Table of Contents

[CHAPTER 3: VARIABLE, DATA TYPE AND EXPRESSION 2](#_Toc416695786)

[Theory 2](#_Toc416695787)

[AIM 6](#_Toc416695788)

[Lab Exercise 3: VARIABLE, DATA TYPE AND EXPRESSION 7](#_Toc416695789)

[1. Variable Declaration 8](#_Toc416695790)

[2. Variable Assignment 16](#_Toc416695791)

[3. Expressions 19](#_Toc416695792)

[SUMMARY 24](#_Toc416695793)

[REFERENCES 25](#_Toc416695794)

[INDEX 26](#_Toc416695795)

# CHAPTER 3: VARIABLE, DATA TYPE AND EXPRESSION

## Theory

PL/SQL programmers surely need to define variables. Variables should be defined in Declaration part in PL/SQL block. In general, any variable is defined in the following rule:

**:=** Initial Value/Expression

**NOT NULL**

Data Type

Variable Name

Mandatory items in the expression are Variable Name and Data Type only. NOT NULL constraint is an optional item that prevents assigning NULL to this variable. Programmer may assign a value to the variable right after declaration by using ':=' operator followed by value or expression. Assigning value in a declaration is optional item.

Constant is a special kind of variable with additional 3 constraints:

1. A keyword **CONSTANT** should be placed after variable name.
2. Initial value is mandatory.
3. The initial value can't be change after initialization.

There are two main data types in PL/SQL: (1) Scalar data types, (2) Composite data types. Composite data types will be covered in next chapters. Scalar data types store values with no internal components. They are indivisible. Scalar data types consist of two main categories: (1) base types and (2) subtypes.

POSITIVE

SIGNTYPE

SIMPLE\_INTEGER

NATURALN

N

POSITIVEN

N

NATURAL

Composite Data Type

REF CURSOR

User Defined Data type

BOOLEAN

SQL Data Types

PLS\_INTEGER

Scalar Data Type

PL/SQL Data types

Base types are the parents of subtypes. Many base types and their subtypes are predefined in STANDARD package; which is implicitly declared in any PL/SQL block. The PL/SQL scalar data types are:

* **The SQL data types**: All SQL data types such as CHAR, VARCHAR, NUMBER, DATE, LONG, etc can be defined in PL/SQL. However, the maximum size of these variables is much less in PL/SQL. For example, CHAR data type has a maximum size 32,767 bytes in SQL while it has a maximum size 2,000 bytes in PL/SQL.
* **BOOLEAN:** A BOOLEAN data type is a special data type that can have just three values (TRUE, FALSE, NULL). Be careful when using BOOLEAN data type; It cannot be stored in TABLE, printed to SQL\*Plus console nor used in SQL query.
* **REF CURSOR:** Cursors will be covered in next chapters.
* **PLS\_INTEGER/BINARY\_INTEGER:** signed integer stored in 32 bits. PLS\_INTEGER is faster and required less storage than NUMBER. When it possible, use PLS\_INTEGER for arithmetic calculation.
* **User-defined subtypes:** PL/SQL allows you to define your subtypes based on PL/SQL base types, subtypes, or even your pre-defined subtypes. Usually, user-defined subtypes are used to enforce constraints on pre-defined base types; for example, NOT NULL, Precision, Scale, and Range of values.

Functions, constraints and operators which can be done to base types implicitly applied to its subtypes. For example, plus operation (+) can be used for PL\_INTEGER base type. Thus, it can be implicitly used for its subtypes POSITIVE, NATURAL and others.

Variable assignment can be done using:

1. Assignment operator [ := ].
2. SELECT INTO or FECTH statement
3. Passing variable as OUT or IN OUT parameter.

Expressions are written with two main objects:

1. Operand(s)
2. Operator

Operand  
V1

Operator  
+

Operand  
 V2

**Unary operator** takes one operand only; for example, negation operator (- v). **Binary operator** takes two operands; for example addition operator ( v1 + v2). Some operator can be done on CHAR/VARCHAR data type only such as concatenation operator ( **||** ), others can be done on PLS\_INTEGER data type such as addition operator ( **+** ). When an expression contains more than one operator, the precedence will be for the operator that has a higher precedence according to the following table:

|  |  |
| --- | --- |
| **Operator** | **Operation** |
| **\*\*** | exponentiation |
| **+, -** | identity, negation |
| **\*, /** | multiplication, division |
| **+, -, ||** | addition, subtraction, concatenation |
| **=, <, >, <=, >=, <>, !=, ~=, ^=, IS NULL, LIKE, BETWEEN, IN** | comparison |
| **NOT** | negation |
| **AND** | conjunction |
| **OR** | inclusion |

Operators with equal precedence are evaluated in **NO** particular order. To force your preferred execution order, use parentheses **()**.

## AIM

The AIM of the following exercise is to demonstrate how to work with variables in Oracle PL/SQL block

The steps involved will include:

* Variable Declaration
* Variable Assignment
* Expressions

In general, lab exercises are done in sequential order. Thus, it is assumed that you successfully completed the previous labs. However, not all previous labs are required. Please be sure to run the following lab before proceeding:

* Installing Oracle Database 12c.

Estimated Completion Time:

25 minutes

# Lab Exercise 3: VARIABLE, DATA TYPE AND EXPRESSION

|  |
| --- |
|  |

## Variable Declaration

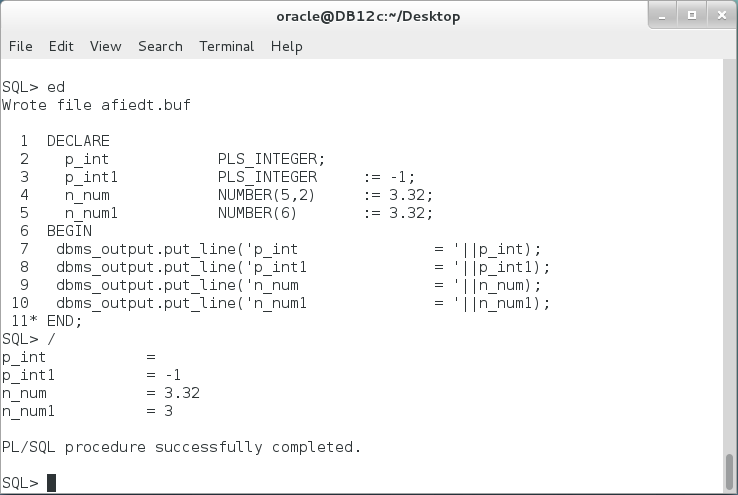
**Step 1:** Open the Terminal, open SQL\*Plus console and connect to hr schema.

|  |  |
| --- | --- |
| Command | Description |
| sqlplus | Open SQL\*Plus console. |
| hr/oracle | connect to **hr** schema. |

****

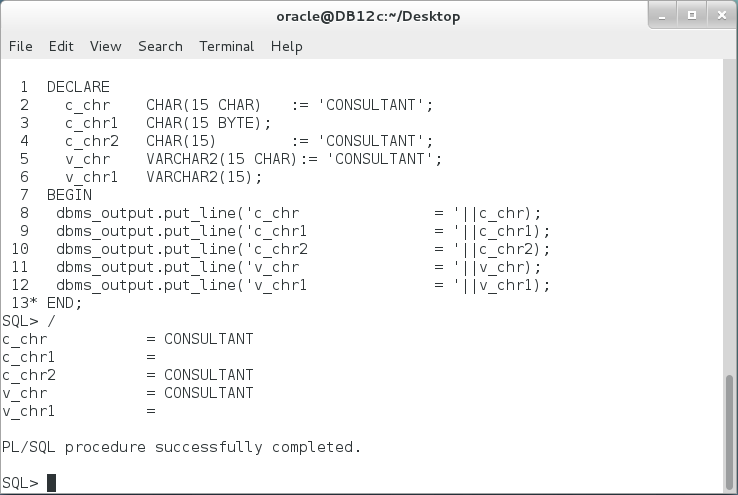
**Step 2:** Execute the following block:

|  |  |
| --- | --- |
| Command | Description |
| DECLARE |  |
| p\_int PLS\_INTEGER; | Define PLS\_INTEGER variable without initial value. |
| p\_int1 PLS\_INTEGER := -1; | with initial value |
| n\_num NUMBER(6,2) := 3.32; | Define NUMBER variable with Precision (6) and Scale(2). So, the number can have up to 3 digits before the decimal and 2 digits after decimal. |
| n\_num1 NUMBER(6) := 3.32; | NUMBER variable with Precision (6). |
| BEGIN |  |
| dbms\_output.put\_line('p\_int = '||p\_int); | Without initialization, a variable will set to **NULL**. |
| dbms\_output.put\_line('p\_int1 = '||p\_int1); | PLS\_INTEGER can store negative number |
| dbms\_output.put\_line('n\_num = '||n\_num); | NUMBER with decimal |
| dbms\_output.put\_line('n\_num1 = '||n\_num1); | NUMBER without decimal will **ignore** any number after decimal. |
| END; |  |
| / |  |

****

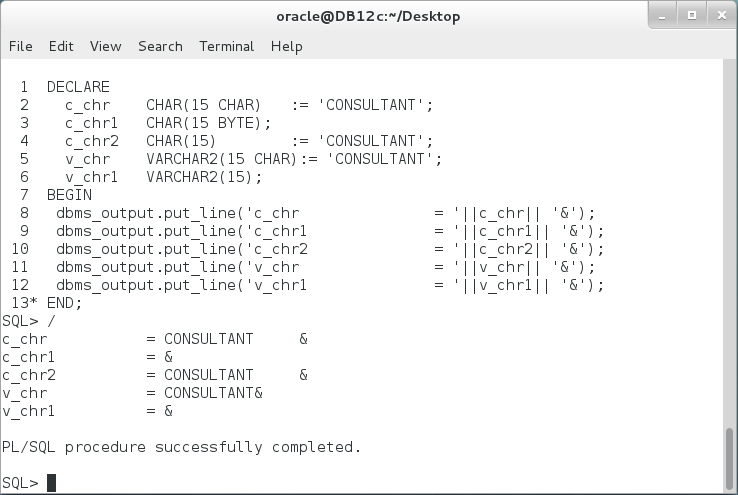
**Step 3:** Execute the following PL/SQL Block:

|  |  |
| --- | --- |
| Line | Description |
| DECLARE |  |
| c\_chr CHAR(15 CHAR) := 'CONSULTANT'; | Reserve 15 characters even if it not fully filled. |
| c\_chr1 CHAR(15 BYTE); | Reserve 15 bytes. It is the same as the previous definition for One-byte-character languages. For Multi-byte languages, it is different. |
| c\_chr2 CHAR(15) := 'CONSULTANT'; | Or simply define it with 15 size only. |
| v\_chr VARCHAR2(15 CHAR):= 'CONSULTANT'; | Reserve **at most** 15 characters. v\_chr variable will store only the size of the value if it is value less than 15 character. |
| v\_chr1 VARCHAR2(15); | Define simply with 15 size. |
| BEGIN |  |
| dbms\_output.put\_line('c\_chr = '||c\_chr); | Explain the output? |
| dbms\_output.put\_line('c\_chr1 = '||c\_chr1); |
| dbms\_output.put\_line('c\_chr2 = '||c\_chr2); |
| dbms\_output.put\_line('v\_chr = '||v\_chr); |
| dbms\_output.put\_line('v\_chr1 = '||v\_chr1); |
| END; |  |



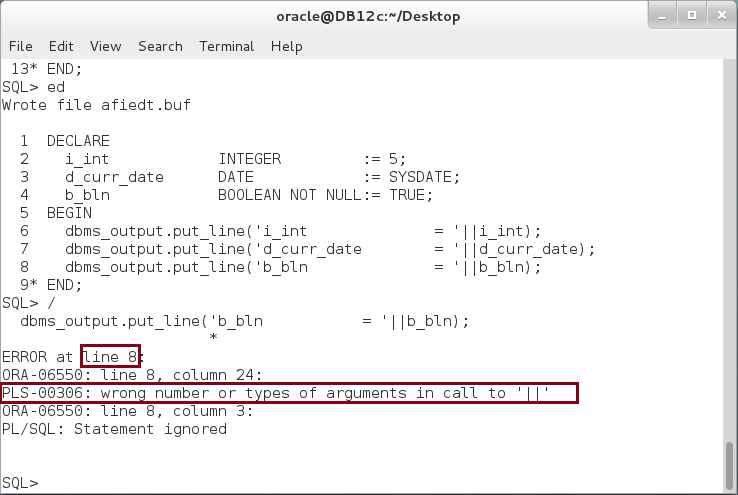
**Step 4:** Modify the previous PL/SQL Block as the following.

|  |  |
| --- | --- |
| Line | Description |
| DECLARE |  |
| c\_chr CHAR(15 CHAR) := 'CONSULTANT'; |  |
| c\_chr1 CHAR(15 BYTE); |  |
| c\_chr2 CHAR(15) := 'CONSULTANT'; |  |
| v\_chr VARCHAR2(15 CHAR):= 'CONSULTANT'; |  |
| v\_chr1 VARCHAR2(15); |  |
| BEGIN |  |
| dbms\_output.put\_line('c\_chr = '||c\_chr **|| '&'**); | **Re-explain the output?** |
| dbms\_output.put\_line('c\_chr1 = '||c\_chr1**||'&'**); |
| dbms\_output.put\_line('c\_chr2 = '||c\_chr2**||'&'**); |
| dbms\_output.put\_line('v\_chr = '||v\_chr **||'&'**); |
| dbms\_output.put\_line('v\_chr1 = '||v\_chr1**||'&'**); |
| END; |  |



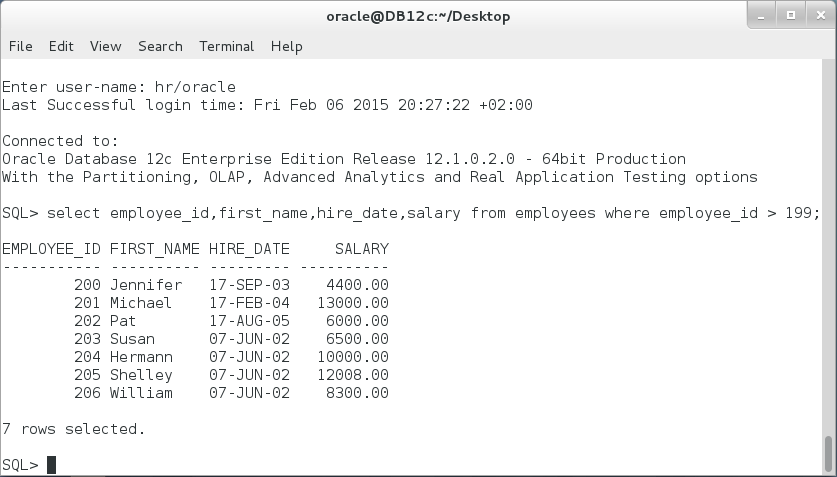
**Step 5:** Execute the following PL/SQL block:

|  |  |
| --- | --- |
| Line | Description |
| DECLARE | Define INTEGER variable |
| i\_int INTEGER := 5; |
| d\_curr\_date DATE := SYSDATE; | Define DATE variable |
| b\_bln BOOLEAN NOT NULL:= TRUE; | BOOLEAN variable |
| BEGIN |  |
| dbms\_output.put\_line('i\_int = '||i\_int); | There is an error in the output. **Explain and fix?** |
| dbms\_output.put\_line('d\_curr\_date = '||d\_curr\_date); |
| dbms\_output.put\_line('b\_bln = '||b\_bln); |
| END; |
| / |  |



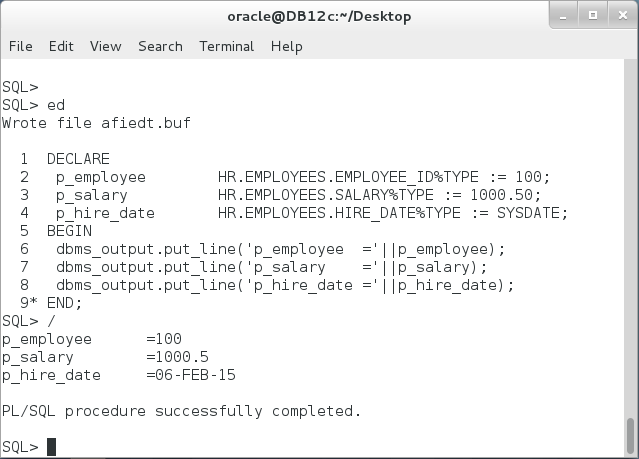
**Step 6:** Execute the following SQL statement:

|  |  |
| --- | --- |
| Command | Description |
| select employee\_id,first\_name,hire\_date,salary  from employees  where employee\_id > 199; | View all employees who's id is greater than 199. |



**Step 7:** Suppose now that you want to manipulate SALARY and HIRE\_DATE columns in your PL/SQL Block. Thus, you need to define your variables' data type the same as the columns. In this case, It is better to use %TYPE attribute. Execute the following PL/SQL block:

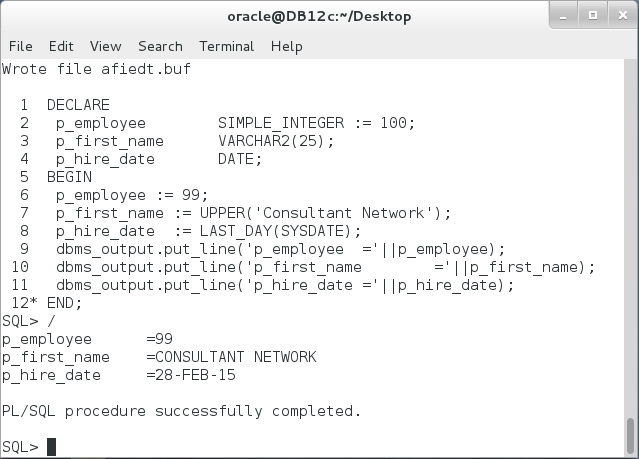
|  |
| --- |
| Line |
| DECLARE |
| p\_employee HR.EMPLOYEES.EMPLOYEE\_ID**%TYPE** := 100; |
| p\_salary HR.EMPLOYEES.SALARY**%TYPE** := 1000.50; |
| p\_hire\_date HR.EMPLOYEES.HIRE\_DATE**%TYPE** := SYSDATE; |
| BEGIN |
| dbms\_output.put\_line('p\_employee ='||p\_employee); |
| dbms\_output.put\_line('p\_salary ='||p\_salary); |
| dbms\_output.put\_line('p\_hire\_date ='||p\_hire\_date); |
| END; |
| / |



## Variable Assignment

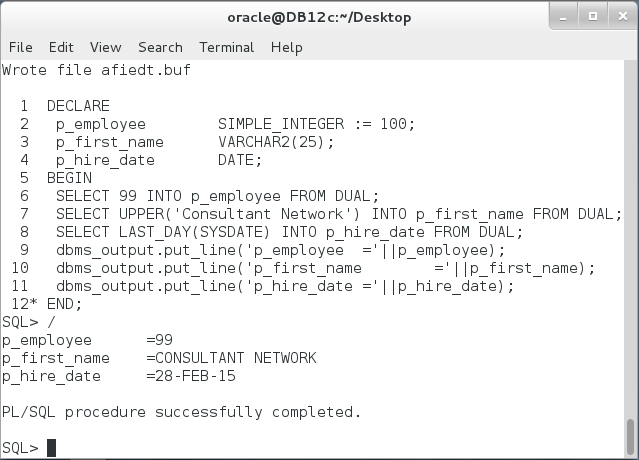
**Step 1:** Variables can be assigned in initialization or using a simple assignment operator [ **:=** ]. Execute the following PL/SQL block:

|  |  |
| --- | --- |
| Line | Description |
| DECLARE |  |
| p\_employee SIMPLE\_INTEGER := 100; | Assign initial value |
| p\_first\_name VARCHAR2(25); |  |
| p\_hire\_date DATE; |  |
| BEGIN |  |
| p\_employee := 99; | Override initial value |
| p\_first\_name := UPPER('Consultant Network'); | Assign variable with a return value of UPPER function. |
| p\_hire\_date := LAST\_DAY(SYSDATE); | Assign variable with a return value of LAST\_DAY function. |
| dbms\_output.put\_line('p\_employee='||p\_employee); |  |
| dbms\_output.put\_line('p\_first\_name='||p\_first\_name); |  |
| dbms\_output.put\_line('p\_hire\_date='||p\_hire\_date); |  |
| END; |  |



**Step 2:** There is another way to assign value to variables in PL/SQL block. Execute the following PL/SQL block.

|  |  |
| --- | --- |
| Command | Description |
| DECLARE |  |
| p\_employee SIMPLE\_INTEGER := 100; |  |
| p\_first\_name VARCHAR2(25); |  |
| p\_hire\_date DATE; |  |
| BEGIN |  |
| SELECT 99 INTO p\_employee FROM DUAL; | Assign value using SELECT INTO.  Note: **DUAL** is a special one-column, one-row table. |
| SELECT UPPER('Consultant Network') INTO p\_first\_name FROM DUAL; |
| SELECT LAST\_DAY(SYSDATE) INTO p\_hire\_date FROM DUAL; |
| dbms\_output.put\_line('p\_employee ='||p\_employee); |  |
| dbms\_output.put\_line('p\_first\_name ='||p\_first\_name); |  |
| dbms\_output.put\_line('p\_hire\_date ='||p\_hire\_date); |  |
| END; |  |

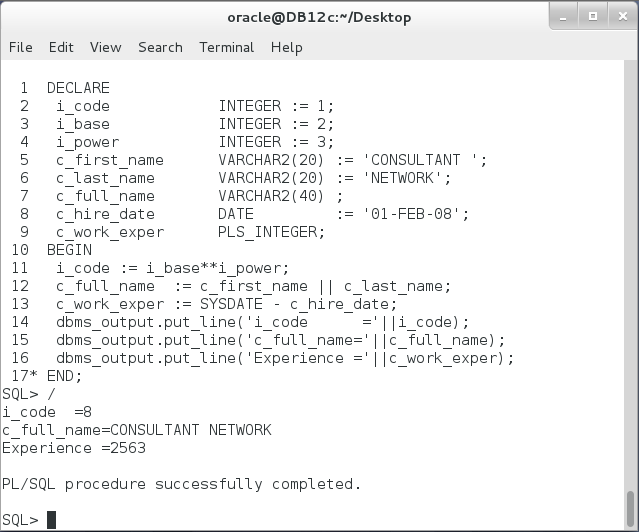


This PL/SQL block does the same as the previous block.

## Expressions

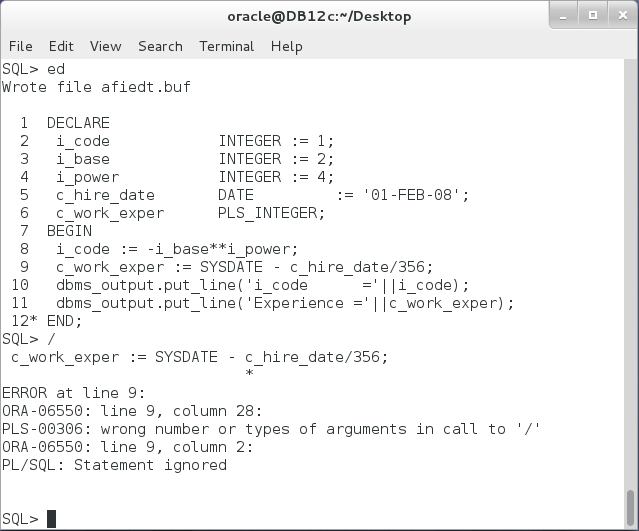
**Step 1:** Execute the following PL/SQL Block:

|  |  |
| --- | --- |
| Command | Description |
| DECLARE |  |
| i\_code INTEGER := 1; |  |
| i\_base INTEGER := 2; |  |
| i\_power INTEGER := 3; |  |
| c\_first\_name VARCHAR2(20) := 'CONSULTANT '; |  |
| c\_last\_name VARCHAR2(20) := 'NETWORK'; |  |
| c\_full\_name VARCHAR2(40) ; |  |
| c\_hire\_date DATE := '01-FEB-08'; |  |
| c\_work\_exper PLS\_INTEGER; |  |
| BEGIN |  |
| i\_code := i\_base\*\*i\_power; | Exponentiation operator. |
| c\_full\_name := c\_first\_name || c\_last\_name; | Concatenation operator. |
| c\_work\_exper := SYSDATE - c\_hire\_date; | Subtraction operator. |
| dbms\_output.put\_line('i\_code='||i\_code); | Explain the output? |
| dbms\_output.put\_line('c\_full\_name='||c\_full\_name); |
| bms\_output.put\_line('Experience='||c\_work\_exper); |
| END; |  |
| / |  |

****

**Step 2:** Update previous PL/SQL block so you attempt to calculate **-2** to the **forth** power and show the **experience in years**, as the following:

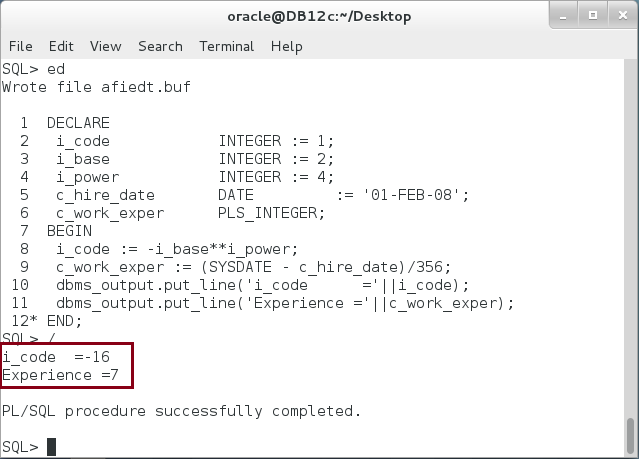
|  |  |
| --- | --- |
| Command | Description |
| DECLARE |  |
| i\_code INTEGER := 1; |  |
| i\_base INTEGER := 2; |  |
| i\_power INTEGER **:= 4**; |  |
| c\_hire\_date DATE := '01-FEB-08'; |  |
| c\_work\_exper PLS\_INTEGER; |  |
| BEGIN |  |
| **i\_code := - i\_base\*\*i\_power;** |  |
| **c\_work\_exper := SYSDATE - c\_hire\_date/356;** |  |
| dbms\_output.put\_line('i\_code ='||i\_code); |  |
| dbms\_output.put\_line('Experience ='||c\_work\_exper); |  |
| END; |  |
| / |  |



PL/SQL block **failed** to run. **Why**?

**Step 3:** Fix the previous block as shown below:

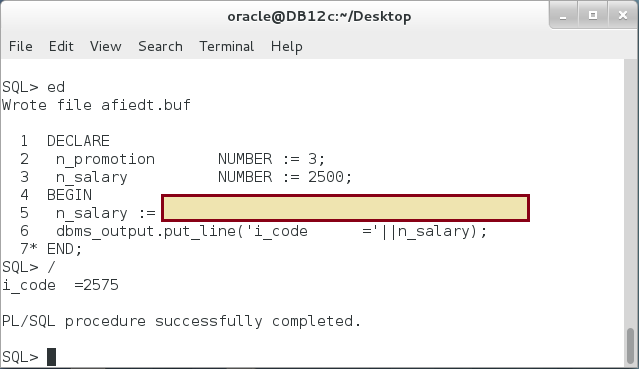
|  |  |
| --- | --- |
| Command | Description |
| DECLARE |  |
| i\_code INTEGER := 1; |  |
| i\_base INTEGER := 2; |  |
| i\_power INTEGER := 4; |  |
| c\_hire\_date DATE := '01-FEB-08'; |  |
| c\_work\_exper PLS\_INTEGER; |  |
| BEGIN |  |
| i\_code := - i\_base\*\*i\_power; | **The output is not as expected, Why?** |
| c\_work\_exper := **(**SYSDATE - c\_hire\_date**)**/356; | **Is it right?** |
| dbms\_output.put\_line('i\_code ='||i\_code); |  |
| dbms\_output.put\_line('Experience ='||c\_work\_exper); |  |
| END; |  |
| / |  |



**Please note**: In the previous block, i\_code should be +16 not -16. Clarify reasons and update the block.

**Step 4:** Complete the following PL/SQL block to compute the new salary, so it is increased by the promotion percentage specified. In this example, the salary was 2500 and the promotion is 3%. The new salary should be 2575 .

|  |
| --- |
| Command |
| DECLARE |
| n\_promotion NUMBER := 3; |
| n\_salary NUMBER := 2500; |
| BEGIN |
| n\_salary := xxxxxxxxxxxxxxxxxxxxxxxxxxxxx |
| dbms\_output.put\_line('i\_code ='||n\_salary); |
| END; |

****

# SUMMARY

Defining variables in PL/SQL is so simple but, perhaps, is not as many other programming languages. Variable declaration can't appear outside the Declaration part. Variable assignment is step to set a value to a variable. Variable assignment may accrue in initialization or declaration or in the block body. The assignment can be done using either a simple [ := ] operator, SELECT INTO statement, or OUT procedure's parameter. The latter will be covered in the next chapters. PL/SQL block is a powerful tool to calculate sophisticated expressions. Oracle data types include two major categories: a) scalar data type and b) composite data type. This chapter shows some features of scalar data types. Scalar data types consists of five different base types: PLS\_INTEGER, BOOLEAN, SQL data types, User-Define types, REF CURSOR. Oracle recommends using PLS\_INTEGER data type, when it is applicable, because it uses hardware arithmetic capabilities which is faster and takes less storage.

After completing this lab exercise, you should be able to define various variables and write expressions with PL/SQL block.

# REFERENCES

* https://docs.oracle.com/database/121/LNPLS/fundamentals.htm
* http://docs.oracle.com/database/121/LNPLS/datatypes.htm#LNPLS003

# INDEX

base types 2, 3, 4, 21

BOOLEAN 3, 11, 21

Composite data types 2

Cursors 3

PLS\_INTEGER 3, 4, 8, 9, 16, 17, 18, 21

Scalar data types 2, 21

SQL data types 3, 21

subtypes 2, 3, 4

Variable 1, 2, 4, 6, 8, 14, 21