Table of Contents

[CHAPTER 4: NESTED BLOCK AND CODE EXECUTION 2](#_Toc411546187)

[Theory 2](#_Toc411546188)

[AIM 6](#_Toc411546189)

[Lab Exercise 4: NESTED BLOCK AND CODE EXECUTION 7](#_Toc411546190)

[1. Code Execution 8](#_Toc411546191)

[2. Nested Block 12](#_Toc411546192)

[3. Variable Scope 18](#_Toc411546193)

[SUMMARY 26](#_Toc411546194)

[REFERENCES 27](#_Toc411546195)

[INDEX 28](#_Toc411546196)

# CHAPTER 4: NESTED BLOCK AND CODE EXECUTION

## Theory

PL/SQL block is mainly seen as execution unit. Normally, the execution runs from the top to the bottom of the block. However, using some control statement, programmer may change the normal execution, such as using EXIT and CONTINUE statements. Normal execution may also change at run-time due to exceptions and run-time errors.

DECLARE

.....

..

BEGIN

....

...<<error>>

..

EXCEPTION

....

END;

**Exceptional Execution**

**Normal Execution**

DECLARE

.....

..

BEGIN

....

...

..

EXCEPTION

....

END;

Other way to alter the normal execution is to use nested block. A nested block is defined as a PL/SQL block which is embedded inside other PL/SQL block. Nested block may not only appear in Executable part of the outer block, it may appear inside the Exception-Handling part as well. But it can't be appear inside the Declaration part. Nested block may contain sub-nested blocks and so on.

DECLARE **INNER BLOCK**

..

BEGIN

....

EXCEPTION

....

END;

DECLARE **OUTER BLOCK**

..

BEGIN

....

EXCEPTION

....

END;

DECLARE  **INNER BLOCK**

..

BEGIN

....

EXCEPTION

....

END;

DECLARE **OUTER BLOCK**

..

BEGIN

....

EXCEPTION

....

END;

There are three reasons to use nested blocks:

1. Readability: Using PL/SQL blocks, human eye can easily recognize related codes.
2. Save memory for allocating variables: Nested block can have its own variables. In this case, these variables will not be allocated in memory until the execution reach the nested block. Moreover, after executing the nested block, these variables will be de-allocated from the memory.
3. Control code execution: Suppose you have a loop that fetches rows from a table. For each row, you run some calculations which failed for one row in the middle of the table. In this case, the block execution will stop after failed row and all remaining rows will not be reached. But you don't want it like that. You need it to run the calculations, record any errors -if any- and proceed to the end of the table. A nested block will best serve in this case just by adding it with exception-handling part inside a loop. See the following figure.

DECLARE

.....

..

BEGIN

LOOP

...<<error>>

END LOOP;

....

EXCEPTION

....

END;

DECLARE

.....

BEGIN

LOOP

END LOOP;

...

EXCEPTION

....

END;

BEGIN

...<<error>>

EXCEPTION

....

END;

The **scope** of a variable is the region of a PL/SQL block from which you can reference the variable. The **visibility** of a variable is the region of a PL/SQL block from which you can reference the variable without qualifying it. A variable is **local** to the PL/SQL block that declares it. If that block has nested block(s), the variable is global to them.

 Visibility   
 Scope

DECLARE

i INTEGER;

BEGIN

....

......

EXCEPTION

....

END;

DECLARE

i INTEGER;

BEGIN

...

EXCEPTION

....

END;

If a nested block re-declares a global variable, then inside the nested block, both variables are **in scope**, but only the local variable is **visible** (can be referenced without qualifying). Please note: the scope of the variable is always bigger than or equal its visibility. In other words, the visibility area is subset of the scope area. To reference the global variable, the nested block **must qualify it** with the name of the block that declared it. If that block has no name, then the nested block(s) cannot reference the global variable.

A PL/SQL block **cannot** reference variables declared in other blocks at the same level, because those variables are neither local nor global to the block.

## AIM

The AIM of the following exercise is to demonstrate how to work with nested PL/SQL blocks.

The steps involved will include:

* Code Execution
* Nested Blocks
* Variable Scope

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:

30 minutes

# Lab Exercise 4: NESTED BLOCK AND CODE EXECUTION

|  |
| --- |
|  |

## Code Execution

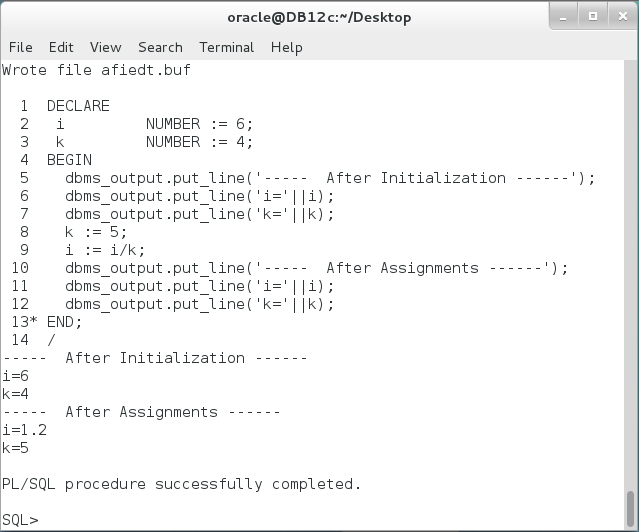
**Step 1:** Open the Terminal, open SQL\*Plus console and connect to hr schema.

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

****

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

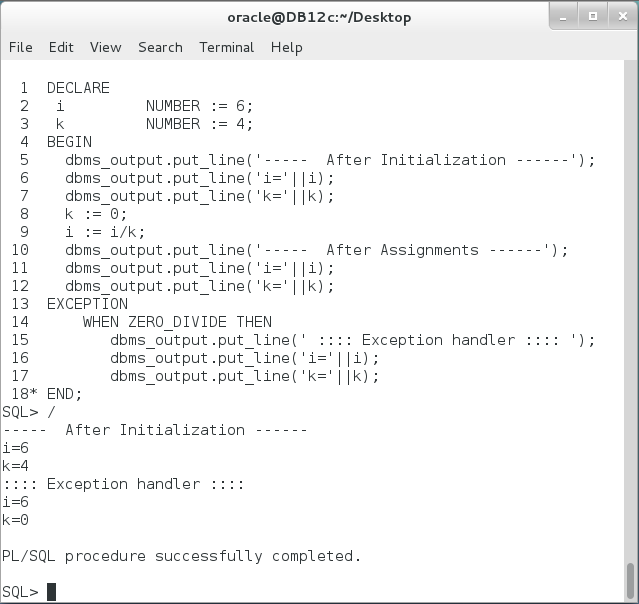
|  |  |
| --- | --- |
| Command | Description |
| DECLARE |  |
| i NUMBER := 6; | Define i variable |
| k NUMBER := 4; | Define k variable |
| BEGIN |  |
| dbms\_output.put\_line('----- After Initialization ------'); | Print variables value after initialization |
| dbms\_output.put\_line('i='||i); |
| dbms\_output.put\_line('k='||k); |
| k := 5; | Assign new values to variables |
| i := i/k; |
| dbms\_output.put\_line('----- After Assignments ------'); | Print variable values after assignment |
| dbms\_output.put\_line('i='||i); |
| dbms\_output.put\_line('k='||k); |
| END; |  |
| / |  |

****

**Please note**: In the previous PL/SQL block, the code was executed sequentially from top to bottom; from initialization to end.

**Step 3:** Modify the previous PL/SQL block as show below:

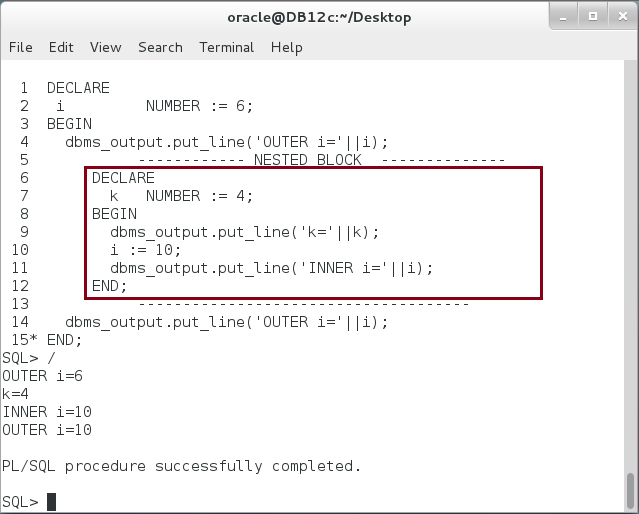
|  |  |
| --- | --- |
| Command | Description |
| DECLARE |  |
| i NUMBER := 6; |  |
| k NUMBER := 4; |  |
| BEGIN |  |
| dbms\_output.put\_line('----- After Initialization ------'); | Print variables value after initialization |
| dbms\_output.put\_line('i='||i); |
| dbms\_output.put\_line('k='||k); |
| **k := 0;** | Set k value to **ZERO** |
| i := i/k; |
| dbms\_output.put\_line('----- After Assignments ------'); | This code was not executed. **WHY?** |
| dbms\_output.put\_line('i='||i); |
| dbms\_output.put\_line('k='||k); |
| EXCEPTION |  |
| WHEN ZERO\_DIVIDE THEN |  |
| dbms\_output.put\_line(' :::: Exception handler :::: '); | The execution jump to this code. **Why**?  **Explain variable values.** |
| dbms\_output.put\_line('i='||i); |
| dbms\_output.put\_line('k='||k); |
| END; |  |
| / |  |



## Nested Block

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

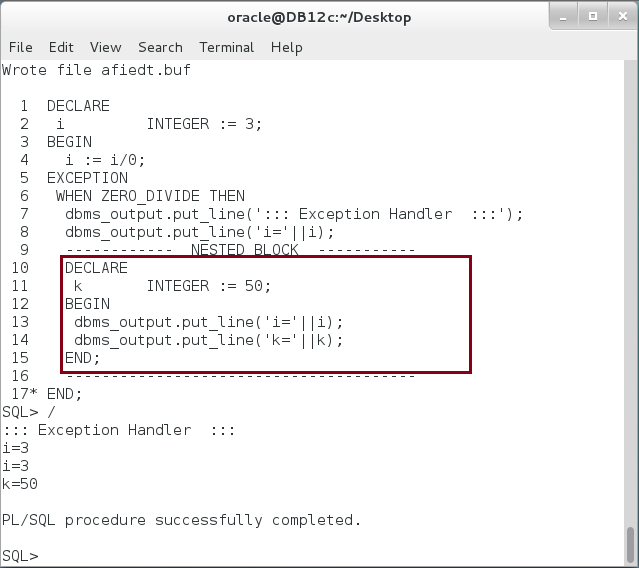
|  |  |
| --- | --- |
| Line | Description |
| DECLARE |  |
| i NUMBER := 6; | **Global** variable i |
| BEGIN |  |
| dbms\_output.put\_line('OUTER i='||i); |  |
| ------------ NESTED BLOCK -------------- | Nested Block in **block body.** |
| DECLARE |
| k NUMBER := 4; | **Local** variable k |
| BEGIN |  |
| dbms\_output.put\_line('k='||k); |  |
| i := 10; | Set global variable value |
| dbms\_output.put\_line('INNER i='||i); |  |
| END; |  |
| ------------------------------------- |  |
| dbms\_output.put\_line('OUTER i='||i); | Global variable has been set |
| END; |  |
| / |  |



**Please note**: Nested block refers and sets the global variable which was defined in the outer block.

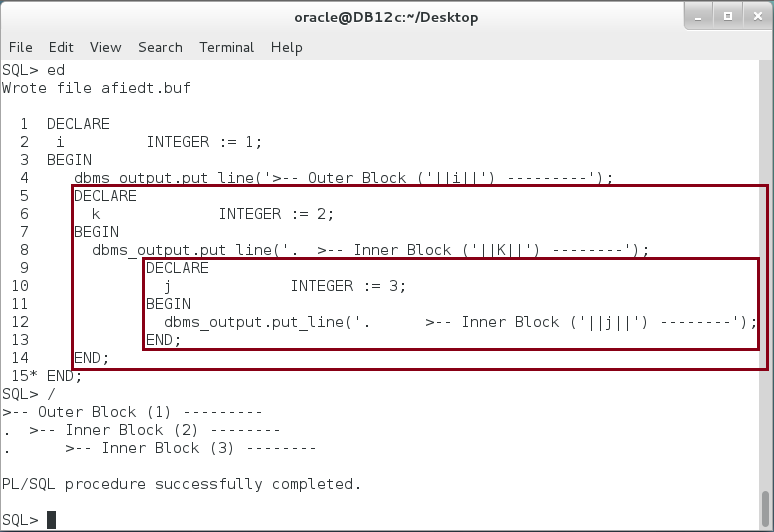
**Step 2:** Nested block can also appear in Exception-Handling part.

|  |  |
| --- | --- |
| Command | Description |
| DECLARE |  |
| i INTEGER := 3; | **Global** variable i |
| BEGIN |  |
| i := i/0; |
| EXCEPTION |
| WHEN ZERO\_DIVIDE THEN |
| dbms\_output.put\_line('::: Exception Handler :::'); |
| dbms\_output.put\_line('i='||i); |
| ------------ NESTED BLOCK ----------- | Nested Block in **Exception-handling part**. |
| DECLARE |
| k INTEGER := 50; | **Local** variable k |
| BEGIN | Print local and global variables. |
| dbms\_output.put\_line('i='||i); |
| dbms\_output.put\_line('k='||k); |
| END; |
| --------------------------------------- |
| END; |
| / |



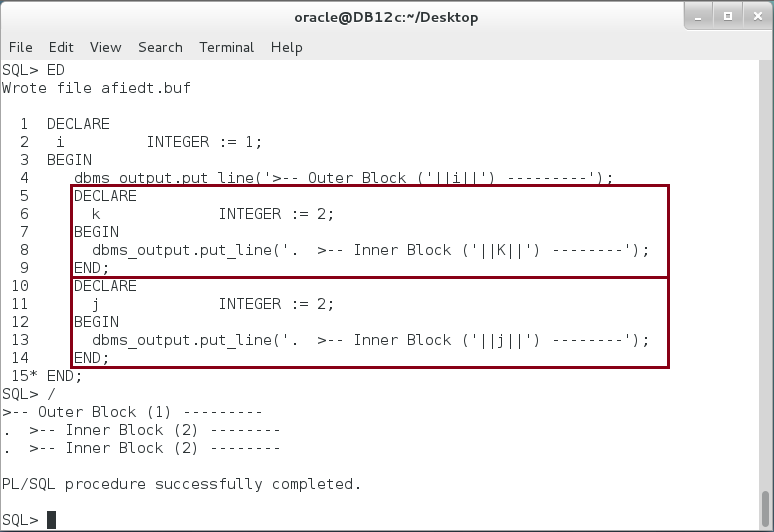
**Step 3:** Nested block can have a nested block. Execute the following block:

|  |  |
| --- | --- |
| Command | Description |
| DECLARE | Outer Block |
| i INTEGER := 1; |  |
| BEGIN |  |
| dbms\_output.put\_line('>-- Outer Block ('||i||') '); |  |
| DECLARE | Inner Block (2) |
| k INTEGER := 2; |  |
| BEGIN |  |
| dbms\_output.put\_line('. >-- Inner Block ('||K||') '); |  |
| DECLARE | Inner-Inner Block. |
| j INTEGER := 3; |
| BEGIN |
| dbms\_output.put\_line('. >-- Inner Block ('||j||')'); |
| END; |
| END; |  |
| END; |  |
| / |  |

****

**Step 4:** PL/SQL block can have many nested blocks in the same level. Move the inner block (3) to be the same level as inner block (2) as show be:

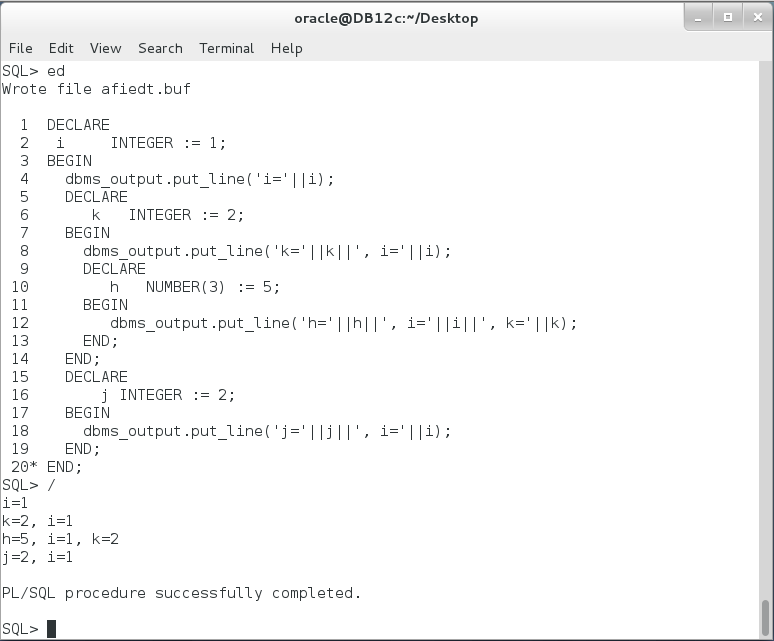
|  |  |
| --- | --- |
| Command | Description |
| DECLARE |  |
| i INTEGER := 1; |  |
| BEGIN |  |
| dbms\_output.put\_line('>-- Outer Block ('||i||') '); |  |
| DECLARE | Inner Block |
| k INTEGER := 2; |
| BEGIN |
| dbms\_output.put\_line('. >-- Inner Block ('||K||') '); |
| END; |
| DECLARE | Inner Block |
| j INTEGER := 2; |
| BEGIN |
| dbms\_output.put\_line('. >-- Inner Block ('||j||') '); |
| END; |
| END; |  |
| / |  |

****

## Variable Scope

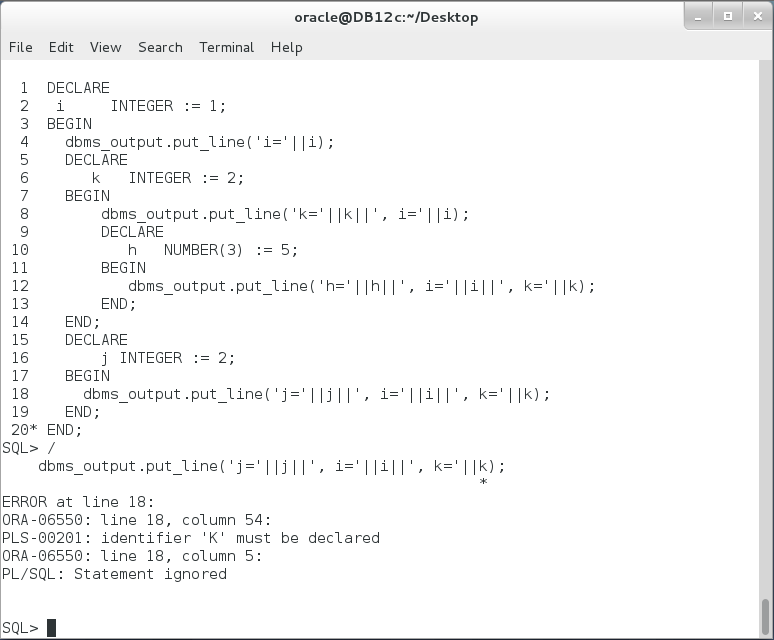
**Step 1:** You already tested the Global and local variables in previous block. In this section, you will extend your knowledge to identified the scope of each variable. In the following block, identify where is the global and local variables:

|  |  |
| --- | --- |
| Command | Description |
| DECLARE |  |
| i INTEGER := 1; |  |
| BEGIN |  |
| dbms\_output.put\_line('i='||i); |  |
| DECLARE |  |
| k INTEGER := 2; |  |
| BEGIN |  |
| dbms\_output.put\_line('k='||k||', i='||i); |  |
| DECLARE |  |
| h NUMBER(3) := 5; |  |
| BEGIN |  |
| dbms\_output.put\_line('h='||h||', i='||i||', k='||k); |  |
| END; |  |
| END; |  |
| DECLARE |  |
| j INTEGER := 2; |  |
| BEGIN |  |
| dbms\_output.put\_line('j='||j||', i='||i); |  |
| END; |  |
| END; |  |
| / |  |

****

**Step 2:** Try to access k variable in a block of the same level, Explain the output:

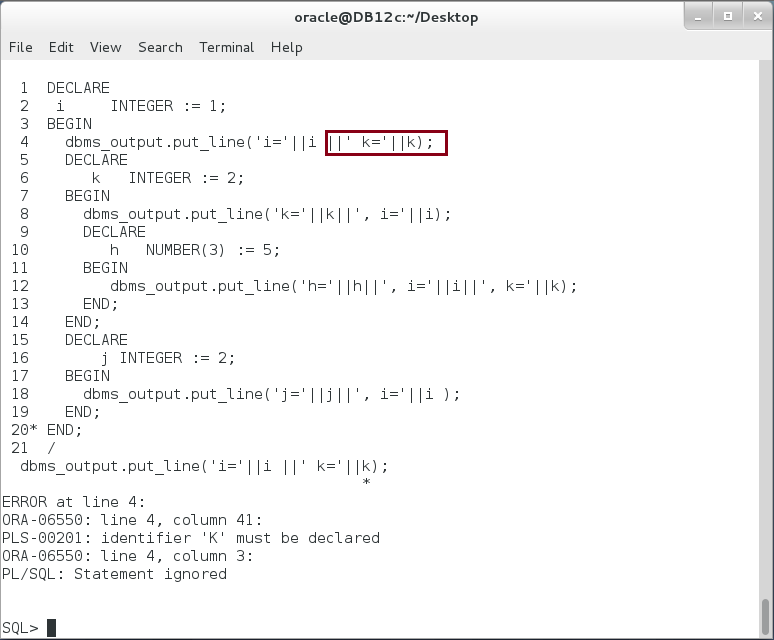
|  |  |
| --- | --- |
| Command | Description |
| DECLARE |  |
| i INTEGER := 1; |  |
| BEGIN |  |
| dbms\_output.put\_line('i='||i); |  |
| DECLARE |  |
| k INTEGER := 2; |  |
| BEGIN |  |
| dbms\_output.put\_line('k='||k||', i='||i); |  |
| DECLARE |  |
| h NUMBER(3) := 5; |  |
| BEGIN |  |
| dbms\_output.put\_line('h='||h||', i='||i||', k='||k); |  |
| END; |  |
| END; |  |
| DECLARE |  |
| j INTEGER := 2; |  |
| BEGIN |  |
| dbms\_output.put\_line('j='||j||', i='||i **|| ', k='||k**); | Access k variable which has already been declared in previous block. **Explain**? |
| END; |  |
| END; |  |
| / |  |

****

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

**Step 3:** In previous block try to access k variable, which has been declared in an inner block, in the most outer block as show below:

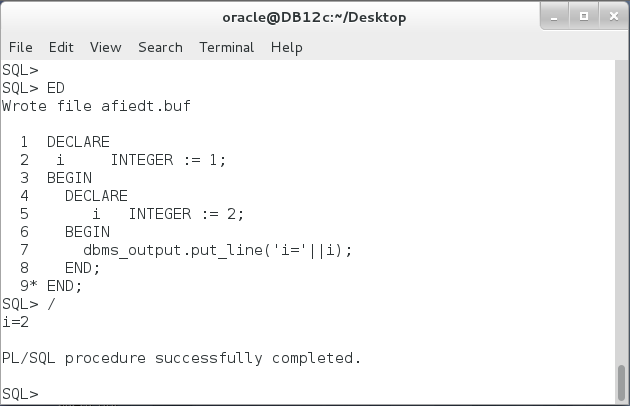
|  |  |
| --- | --- |
| Command | Description |
| DECLARE |  |
| i INTEGER := 1; |  |
| BEGIN |  |
| dbms\_output.put\_line('i='||i ||**' k='||k**); | Try to access a variable k which was declare in a child block. **Explain the output?** |
| DECLARE |  |
| k INTEGER := 2; |  |
| BEGIN |  |
| dbms\_output.put\_line('k='||k||', i='||i); |  |
| DECLARE |  |
| h NUMBER(3) := 5; |  |
| BEGIN |  |
| dbms\_output.put\_line('h='||h||', i='||i||', k='||k); |  |
| END; |  |
| END; |  |
| DECLARE |  |
| j INTEGER := 2; |  |
| BEGIN |  |
| dbms\_output.put\_line('j='||j||', i='||i ); |  |
| END; |  |
| END; |  |
| / |  |



PL/SQL block failed again, **Why**?

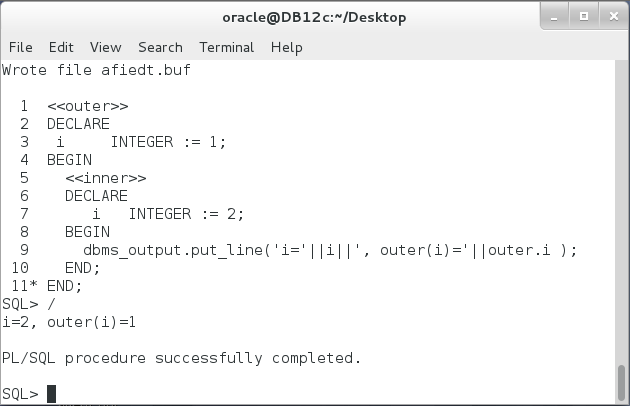
**Step 4:** In previous PL/SQL blocks, you see that you can't use a variable defined in an inner block in outer block. You also can't use variable defined in a block in the same level block. In the other hand, you may use a variable defined in the outer block in all of its inner blocks. In the next PL/SQL block, you will examine what would happen when you define a variable in the inner block and its name as the same variable name defined in the outer block.

|  |  |
| --- | --- |
| Command | Description |
| DECLARE |  |
| i INTEGER := 1; | Variable i definition. |
| BEGIN |  |
| DECLARE |  |
| i INTEGER := 2; | Variable i was defined in the outer block. |
| BEGIN |  |
| dbms\_output.put\_line('i='||i); | **Print local variable i value = 2** |
| END; |  |
| END; |  |
| / |  |

****

**Step 5:** In previous PL/SQL, a local variable has a higher precedence than the global variable. Remember: a global variable scope extends over all inner blocks. Thus, a global variable can be accessed in the inner block(s) but its visibility was limited because of the local variable of the same name. To access the outer variable (i) in the inner block, qualify it with a block name as shown below:

|  |  |
| --- | --- |
| Command | Description |
| <<outer>> | Outer block label |
| DECLARE |  |
| i INTEGER := 1; |  |
| BEGIN |  |
| <<inner>> |  |
| DECLARE |  |
| i INTEGER := 2; |  |
| BEGIN |  |
| dbms\_output.put\_line('i='||i||', outer(i)='||outer.i); | **Local and Global** |
| END; |  |
| END; |  |
| / |  |



# SUMMARY

Normally, code execution runs from top to bottom in PL/SQL block. Programmers can change the execution default flow using control statements or using nested block. A nested block is a PL/SQL block defined inside other PL/SQL block called outer block. Nested block may be also used to save memory. Any variable define inside a PL/SQL block is called a local variable for that block and global variable for all inner block(s). Variable scope starts from its definition in the Declaration part to the END keyword of the block. The scope means a code region in which the variable can be accessed. A visibility is a sub-region of scope region in which the variable can be accessed without full name qualifier. Inner variable can't be referenced in the outer block nor in any block of the same level.

After completing this lab exercise, you should be able to use PL/SQL nested block and clearly understand the scope and visibility of a variable.

# REFERENCES

* http://docs.oracle.com/database/121/LNPLS/fundamentals.htm#LNPLS99945
* http://www.oracle.com/technetwork/issue-archive/2011/11-mar/o21plsql-242570.html
* http://www.zentut.com/plsql-tutorial/plsql-nested-block/

# INDEX

nested block 2, 3, 4, 5, 13, 22

**scope** 4, 5, 15, 20, 22

**visibility** 4, 5, 20, 22