six months. (Refer to Table 5.14.)

```
SELECT CAST('1-6' AS INTERVAL YEAR TO MONTH) "CAST",
       TO_YMINTERVAL('1-6') "TO_YMINTERVAL",
       NUMTOYMINTERVAL(1.5, 'YEAR') "NUMTOYMINTERVAL"
  FROM dual
CAST   TO_YMINTERVAL NUMTOYMINTERVAL
------ ------------- ---------------
1-6    1-6           1-6

1 row selected.
```

Following is an example of a SQL statement that casts the COST column of the COURSE table into a BINARY_FLOAT data type. Alternatively, you can also use the TO_BINARY_FLOAT conversion function.

```
SELECT CAST(cost AS BINARY_FLOAT) AS cast,
       TO_BINARY_FLOAT(cost) AS to_binary_float
  FROM course
 WHERE course_no < 80

      CAST TO_BINARY_FLOAT
---------- ---------------
1.195E+003       1.195E+003
1.195E+003       1.195E+003
1.195E+003       1.195E+003

3 rows selected.
```

CAST Versus Oracle’s Conversion Functions

You might wonder why you should use CAST instead of any of the other Oracle conversion functions. The CAST function is ANSI SQL 1999 compliant, so there is no need to learn multiple Oracle-specific functions. However, some of Oracle’s built-in data types, such as the various LOB types, LONG RAW, and LONG, cannot be converted from one data type to another using CAST. Instead, you must use Oracle’s individual conversion functions. One disadvantage of the CAST function is casting into VARCHAR2 and CHAR data types because they need to be constrained to a determined length. The TO_DATE function and TO_CHAR functions are overall very versatile as they allow you a large variety of different format model choices. So you may choose the functions that fit your specific requirements.

Table 5.15 provides an overview of Oracle conversion functions. This lab concentrates on the TO_NUMBER, TO_CHAR, and CAST functions. In previous labs, you learned about the TO_DATE and TO_CHAR conversion functions, as well as conversion functions related to datetime and interval data types (refer to Table 5.11 and Table 5.14).

TABLE 5.15 Frequently Used Data Type Conversion Functions

FUNCTION	PURPOSE
TO_NUMBER(char [, format_mask])	Converts a VARCHAR2 or CHAR to a NUMBER.
TO_BINARY_FLOAT(expression [,format_mask])	Converts a character or numeric value to BINARY_FLOAT.
TO_BINARY_DOUBLE(expression, [format_mask])	Converts a character or numeric value to BINARY_DOUBLE.
TO_CHAR(datetime [, format_mask])	Converts a datetime value to a VARCHAR2.
TO_CHAR(number [, format_mask])	Converts a NUMBER to a VARCHAR2.
TO_CLOB(char)	Converts a VARCHAR2 or CHAR to a CLOB.

TO_DATE(char [, format_mask])	Converts a VARCHAR2 or CHAR to a DATE. For timestamp and time zones, see Lab 5.3.
CAST(expression AS data type)	Converts from one data type to another. Can be used for Oracle's most commonly used data types and for user-defined data types.

Formatting Data

The TO_CHAR conversion function is useful not only for data conversions between different data types but also for formatting data. The next SQL statement shows how a format mask can be applied with this function. To display a formatted result for the COST column, for instance, you can apply the format mask ‘999,999'. The values in the COST column are then formatted with a comma separating the thousands.

```
SELECT course_no, cost,
       TO_CHAR(cost, '999,999') formatted
  FROM course
 WHERE course_no < 25
COURSE_NO       COST FORMATTED
---------- ---------- ---------
        10       1195     1,195
        20       1195     1,195

2 rows selected.
```

The conversion function used in the SELECT statement does not modify the values stored in the database but rather performs a "temporary" conversion for the purpose of executing the statement. In Chapter 2, "SQL: The Basics," you learned about the SQL*Plus COLUMN FORMAT command, which achieves the same result. However, if you execute the SQL statement from a program other than SQL*Plus, the COLUMN command is not available, and you must use the TO_CHAR function to format the result.

The following statement shows both the effects of the SQL*Plus COLUMN FORMAT command and the TO_CHAR function. The column labeled "SQL*PLUS" is formatted with the COL "SQL*PLUS" FORMAT 999,999 command, the last column, labeled "CHAR," is formatted with the TO_CHAR function.

```
COL "SQL*PLUS" FORMAT 999,999
SELECT course_no, cost "SQL*PLUS",
       TO_CHAR(cost, '999,999') "CHAR"
  FROM course
 WHERE course_no < 25
COURSE_NO SQL*PLUS CHAR
--------- -------- --------
       10    1,195    1,195
       20    1,195    1,195

2 rows selected.
```

Table 5.16 provides an overview of the most popular NUMBER format models in conjunction with the TO_CHAR function.

Rounding can be accomplished not only with the ROUND function but also with a format model. 254

TABLE 5.16 Common NUMBER Format Models

FORMAT MASK	EXAMPLE VALUE	APPLIED TO_CHAR FUNCTION	RESULT
999,990.99	.45	TO_CHAR(.45, ‘999,990.99’)	0.45 (Note the leading zero)
$99,999.99	1234	TO_CHAR(1234,'$99,999.99’)	$1,234.00
0999	123	TO_CHAR(123, ‘0999’)	0123
L9999.99	1234.99	TO_CHAR(1234.99, ‘L9999.99’)	$1234.99 (local currency)
L99G999D99	1234.56	TO_CHAR(1234.56, ‘L99G999D99’)	$1,234.56 (local values for currency, group, and decimal separators)
999PR	-123	TO_CHAR(-123, ‘999PR’)	<123>
999MI	-123	TO_CHAR(-123, ‘999MI’)	123-
999s	-123	TO_CHAR(-123, ‘999s’)	123-
s999	-123	TO_CHAR(-123, ‘s999’)	-123
999	123.59	TO_CHAR(123.59, ‘999’)	124 (Note the rounding)

LAB 5.5 EXERCISES

Use the following SQL statement as the source query for Exercises a through c.

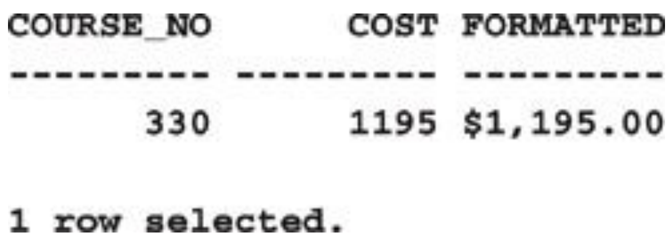
```
SELECT zip, city
FROM zipcode
WHERE zip = 10025
```

a) Rewrite the query, using the TO_CHAR function in the WHERE clause.

b) Rewrite the query, using the TO_NUMBER function in the WHERE clause.

c) Rewrite the query, using CAST in the WHERE clause.

d) Write the SQL statement that displays the following result. Note that the last column in the result shows the formatted COST column with a leading dollar sign and a comma to separate the thousands. Include the cents in the result as well.



```
COURSE_NO      COST FORMATTED
--------- --------- ---------
      330      1195 $1,195.00

1 row selected.
```

e) List the COURSE_NO and COST columns for courses that cost more than 1500. The third, fourth, and fifth columns show the cost increased by 15 percent. Display the increased cost columns, one with a leading dollar sign, and separate the thousands, and in another column show the same formatting but rounded to the nearest dollar. The result should look similar to the following output.

```
Increase
-------------------------------------------------------------
The price for course# 80 has been increased to $1,834.25.

1 row selected.
```

f) Based on Exercise e, write the query to achieve this result. Use the fm format mask to eliminate the extra spaces.

```
Increase
-------------------------------------------------------------
The price for course# 80 has been increased to $1,834.25.

1 row selected.
```

LAB 5.5 EXERCISE ANSWERS

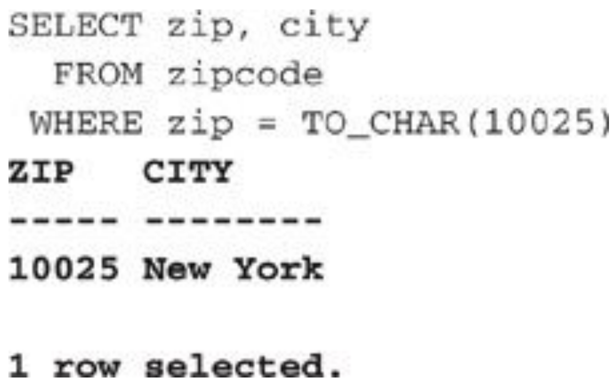
Use the following SQL statement as the source query for Exercises a through c.

Type and execute the following query for the following exercises.

```
SELECT zip, city
   FROM zipcode
 WHERE zip = 10025
```

a) Rewrite the query, using the TO_CHAR function in the WHERE clause.

ANSWER: The TO_CHAR function converts the number literal to a VARCHAR2 data type, which makes it equal to the VARCHAR2 data type of the ZIP column.



```
SELECT zip, city
  FROM zipcode
 WHERE zip = TO_CHAR(10025)
ZIP   CITY
----- --------
10025 New York

1 row selected.
```

b) Rewrite the query, using the TO_NUMBER function in the WHERE clause.

ANSWER: The VARCHAR2 data type of the ZIP column is converted to a NUMBER data type by applying the TO_NUMBER function.

Oracle then compares it to the number literal 10025.

```
SELECT zip, city
  FROM zipcode
 WHERE TO_NUMBER(zip) = 10025
ZIP   CITY
----- --------
10025 New York

1 row selected.
```

When you compare the results of the SQL statements from Answers a and b, they are identical. Answer b is less desirable because a function is applied to a database column in the WHERE clause. This disables the use of any indexes that may exist on the ZIP column and may require Oracle to read every row in the table instead of looking up the value in the index. Applying functions to database columns in the SELECT clause does not affect performance.

It is best to explicitly specify the data type conversion functions and not rely on implicit conversions: Your statements are easier to understand, and the behavior is predictable. Oracle’s algorithms for implicit conversion may be subject to change across versions and products, and implicit conversion can have a negative impact on performance if the queried column is indexed. You will learn about indexes in Chapter 13 and about performance considerations in Chapter 18, "SQL Optimization."

c) Rewrite the query, using CAST in the WHERE clause.

ANSWER: You can write the query in one of the following ways.

```
SELECT zip, city
  FROM zipcode
 WHERE CAST(zip AS NUMBER) =
10025
SELECT zip, city
  FROM zipcode
 WHERE zip = CAST(10025 AS
VARCHAR2(5))
```

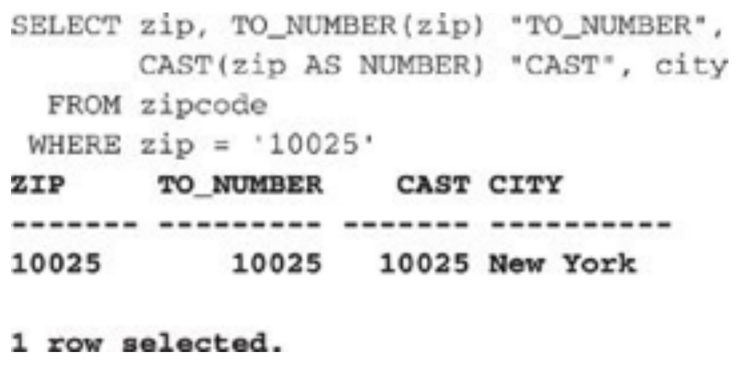
If you specify a too-short length for the VARCHAR2 data type, you receive an error similar to the following.

```
SELECT zip, city
  FROM zipcode
```

```
 WHERE zip = CAST(10025 AS
VARCHAR2(3))
```

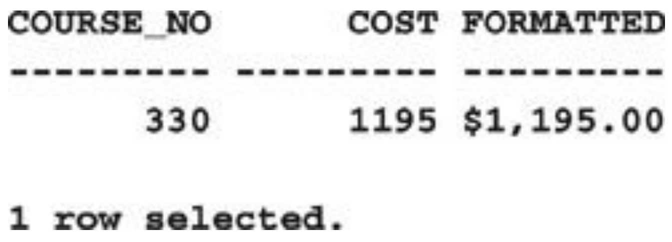
```
WHERE zip = CAST(10025 AS
VARCHAR2(3))
*
ERROR at line 3:
ORA-25137: Data value out of
range
```

In the next SQL query result, observe the way SQL*Plus displays the result of a NUMBER column versus the result in a character type column. In the output, you see as the first column the ZIP column in data type VARCHAR2. It is left aligned, just like the VARCHAR2 column CITY. In general, both in SQL*Plus and the SQL Developer Results window, values of the NUMBER data type are always right aligned; character values are always left aligned. However, in SQL Developer's Script Output tab, all columns are left aligned.



```
SELECT zip, TO_NUMBER(zip) "TO_NUMBER",
       CAST(zip AS NUMBER) "CAST", city
  FROM zipcode
 WHERE zip = '10025'
ZIP     TO_NUMBER    CAST CITY
------- --------- ------- ----------
10025       10025   10025 New York

1 row selected.
```

d) Write the SQL statement that displays the following result. Note that the last column in the result shows the formatted COST column with a leading dollar sign and a comma to separate the thousands. Include the cents in the result as well.

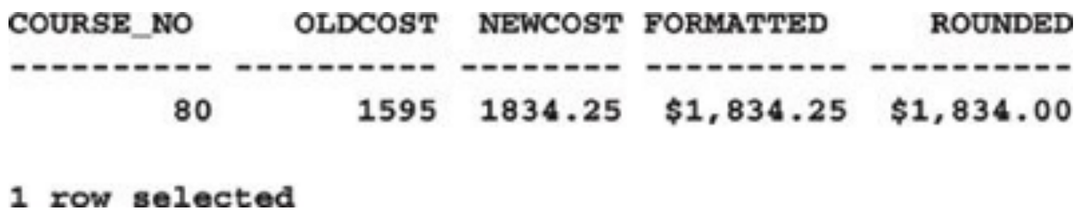


```
COURSE_NO      COST FORMATTED
--------- --------- ---------
      330      1195 $1,195.00

1 row selected.
```

ANSWER: The TO_CHAR function, together with the format mask in the SELECT clause of the statement, achieves the desired formatting.

```
SELECT course_no, cost,
TO_CHAR(cost, '$999,999.99')
Formatted
FROM course
WHERE course_no = 330
```

e) List the COURSE_NO and COST columns for courses that cost more than 1500. The third, fourth, and fifth columns show the cost increased by 15 percent. Display the increased cost columns, one with a leading dollar sign, and separate the thousands, and in another column show the same formatting but rounded to the nearest dollar. The result should look similar to the following output.

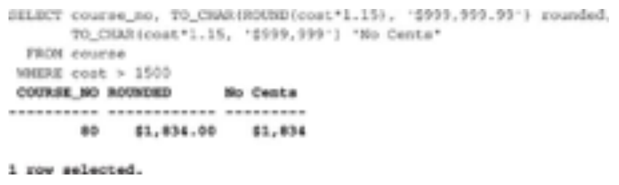


```
COURSE_NO    OLDCOST  NEWCOST FORMATTED     ROUNDED
---------- ---------- -------- ---------- ----------
        80       1595  1834.25  $1,834.25  $1,834.00

1 row selected
```

ANSWER: An increase of 15 percent requires a multiplication of the column COST by 1.15. You can round to the nearest dollar by using the ROUND function.

```
SELECT course_no, cost oldcost,
   cost*1.15 newcost,
   TO_CHAR(cost*1.15,
'$999,999.99') formatted,
   TO_CHAR(ROUND(cost*1.15),
'$999,999.99') rounded
 FROM course
  WHERE cost > 1500
```

Alternatively, the identical result is achieved with the format mask '$999,999', which omits the digits after the decimal point and rounds the cents, as shown in the next statement.



```
SELECT course_no, TO_CHAR(ROUND(cost*1.15), '$999,999.99') rounded,
       TO_CHAR(cost*1.15, '$999,999') "No Cents"
  FROM course
 WHERE cost > 1500
COURSE_NO ROUNDED      No Cents
---------- ------------ ---------
        80    $1,834.00    $1,834

1 row selected.
```

f) Based on the Exercise e, write the query to achieve this result. Use the fm format mask to eliminate the extra spaces.

```
Increase
------------------------------------------------------------
The price for course# 80 has been increased to $1,834.25.

1 row selected.
```

ANSWER: The following query achieves the desired result set. The fm format mask eliminates the blank padding.

```
SELECT 'The price for course# '||course_no||' has been increased to
➥'||
       TO_CHAR(cost*1.15, 'fm$999,999.99')||'.'
       "Increase"
  FROM course
 WHERE cost > 1500
```

Lab 5.5 Quiz

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

1) Which SQL statement results in an error? Select all that apply.

______ a) SELECT TO_CHAR('123') FROM dual

______ b) SELECT TO_CHAR(123) FROM dual

______ c) SELECT TO_NUMBER('001.99999') FROM dual

______ d) SELECT TO_NUMBER('A123') FROM dual

______ e) SELECT TO_CHAR('A123') FROM dual

______ f) SELECT TO_NUMBER(' 000123 ') FROM dual

2) Which of the following NUMBER format masks are valid? Select all that apply.

______ a) SELECT TO_CHAR(1.99,'9,9999.9X') FROM dual

______ b) SELECT TO_CHAR(1.99,'A99.99) FROM dual

______ c) SELECT TO_CHAR(1.99,'$000.99') FROM dual

______ d) SELECT TO_CHAR(1.99,'999.99') FROM dual

______ e) SELECT TO_CHAR(1.99,'.99') FROM dual

3) Explicit data type conversion is preferable to Oracle's implicit conversion.

______ a) True

______ b) False

4) The TO_CHAR, TO_NUMBER, and TO_DATE conversion functions are single-row functions.

______ a) True

______ b) False

5) How can you correct the following SQL error message?

```
 SQL> SELECT *
   2 FROM conversion_example
   3 WHERE course_no = CAST(123 AS
VARCHAR2)
   4 /
 WHERE course_no = CAST(123 AS
VARCHAR2)
    *
ERROR at line 3:
ORA-00906: missing left
parenthesis
```

a) _____Change the data type to CHAR.

b) _____Add a column length definition.

c) _____Choose a different aggregate function.

d) _____This query does not make sense.

ANSWERS APPEAR IN APPENDIX A.

WORKSHOP

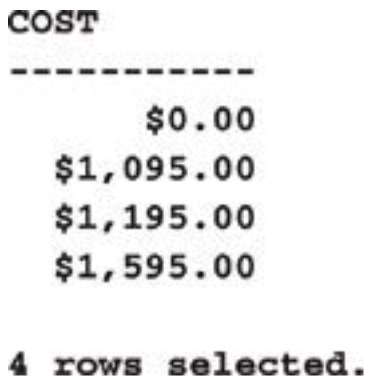
The projects in this section are meant to prompt you to utilize all the skills you have acquired throughout this chapter. The answers to these projects can be found at the companion Web site to this book, located at www.oraclesqlbyexample.com.

1) Display all the sections where classes start at 10:30 A.M.

2) Write a query that accomplishes the following result. The output shows you all the days of the week where sections 99, 89, and 105 start. Note the order of the days.

```
DAY  SECTION_ID
---- ----------
Mon          99
Tue          89
Wed         105

3 rows selected.
```

3) Select the distinct course costs for all the courses. If a course cost is unknown, substitute a zero. Format the output with a leading $ sign and separate the thousands with a comma. Display two digits after the decimal point. The query's output should look like the following result.



```
COST
-----------
      $0.00
  $1,095.00
  $1,195.00
  $1,595.00

4 rows selected.
```

4) List all rows of the GRADE_TYPE table that were created in the year 1998.

5) What, if anything, is wrong with the following SQL statement?

```
SELECT zip + 100
  FROM zipcode
```

6) For the students enrolled on January 30, 2007, display the columns STUDENT_ID and ENROLL_DATE.