**Working with Dates in PL/SQL**

*By Steven Feuerstein Oracle ACE Director*

**Part 5 in a series of articles on understanding and using PL/SQL**

The previous articles in this introductory PL/SQL series focused on working with strings and numbers in PL/SQL-based applications. Without a doubt, strings and numbers are important, but it is certainly a very rare application that does not also rely on dates. You need to keep track of when events occurred, when people were born, and much more.

As a result, you will quite often need to

* Declare variables and constants for dates
* Use built-in functions to display and modify date values
* Perform computations on dates

A date is also a considerably more complex datatype than a string or a number. It has multiple parts (year, month, day, hour, and so on), and there are many rules about what constitutes a valid date. This article gives you all the information you need in order to begin working with dates in your PL/SQL programs.

**Dates, Time Stamps, and Intervals in PL/SQL**

Most applications require the storage and manipulation of dates and times. Unlike strings and numbers, dates are quite complicated: not only are they highly formatted data, but there are also many rules for determining valid values and valid calculations (leap days and years, daylight saving time changes, national and company holidays, date ranges, and so on).

Fortunately, Oracle Database and PL/SQL provide a set of true date and time datatypes that store both date and time information in a standard internal format, and they also have an extensive set of built-in functions for manipulating the date and time.

There are three datatypes you can use to work with dates and times:

* DATE—This datatype stores a date and a time, resolved to the second. It does not include the time zone. DATE is the oldest and most commonly used datatype for working with dates in Oracle applications.
* TIMESTAMP—Time stamps are similar to dates, but with these two key distinctions: (1) you can store and manipulate times resolved to the nearest *billionth* of a second (9 decimal places of precision), and (2) you can associate a time zone with a time stamp, and Oracle Database will take that time zone into account when manipulating the time stamp.
* INTERVAL—Whereas DATE and TIMESTAMP record a specific point in time, INTERVAL records and computes a time *duration*. You can specify an interval in terms of years and months, or days and seconds.

Listing 1 includes example variables whose declaration is based on these datatypes.

**Code Listing 1:** Declaring DATE, TIMESTAMP, and INTERVAL variables

DECLARE

l\_today\_date DATE := SYSDATE;

l\_today\_timestamp TIMESTAMP := SYSTIMESTAMP;

l\_today\_timetzone TIMESTAMP WITH TIME ZONE := SYSTIMESTAMP;

l\_interval1 INTERVAL YEAR (4) TO MONTH := '2011-11';

l\_interval2 INTERVAL DAY (2) TO SECOND := '15 00:30:44';

BEGIN

null;

END;

Working with intervals and time stamps with time zones can be very complicated; relatively few developers will need these more advanced features. This article focuses on the core DATE and TIMESTAMP types, along with the most commonly used built-in functions.

**Choosing a datatype.** With such an abundance of riches, how do you decide which of these date-and-time datatypes to use? Here are some guidelines:

* Use one of the TIMESTAMP types if you need to track time down to a fraction of a second.
* You can, in general, use TIMESTAMP in place of DATE. A time stamp that does not contain subsecond precision takes up 7 bytes of storage, just as a DATE datatype does. When your time stamp does contain subsecond data, it takes up 11 bytes of storage.
* Use TIMESTAMP WITH TIME ZONE if you need to keep track of the session time zone in which the data was entered.
* Use TIMESTAMP WITH LOCAL TIME ZONE if you want the database to automatically convert a time between the database and session time zones.
* Use DATE when it’s necessary to maintain compatibility with an existing application written before any of the TIMESTAMP datatypes were introduced.
* Use datatypes in your PL/SQL code that correspond to, or are at least compatible with, the underlying database tables. Think twice, for example, before reading a TIMESTAMP value from a table into a DATE variable, because you might lose information (in this case, the fractional seconds and perhaps the time zone).

**Getting the current date and time.** PL/SQL developers often need to retrieve and work with the current date and time. Most developers use the classic SYSDATE function, but Oracle Database now offers several functions to provide variations of this information, as shown in Table 1.

|  |  |  |
| --- | --- | --- |
| Function | Time Zone | Datatype Returned |
| CURRENT\_DATE | Session | DATE |
| CURRENT\_TIMESTAMP | Session | TIMESTAMP WITH TIME ZONE |
| LOCALTIMESTAMP | Session | TIMESTAMP |
| SYSDATE | Database server | DATE |
| SYSTIMESTAMP | Database server | TIMESTAMP WITH TIME ZONE |

**Table 1:** SYSDATE and other options for working with the current date and time

Listing 2 displays the values returned by calls to SYSDATE and SYSTIMESTAMP.

**Code Listing 2:** Calls to SYSDATE and SYSTIMESTAMP and the returned values

BEGIN

DBMS\_OUTPUT.put\_line (SYSDATE);

DBMS\_OUTPUT.put\_line (SYSTIMESTAMP);

DBMS\_OUTPUT.put\_line (SYSDATE - SYSTIMESTAMP);

END;

/

Here is the output:

07-AUG-11

07-AUG-11 08.46.16.379000000 AM -05:00

-000000000 00:00:00.379000000

Because I have passed dates and time stamps to DBMS\_OUTPUT.PUT\_LINE, Oracle Database implicitly converts them to strings, using the default format masks for the database or the session (as specified by the National Language Settings NLS\_DATE\_FORMAT parameter). A default installation of Oracle Database sets the default DATE format to DD-MON-YYYY. The default TIMESTAMP format includes both the date offset and the time zone offset.

Note that it is possible to perform date arithmetic: I subtract the value returned by SYSTIMESTAMP from the value returned by SYSDATE. The result is an *interval* that is *very close* (but not quite equal) to zero.

**Converting dates to strings and strings to dates.** As with TO\_CHAR for numbers, you use another version of the TO\_CHAR function to convert a date or a time stamp to a string. And, again as with numbers, Oracle Database offers a large set of format elements to help you tweak that string so it appears exactly as you need it. Here are some examples:

1. Use TO\_CHAR without a format mask. If you do not include a format mask, the string returned by TO\_CHAR will be the same as that returned when Oracle Database performs an implicit conversion:

BEGIN

DBMS\_OUTPUT.put\_line (

TO\_CHAR (SYSDATE));

DBMS\_OUTPUT.put\_line (

TO\_CHAR (SYSTIMESTAMP));

END;

/

07-AUG-11

07-AUG-11 08.55.00.470000000 AM -05:00

1. Use TO\_CHAR to display the full names of both the day and the month in the date:

BEGIN

DBMS\_OUTPUT.put\_line (

TO\_CHAR (SYSDATE,

'Day, DDth Month YYYY'));

END;

/

Sunday , 07TH August 2011

Note: The language used to display these names is determined by the NLS\_DATE\_LANGUAGE setting, which can also be specified as the third argument in the call to TO\_CHAR, as in

BEGIN

DBMS\_OUTPUT.put\_line (

TO\_CHAR (SYSDATE,

'Day, DDth Month YYYY',

'NLS\_DATE\_LANGUAGE=Spanish'));

END;

/

Domingo , 07TH Agosto 2011

|  |
| --- |
| **Answers to the Challenge**    Here are the answers to the PL/SQL Challenge questions in last issue’s “Working with Numbers in PL/SQL” article:  **Answer 1:** The plch\_ceil\_and\_floor function always returns either 1 or 0: 0 if the number passed to the function is an integer, 1 otherwise.  **Answer 2:** (a) and (b) are correct; (c) is incorrect.  For full explanations of both of these answers, visit [plsqlchallenge.com](http://www.plsqlchallenge.com/), register or log in, and click the **Closed/Taken** tab in Play a Quiz, or go to [bit.ly/r1SwvP](http://bit.ly/r1SwvP). |

1. Use TO\_CHAR to display the full names of both the day and the month in the date—but without all those extra spaces in the date-as-string. Oracle Database, by default, pads the string with spaces to match the maximum length of the day or the month. In most situations, you don’t want to include that extra text, and Oracle Database offers a format element modifier, FM, to control blank and zero padding. In the following block, I prefix the format mask with FM and remove the 0 (before 7) and extra spaces after August:
3. BEGIN
4. DBMS\_OUTPUT.put\_line (
5. TO\_CHAR (SYSDATE,
6. 'FMDay, DDth Month YYYY'));
7. END;
8. /
9. Sunday, 7TH August 2011

You can also use the format mask to extract just a portion of, or information about, the date, as shown in the following examples:

1. What quarter is it?
2. TO\_CHAR (SYSDATE, 'Q')

1. What is the day of the year (1-366) for today’s date?
2. TO\_CHAR (SYSDATE, 'DDD')

1. What are the date *and time* of a DATE variable? (This is a very common requirement, because the default format mask for a date does *not* include the time component, which means that asking DBMS\_OUTPUT.PUT\_LINE to display a date leaves out the time.)

BEGIN

DBMS\_OUTPUT.put\_line (

TO\_CHAR (SYSDATE,

'YYYY-MM-DD HH24:MI:SS'));

END;

/

You can also use EXTRACT to extract and return the value of a specified element of a date. For example

1. What year is it?

EXTRACT (YEAR FROM SYSDATE)

1. What is the day for today’s date?

EXTRACT (DAY FROM SYSDATE)

To convert a string to a date, use the TO\_DATE or the TO\_TIMESTAMP built-in function. Provide the string and Oracle Database returns a date or a time stamp, using the default format mask for the session:

DECLARE

l\_date DATE;

BEGIN

l\_date := TO\_DATE ('12-JAN-2011');

END ;

If the string you provide does not match the default format, Oracle Database will raise an exception:

DECLARE

l\_date DATE;

BEGIN

l\_date := TO\_DATE ('January 12 2011');

END;

/

ORA-01858: a non-numeric character was

found where a numeric was expected

You should not assume that the literal value you provide in your call to TO\_DATE matches the default format. What if the format changes over time? Instead, always provide a format mask when converting strings to dates, as in

l\_date := TO\_DATE ('January 12 2011',

'Month DD YYYY');

**Date truncation.** Use the TRUNC built-in function to truncate a date to the specified unit of measure. The most common use of TRUNC is TRUNC (date)—without any format mask specified. In this case, TRUNC simply sets the time to 00:00:00. You can also use TRUNC to easily obtain the first day in a specified period. Here are some TRUNC examples:

1. Set l\_date to today’s date, but with the time set to 00:00:00:

l\_date := TRUNC (SYSDATE);

1. Get the first day of the month for the specified date:

l\_date := TRUNC (SYSDATE, 'MM');

1. Get the first day of the quarter for the specified date:

l\_date := TRUNC (SYSDATE, 'Q');

1. Get the first day of the year for the specified date:

l\_date := TRUNC (SYSDATE, 'Y');

**Date arithmetic.** Oracle Database enables you to perform arithmetic operations on dates and time stamps in several ways:

* Add a numeric value to or subtract it from a date, as in SYSDATE + 7; Oracle Database treats the number as the number of days.
* Add one date to or subtract it from another, as in l\_hiredate - SYSDATE.
* Use a built-in function to “move” a date by a specified number of months or to another date in a week.

Here are some examples of date arithmetic with a date and a number (assume in all cases that the l\_date variable has been declared as DATE):

1. Set a local variable to tomorrow’s date:

l\_date := SYSDATE + 1;

1. Move back one hour:

l\_date := SYSDATE - 1/24;

1. Move ahead 10 seconds:

l\_date := SYSDATE + 10 / (60 \* 60 \* 24);

When you add one date to or subtract it from another, the result is the number of days between the two. As a result, executing this block:

DECLARE

l\_date1 DATE := SYSDATE;

l\_date2 DATE := SYSDATE + 10;

BEGIN

DBMS\_OUTPUT.put\_line (

l\_date2 - l\_date1);

DBMS\_OUTPUT.put\_line (

l\_date1 - l\_date2);

END;

returns the following output:

10

-10

|  |
| --- |
| **Next Steps**    **LEARN more about** [**PL/SQL datatypes**](http://bit.ly/nrpGAw) [**functions**](http://bit.ly/qbrR7I)  **DOWNLOAD**[**Oracle Database 11*g***](http://www.oracle.com/technetwork/database/enterprise-edition/downloads/index-092322.html)  **TEST**[**your PL/SQL knowledge**](http://www.plsqlchallenge.com/)  **READ**[**PL/SQL 101, Parts 1-4**](http://www.oracle.com/technetwork/issue-archive/index-087690.html) |

And the following function can be used to compute the age of a person, assuming that the person’s correct birth date is passed as the function’s only argument:

CREATE OR REPLACE FUNCTION

your\_age (birthdate\_in IN DATE)

RETURN NUMBER

IS

BEGIN

RETURN SYSDATE -

birthdate\_in;

END your\_age;

Oracle Database offers several built-in functions for shifting a date by the requested amount or finding a date:

* ADD\_MONTHS—adds the specified number of months to or subtracts it from a date (or a time stamp)
* NEXT\_DAY—returns the date of the first weekday named in the call to the function
* LAST\_DAY—returns the date of the last day of the month of the specified date

Here are some examples that use these built-in functions:

1. Move ahead one month:

l\_date := ADD\_MONTHS (SYSDATE, 1);

1. Move backward three months:

l\_date := ADD\_MONTHS (SYSDATE, -3);

1. Starting with the last day of January, move ahead one month. Starting from a different date, go back one month. Starting with the last day of February, go back one month. Listing 3 shows three different calls to the ADD\_MONTHS function along with the results.

**Code Listing 3:** Calls to ADD\_MONTHS

BEGIN

DBMS\_OUTPUT.put\_line (

ADD\_MONTHS (TO\_DATE ('31-jan-2011', 'DD-MON-YYYY'), 1));

DBMS\_OUTPUT.put\_line (

ADD\_MONTHS (TO\_DATE ('27-feb-2011', 'DD-MON-YYYY'), -1));

DBMS\_OUTPUT.put\_line (

ADD\_MONTHS (TO\_DATE ('28-feb-2011', 'DD-MON-YYYY'), -1));

END;

Here is the output:

28-FEB-11

27-JAN-11

31-JAN-11

You might be surprised at the third date in Listing 3. The first date (28 February) makes perfect sense. There is no 31st day in February, so Oracle Database returns the last day of the month. The second call to ADD\_MONTHS moves the date from 27 February to 27 January: exactly one month’s change. But in the third call to ADD\_MONTHS, Oracle Database notices that 28 February is the last day of the month, so it returns the last day of the month specified by the second argument.

1. Find the next Saturday after today’s date:

l\_date := NEXT\_DAY (SYSDATE, 'SAT');  
-- or  
l\_date := NEXT\_DAY (SYSDATE, 'SATURDAY');

The second argument must be a day of the week in the date language of your session (specified by NLS\_DATE\_LANGUAGE), provided as either the full name or the abbreviation. The returned date has the same time component as the date.

**Bad Things Happen—Even in Good Programs**

Now that you have a solid foundation in working with key datatypes such as strings, numbers, and dates, I will switch focus in the next article of this series to an in-depth introduction to exceptions: how they can be raised and how you can handle them.

|  |
| --- |
| **Take the Challenge!**  Each PL/SQL 101 article offers a quiz to test your knowledge of the information provided in the article. The quiz questions are shown below and also at PL/SQL Challenge ([plsqlchallenge.com](http://www.plsqlchallenge.com/)), a Website that offers online quizzes for the PL/SQL language. You can read and take the quiz here in *Oracle Magazine* and then check your answers in the next issue. If, however, you take the quiz at PL/SQL Challenge, you will be entered into a raffle to win an e-book from O’Reilly Media ([oreilly.com](http://www.oreilly.com/)).  **Question 1**  Oracle Database provides a function for returning the date of the last day of the month. It does not, however, provide a function for returning the date of the first day. Which of the following can be used to do this?     1. CREATE OR REPLACE FUNCTION plch\_first\_day (date\_in IN DATE)    RETURN DATE IS BEGIN    RETURN TRUNC (date\_in); END; /      1. CREATE OR REPLACE FUNCTION plch\_first\_day (date\_in IN DATE)    RETURN DATE IS BEGIN    RETURN TRUNC (date\_in, 'MM'); END; /      1. CREATE OR REPLACE FUNCTION plch\_first\_day (date\_in IN DATE)    RETURN DATE IS BEGIN    RETURN TRUNC (date\_in, 'MONTH'); END; /      1. CREATE OR REPLACE FUNCTION plch\_first\_day (date\_in IN DATE)    RETURN DATE IS BEGIN    RETURN TO\_DATE (TO\_CHAR (date\_in, 'YYYY-MM')  || '-01', 'YYYY-MM-DD'); END; /   **Question 2**  Given this declaration section:    DECLARE    c\_format   CONSTANT VARCHAR2 (22)       := 'YYYY-MM-DD HH24:MI:SS' ;    l\_new\_year          DATE       := TO\_DATE (             '2012-01-02 00:00:01'           ,  c\_format);  which of the following blocks offers an exception section so that after that block is executed, the date and time 2012-01-01 00:00:01 will be displayed on the screen?   1. BEGIN    DBMS\_OUTPUT.put\_line (       TO\_CHAR (          l\_new\_year - 24        ,  c\_format)); END; 2. BEGIN    DBMS\_OUTPUT.put\_line (       TO\_CHAR (l\_new\_year - 1              ,  c\_format)); END; 3. BEGIN    DBMS\_OUTPUT.put\_line (       TO\_CHAR (            l\_new\_year          - 24 \* 60 \* 60        ,  c\_format)); END; 4. BEGIN    DBMS\_OUTPUT.put\_line (       TO\_CHAR (            TRUNC (l\_new\_year)          - 1          + 1 / (24 \* 60 \* 60)        ,  c\_format)); END; |