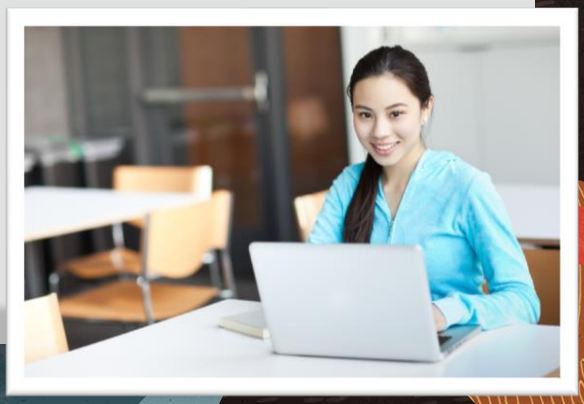

ORACLE
Academy

Database Programming with PL/SQL

2-5

Writing PL/SQL Executable Statements



ORACLE
Academy

Copyright © 2020, Oracle and/or its affiliates. All rights reserved.

Objectives

- This lesson covers the following objectives:
  - Construct accurate variable assignment statements in PL/SQL
  - Construct accurate statements using built-in SQL functions in PL/SQL
  - Differentiate between implicit and explicit conversions of data types
  - Describe when implicit conversions of data types take place
  - List the drawbacks of implicit data type conversions
  - Construct accurate statements using functions to explicitly convert data types
  - Construct statements using operators in PL/SQL

Copyright © 2020, Oracle and/or its affiliates. All rights reserved.

Purpose

- We've introduced variables and identifiers
- In this lesson, you build your knowledge of the PL/SQL programming language by writing code to assign variable values
- These values can be literals or values returned by a function
- SQL provides a number of predefined functions that you can use in SQL statements
- Most of these functions are also valid in PL/SQL expressions

Copyright © 2020, Oracle and/or its affiliates. All rights reserved.

Assigning New Values to Variables

- Character and date literals must be enclosed in single quotation marks

```
v_name  := 'Henderson';
v_start_date := '12-Dec-2005';
```

- Statements can continue over several lines

```
v_quote := 'The only thing that we can know is that we know
nothing and that is the highest flight of human reason.';
```

- Numbers can be simple values or scientific notation (2E5 meaning 2x10 to the power of 5 = 200,000)

```
v_my_integer := 100;
v_my_sci_not := 2E5;
```

ORACLE Academy

Copyright © 2020, Oracle and/or its affiliates. All rights reserved.

Assigning a value to a variable: the variable must always be on the left side of the assignment symbol (:=); the value will always be on the right side of the assignment symbol.

v_num := v_count + c_people

v_num is the variable memory location that will be assigned the value in the variable memory location v_count plus the value of the memory location c_people.

SQL Functions in PL/SQL

- You are already familiar with functions in SQL statements, for example:

```
SELECT LAST_DAY(SYSDATE)
 FROM DUAL;
```

- You can also use these functions in PL/SQL procedural statements, for example:

```
DECLARE
 v_last_day DATE;
BEGIN
 v_last_day := LAST_DAY(SYSDATE);
 DBMS_OUTPUT.PUT_LINE(v_last_day);
END;
```

ORACLE Academy

Copyright © 2020, Oracle and/or its affiliates. All rights reserved.

Functions are used as short cuts. Someone already programmed a block of code to accomplish a specific process. These blocks of code are called procedures and functions. Use them to make writing your program easier. Later in this course, you will learn how to write your own procedures and functions.

SQL Functions in PL/SQL

- Functions available in procedural statements:
  - Single-row character
  - Single-row number
  - Date
  - Data-type conversion
  - Miscellaneous functions
- Not available in procedural statements:
  - DECODE (CASE is used instead)
  - Group functions (AVG, MIN, MAX etc. may be used ONLY within a SQL statement)

Copyright © 2020, Oracle and/or its affiliates. All rights reserved.

SQL Functions in PL/SQL

- SQL functions help you to manipulate data; they fall into the following categories:
  - Character
  - Number
  - Date
  - Conversion
  - Miscellaneous

Copyright © 2020, Oracle and/or its affiliates. All rights reserved.

Character Functions

- Valid character functions in PL/SQL include:

ASCII	LENGTH	RPAD
CHR	LOWER	RTRIM
CONCAT	LPAD	SUBSTR
INITCAP	LTRIM	TRIM
INSTR	REPLACE	UPPER

- This is not an exhaustive list
- Refer to the Oracle documentation for the complete list

Copyright © 2020, Oracle and/or its affiliates. All rights reserved.

Examples of Character Functions

- Get the length of a string:

```
v_desc_size  INTEGER(5);
v_prod_description VARCHAR2(70):='You can use this product
with your radios for higher frequency';

-- get the length of the string in prod_description
v_desc_size:= LENGTH(v_prod_description);
```

- Convert the name of the country capitol to upper case:

```
v_capitol_name:= UPPER(v_capitol_name);
```

- Concatenate the first and last names:

```
v_emp_name:= v_first_name||' '||v_last_name;
```


Copyright © 2020, Oracle and/or its affiliates. All rights reserved.

Number Functions

- Valid number functions in PL/SQL include:

ABS	EXP	ROUND
ACOS	LN	SIGN
ASIN	LOG	SIN
ATAN	MOD	TAN
COS	POWER	TRUNC

- This is not an exhaustive list
- Refer to the Oracle documentation for the complete list

Copyright © 2020, Oracle and/or its affiliates. All rights reserved.

Examples of Number Functions

- Get the sign of a number:

```
DECLARE
 v_my_num BINARY_INTEGER := -56664;
BEGIN
 DBMS_OUTPUT.PUT_LINE(SIGN(v_my_num));
END;
```

- Round a number to 0 decimal places:

```
DECLARE
  v_median_age NUMBER(6,2);
BEGIN
  SELECT meian_age INTO v_median_age
  FROM countries
  WHERE country_id = 27;
  DBMS_OUTPUT.PUT_LINE(ROUND(v_median_age,0));
END;
```


Copyright © 2020, Oracle and/or its affiliates. All rights reserved.

Date Functions

- Valid date functions in PL/SQL include:

ADD_MONTHS	MONTHS_BETWEEN
CURRENT_DATE	ROUND
CURRENT_TIMESTAMP	SYSDATE
LAST_DAY	TRUNC

- This is not an exhaustive list
- Refer to the Oracle documentation for the complete list

Copyright © 2020, Oracle and/or its affiliates. All rights reserved.

Examples of Date Functions

- Add months to a date:

```
DECLARE
 v_new_date DATE;
 v_num_months NUMBER := 6;
BEGIN
 v_new_date := ADD_MONTHS(SYSDATE,v_num_months);
 DBMS_OUTPUT.PUT_LINE(v_new_date);
END;
```

- Calculate the number of months between two dates:

```
DECLARE
  v_no_months PLS_INTEGER :=0;
BEGIN
  v_no_months := MONTHS_BETWEEN('31-Jan-2006','31-May-2005');
  DBMS_OUTPUT.PUT_LINE(v_no_months);
END;
```


Copyright © 2020, Oracle and/or its affiliates. All rights reserved.

Data-Type Conversion

- In any programming language, converting one data type to another is a common requirement
- PL/SQL can handle such conversions with scalar data types
- Data-type conversions can be of two types:
  - Implicit conversions
  - Explicit conversions

ORACLE Academy

Copyright © 2020, Oracle and/or its affiliates. All rights reserved.

Implicit Conversions

- In implicit conversions, PL/SQL attempts to convert data types dynamically if they are mixed in a statement
- Implicit conversions can happen between many types in PL/SQL, as illustrated by the following chart

	DATE	LONG	NUMBER	PLS_INTEGER	VARCHAR2
DATE	N/A	X			X
LONG		N/A			X
NUMBER		X	N/A	X	X
PLS_INTEGER		X	X	N/A	X
VARCHAR2	X	X	X	X	N/A

ORACLE Academy

Copyright © 2020, Oracle and/or its affiliates. All rights reserved.

Whenever PL/SQL detects that a conversion is necessary, it attempts to change the values as required to perform the operation.

In the chart, the cells marked 'X' show which implicit conversions can be done.

For this course, we will focus on implicit conversions between:

Characters and numbers

Characters and dates

For more information about the above chart, refer to “Converting PL/SQL Data Types” in the PL/SQL User’s Guide and Reference.

Example of Implicit Conversion

- In this example, the variable v_sal_increase is of type VARCHAR2
- While calculating the total salary, PL/SQL first converts v_sal_increase to NUMBER and then performs the operation
- The result of the operation is the NUMBER type

```
DECLARE
 v_salary  NUMBER(6) := 6000;
 v_sal_increase VARCHAR2(5) := '1000';
 v_total_salary v_salary%TYPE;
BEGIN
  v_total_salary := v_salary + v_sal_increase;
DBMS_OUTPUT.PUT_LINE(v_total_salary);
END;
```


Copyright © 2020, Oracle and/or its affiliates. All rights reserved.

Drawbacks of Implicit Conversions

- At first glance, implicit conversions might seem useful; however, there are several drawbacks:
  - Implicit conversions can be slower
  - When you use implicit conversions, you lose control over your program because you are making an assumption about how Oracle handles the data
  - If Oracle changes the conversion rules, then your code can be affected
  - Code that uses implicit conversion is harder to read and understand

Copyright © 2020, Oracle and/or its affiliates. All rights reserved.

Drawbacks of Implicit Conversions

- Additional drawbacks:
  - Implicit conversion rules depend upon the environment in which you are running
    - For example, the date format varies depending on the language setting and installation type
    - Code that uses implicit conversion might not run on a different server or in a different language
  - It is strongly recommended that you AVOID allowing SQL or PL/SQL to perform implicit conversions on your behalf
  - You should use conversion functions to guarantee that the right kinds of conversions take place

Copyright © 2020, Oracle and/or its affiliates. All rights reserved.

Drawbacks of Implicit Conversions

- It is the programmer's responsibility to ensure that values can be converted
- For instance, PL/SQL can convert the CHAR value '02-Jun-1992' to a DATE value, but cannot convert the CHAR value 'Yesterday' to a DATE value
- Similarly, PL/SQL cannot convert a VARCHAR2 value containing alphabetic characters to a NUMBER value

Valid?	Statement
Yes	v_new_date DATE := '02-Jun-1992';
No	v_new_date DATE := 'Yesterday';
Yes	v_my_number NUMBER := '123';
No	v_my_number NUMBER := 'abc';

Copyright © 2020, Oracle and/or its affiliates. All rights reserved.

Explicit Conversions

- Explicit conversions convert values from one data type to another by using built-in functions
- Examples of conversion functions include:

TO_NUMBER()	ROWIDTONCHAR()
TO_CHAR()	HEXTORAW()
TO_CLOB()	RAWTOHEX()
CHARTOROWID()	RAWTONHEX()
ROWIDTOCHAR()	TO_DATE()

Copyright © 2020, Oracle and/or its affiliates. All rights reserved.

Examples of Explicit Conversions

- TO_CHAR

```
BEGIN
 DBMS_OUTPUT.PUT_LINE(TO_CHAR(SYSDATE,'Month YYYY'));
END;
```

- TO_DATE

```
BEGIN
 DBMS_OUTPUT.PUT_LINE(TO_DATE('April-1999','Month-YYYY'));
END;
```


Copyright © 2020, Oracle and/or its affiliates. All rights reserved.

Examples of Explicit Conversions

- TO_NUMBER

```
DECLARE
 v_a VARCHAR2(10) := '-123456';
 v_b VARCHAR2(10) := '+987654';
 v_c PLS_INTEGER;
BEGIN
 v_c := TO_NUMBER(v_a) + TO_NUMBER(v_b);
 DBMS_OUTPUT.PUT_LINE(v_c);
END;
```

- Note that the DBMS_OUTPUT.PUT_LINE procedure expects an argument with a character type such as VARCHAR2
- Variable v_c is a number, therefore we should explicitly code: DBMS_OUTPUT.PUT_LINE(TO_CHAR(v_c));

Copyright © 2020, Oracle and/or its affiliates. All rights reserved.

Data Type Conversion Examples

- Example #1

```
v_date_of_joining DATE := '02-Feb-2014';
```

- Example #2

```
v_date_of_joining DATE := 'February 02, 2014';
```

- Example #3

```
v_date_of_joining DATE := TO_DATE('February 02, 2014',
 'Month DD, YYYY');
```


Copyright © 2020, Oracle and/or its affiliates. All rights reserved.

The examples in the slide show implicit and explicit conversions of the DATE data type.

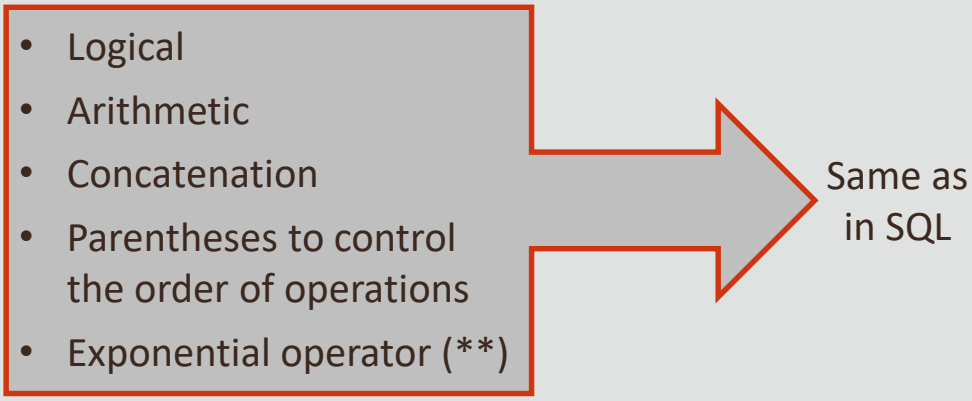
Example #1 - Implicit conversion happens in this case and the date is assigned to v_date_of_joining.

Example #2 - PL/SQL gives you an error because the date that is being assigned is not in the default format.

Example #3 - This is how it should be done. Use the TO_DATE function to explicitly convert the given date in a particular format and assign it to the DATE data type variable date_of_joining.

Operators in PL/SQL

- The operations within an expression are performed in a particular order depending on their precedence (priority)



Copyright © 2020, Oracle and/or its affiliates. All rights reserved.

Operators in PL/SQL

- The following table shows the default order of operations from high priority to low priority:

Operator	Operation
**	Exponentiation
+, -	Identity, negation
*, /	Multiplication, division
+, -, ||	Addition, subtraction, concatenation
=, <, >, <=, >=, <>, !=, ~=, ^=, IS NULL, LIKE, BETWEEN, IN	Comparison
NOT	Logical negation
AND	Conjunction
OR	Inclusion

Copyright © 2020, Oracle and/or its affiliates. All rights reserved.

Operators in PL/SQL Examples

- Increment the counter for a loop

```
v_loop_count := v_loop_count + 1;
```

- Set the value of a Boolean flag

```
v_good_sal := v_sal BETWEEN 50000 AND 150000;
```

- Validate whether an employee number contains a value

```
v_valid      := (v_empno IS NOT NULL);
```

ORACLE Academy

Copyright © 2020, Oracle and/or its affiliates. All rights reserved.

The second example is particularly elegant when you consider the same result could be obtained by the following code:

```
IF v_sal BETWEEN 50000 AND 150000 THEN
 v_good_sal := TRUE;
ELSE
 v_good_sal := FALSE;
END IF;
```

Terminology

- Key terms used in this lesson included:
  - Explicit conversion
  - Implicit conversion

Copyright © 2020, Oracle and/or its affiliates. All rights reserved.

- Explicit conversion – Converts values from one data type to another by using built-in functions.
- Implicit conversion – Converts data types dynamically if they are mixed in a statement.

Summary

- In this lesson, you should have learned how to:
  - Construct accurate variable assignment statements in PL/SQL
  - Construct accurate statements using built-in SQL functions in PL/SQL
  - Differentiate between implicit and explicit conversions of data types
  - Describe when implicit conversions of data types take placeList the drawbacks of implicit data type conversions
  - Construct accurate statements using functions to explicitly convert data types
  - Construct statements using operators in PL/SQL

Copyright © 2020, Oracle and/or its affiliates. All rights reserved.


ORACLE
Academy