Table of Contents

[CHAPTER 2: PL/SQL BLOCK 2](#_Toc416695179)

[Theory 2](#_Toc416695180)

[AIM 3](#_Toc416695181)

[Lab Exercise 2: PL/SQL BLOCK 4](#_Toc416695182)

[1. Using SQL\*Plus 5](#_Toc416695183)

[2. Creating Anonymous PL/SQL Block 11](#_Toc416695184)

[3. Labels and Comments 25](#_Toc416695185)

[SUMMARY 31](#_Toc416695186)

[REFERENCES 32](#_Toc416695187)

[INDEX 33](#_Toc416695188)

# CHAPTER 2: PL/SQL BLOCK

## Theory

In previous chapter, you have learnt that PL/SQL Engine runs PL/SQL blocks inside Oracle Database. Oracle Database was installed. You can build, run, and debug PL/SQL block using SQL\*Plus or any other PL/SQL development tools such as SQL Developer. This course assumes that the reader is well familiar with SQL. Readers who don't know SQL basics are highly recommended to review or to take SQL course before proceeding in this course.

The basic unit of a PL/SQL source program is the block, which groups related declarations and statements. The block may consist of three parts:

1. Declarative part
2. Executable part
3. Exception-handling part

Declaration Part:   
Declarations of local types, variables, & subprograms

Executable Part:  
Control statements, Loop statements, Variables assignments and arithmetic, SQL Statements

Exception-Handling Part:  
Exception handlers for exceptions (errors) raised in executable part

DECLARE

BEGIN

EXCEPTION

END;

Four reserved words are used to create these three parts: DECLARE, BEGIN, EXCEPTION, and END;

Executable Part is the only mandatory part, other parts (Declaration, Exception-Handling parts) are optional. Declaration Part starts with DECLARE and ends with BEGIN keywords. Exception-Handling Part starts with EXCEPTION and ends with END; keywords. Executable part starts with BEGIN and ends with either EXCEPTION or ENDS; keywords. Remember that Exception-Handling part is optional. Thus, if it is exists, then the Executable Part ends with EXCEPTION keyword. Otherwise, it ends with END;

## AIM

The AIM of the following exercise is to demonstrate the basic of Oracle PL/SQL Anonymous block

The steps involved will include:

* Using SQL\*Plus
* Creating Anonymous PL/SQL Block
* Labels and Comments

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 2: PL/SQL BLOCK

|  |
| --- |
|  |

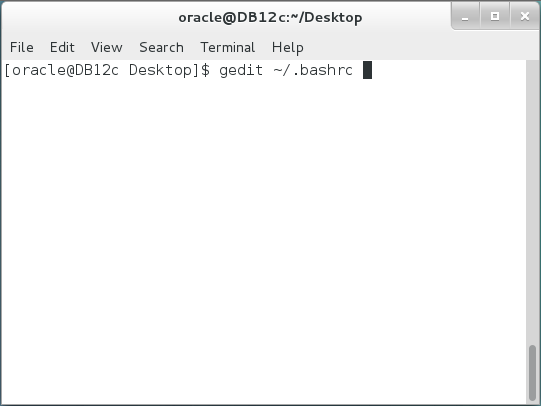
## Using SQL\*Plus

**Step 1:** SQL\*Plus is the easiest way to execute PL/SQL block. Before using SQL\*Plus you should be sure that:

1. Oracle Database 12c is installed and configured as shown in Chapter 1.
2. Oracle Database service and Listener are up and running. Ask your trainer to help you in this issue.

**Step 2:** Set your environment parameter before using SQL\*Plus to avoid unnecessary repetition throughout the course. Login as **'oracle'**, open the Terminal and execute the following:.

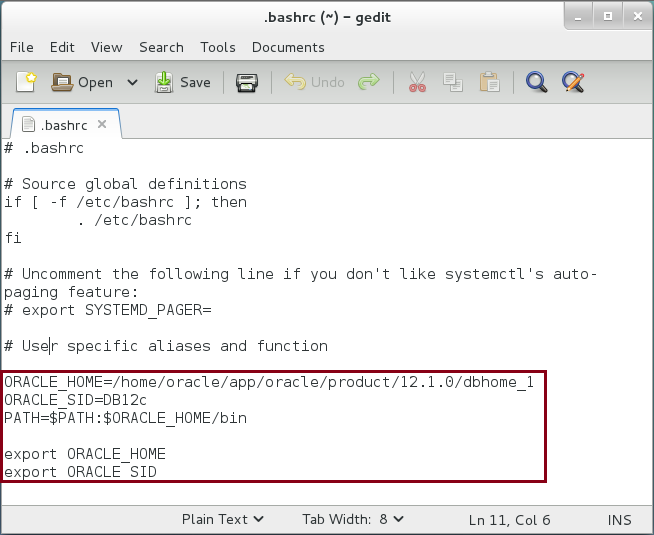
|  |  |
| --- | --- |
| Command | Description |
| gedit ~/.bashrc | Edit a user bash file to add ORACLE\_HOME and ORACLE\_SID parameters |

****

**Step 3:** The editor opens **bashrc** file. Edit the file by adding the following lines:

|  |  |
| --- | --- |
| Line | Description |
| ORACLE\_HOME=/home/oracle/app/oracle/product/12.1.0/dbhome\_1 | Define ORACLE\_HOME parameter |
| ORACLE\_SID=DB12c | Define ORACLE\_SID parameter |
| PATH=$PATH:$ORACLE\_HOME/bin | Add ORACLE\_HOME/bin to the default path. |
| export ORACLE\_HOME | Set ORACLE\_HOME parameter |
| export ORACLE\_SID | Set ORACLE\_SID parameter |

Save changes and exit.

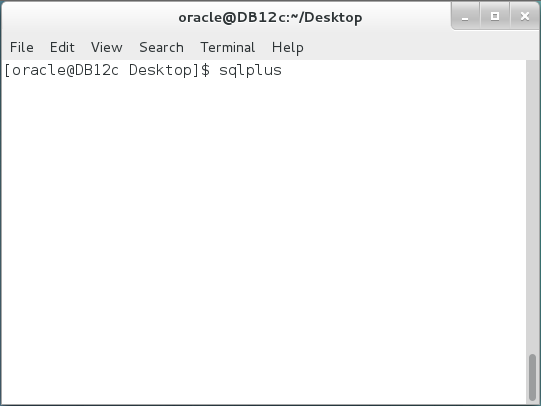


**Step 4:** Re-open the Terminal again and execute the following command to run SQL\*Plus:

|  |  |
| --- | --- |
| Command | Description |
| sqplus | Run SQL\*Plus |

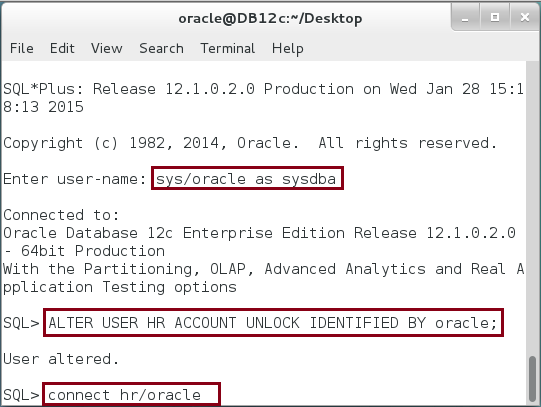
If you face any problem running SQL\*Plus, please be sure that ORACLE\_HOME, ORACLE\_SID and PATH variable are set correctly in your shell. For example, you may view PATH environment variable by executing the following command:

|  |  |
| --- | --- |
| Command | Description |
| echo $PATH | Show PATH variable. |



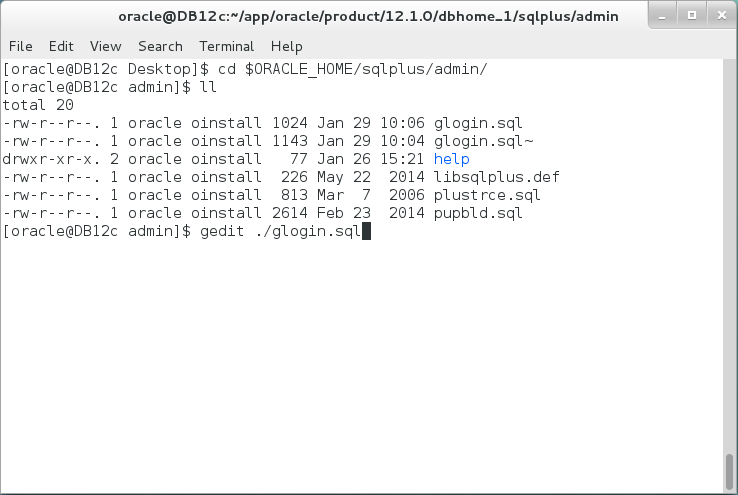
**Step 5:** We are going to use **'hr'** account throughout this course. To use this account, execute the following commands:

|  |  |
| --- | --- |
| Command | Description |
| sys/oracle as sysdba | login as sys |
| ALTER USER HR ACCOUNT UNLOCK IDENTIFIED BY oracle; | Unlock **'hr'** user |
| connect hr/oracle | connect to **'hr'** |



**Step 6:** SQL\*Plus environment settings help you to view query or PL/SQL result better on SQL\*Plus console or spool file. To avoid doing these settings each time you open SQL\*Plus, it is better to set them permanently. Open the Terminal and execute the following commands:

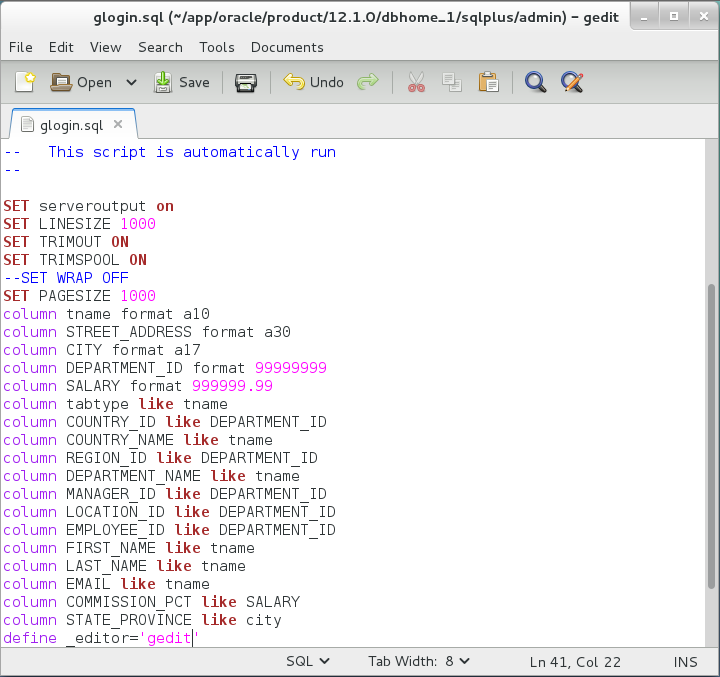
|  |  |
| --- | --- |
| Command | Description |
| cd $ORACLE\_HOME/sqlplus/admin/ | Change directory |
| ll | list the content of the directory |
| gedit ./glogin.sql | Open **'glogin.sql**' file to edit SQL\*Plus settings. |



**Step 7:** Add the following settings to glogin.sql file:

|  |
| --- |
| **Settings** |
| SET serveroutput on  SET LINESIZE 1000  SET TRIMOUT ON  SET TRIMSPOOL ON  SET PAGESIZE 1000  column tname format a10  column STREET\_ADDRESS format a30  column CITY format a17  column DEPARTMENT\_ID format 99999999  column SALARY format 999999.99  column tabtype like tname  column COUNTRY\_ID like DEPARTMENT\_ID  column COUNTRY\_NAME like tname  column REGION\_ID like DEPARTMENT\_ID  column DEPARTMENT\_NAME like tname  column MANAGER\_ID like DEPARTMENT\_ID  column LOCATION\_ID like DEPARTMENT\_ID  column EMPLOYEE\_ID like DEPARTMENT\_ID  column FIRST\_NAME like tname  column LAST\_NAME like tname  column EMAIL like tname  column COMMISSION\_PCT like SALARY  column STATE\_PROVINCE like city  define \_editor='gedit' |

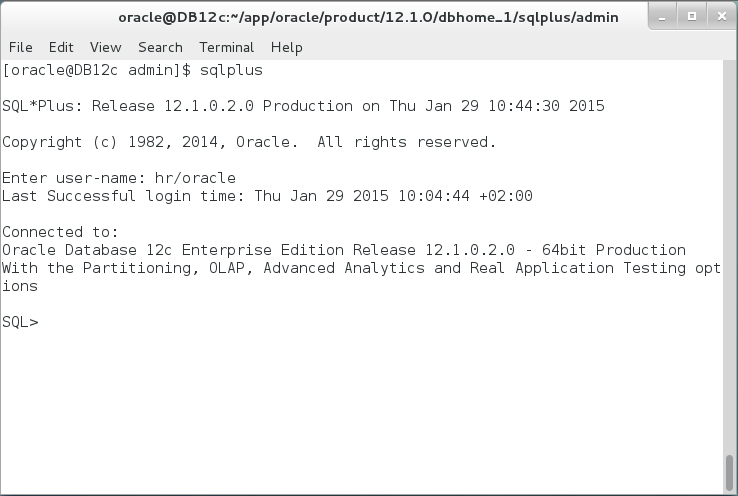
Save the file and exit.



## Creating Anonymous PL/SQL Block

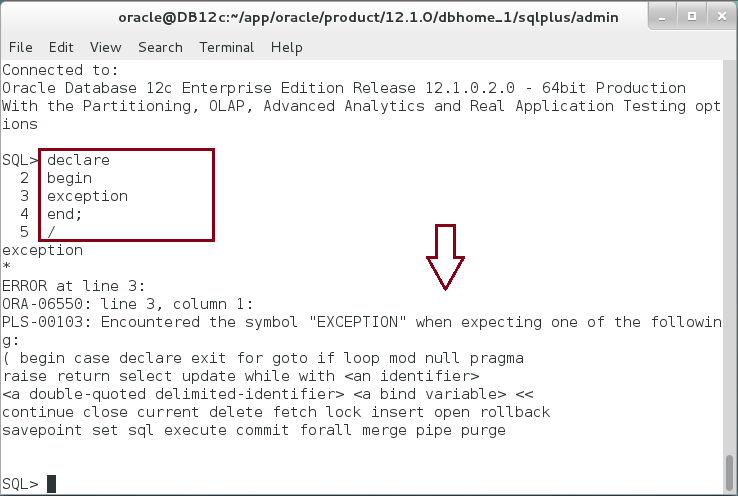
**Step 1:** In previous section, you prepared your SQL\*Plus settings. In this section, you will write your first PL/SQL block. Open the Terminal and connect to SQL\*Plus using 'hr' account:

|  |  |
| --- | --- |
| Command | Description |
| sqlplus | Open SQL\*Plus console |
| hr/oracle | Connect as **'hr'** user. |



**Step 2:** Write the following lines:

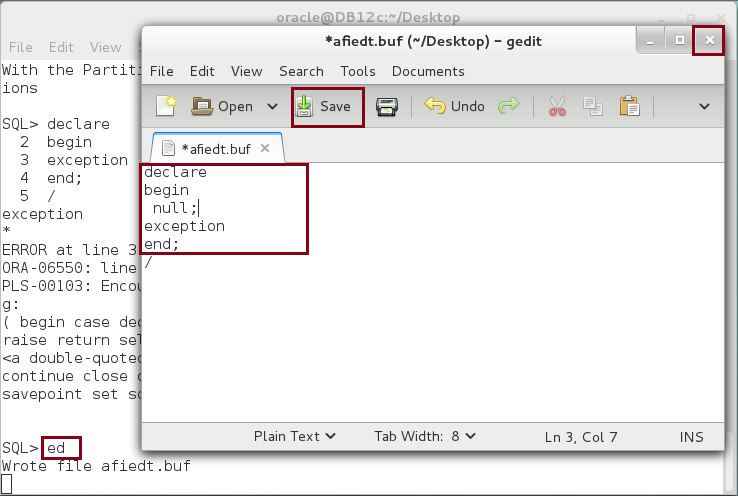
|  |  |
| --- | --- |
| Command | Description |
| declare | Start Declaration Part |
| begin | Start Executable Part |
| Exception | Start Exception-Handling Part |
| End; | Ends PL/SQL block |
| / | Run PL/SQL block. |

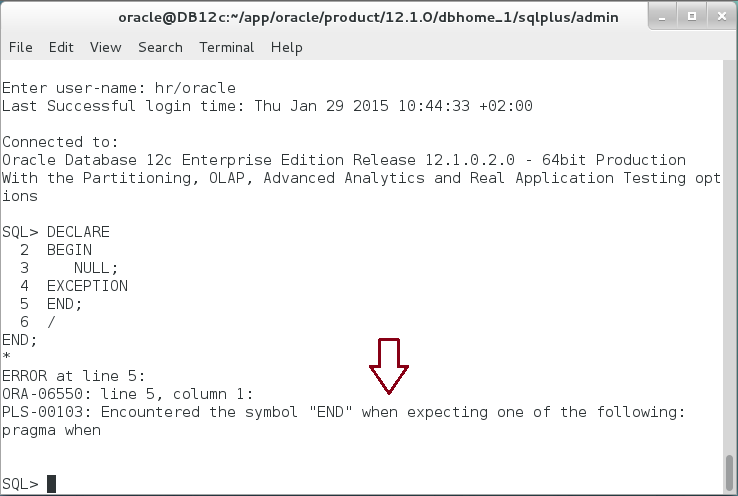
This is the first attempt to write PL/SQL block. Unfortunately, the attempt failed with error. The error says that "EXCEPTION" keyword should not appear on this place. This because Executable part can't be empty. 

**Step 3:** Add 'null;' command and execute the Block:

|  |  |
| --- | --- |
| Command | Description |
| declare | Start Declaration Part |
| begin | Start Executable Part |
| null; | 'null;' is an executable statement that do nothing. Every executable statement ends with semicolon. |
| Exception | Start Exception-Handling Part |
| End; | Ends PL/SQL block |
| / | Run PL/SQL block. |

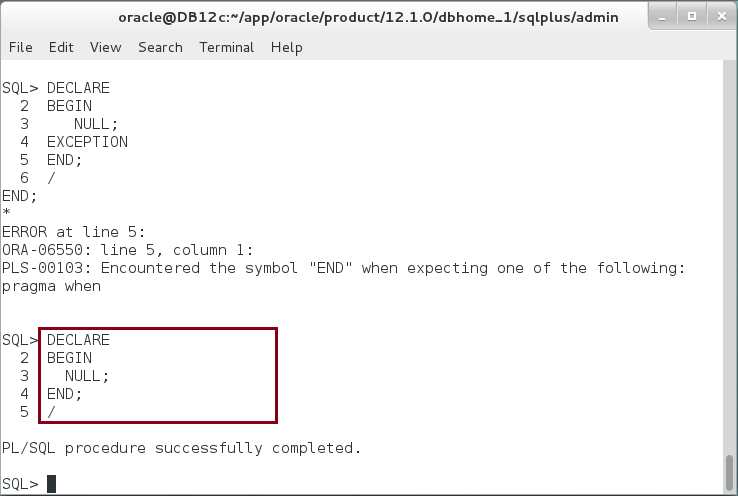
Again, the block failed to execute with error. This time, 'END' keyword should not appear here. Thus, Exception-handling part should not be empty. However, remember that Exception-Handling part is optional part.

**Important Note**: Each time you are asked to update the previous PL/SQL block, you may use **'ed'** command to open the previous executed PL/SQL in the editor instead of re-writing the whole text again. Save the file, close it and don't forget to execute the new PL/SQL block by '**/**' and **ENTER**. 



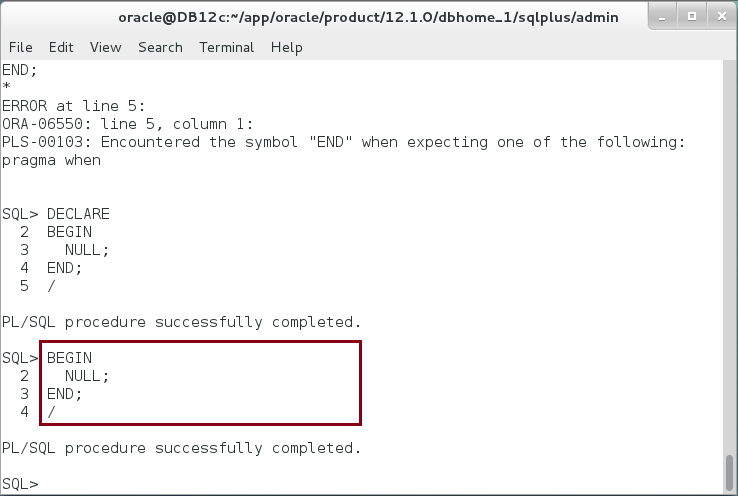
**Step 4:** Remove 'EXCEPTION' keyword:

|  |  |
| --- | --- |
| Command | Description |
| declare | Start Declaration Part |
| begin | Start Executable Part |
| null; | 'null;' is an executable statement that do nothing. Every executable statement ends with semicolon. |
| End; | Ends PL/SQL block |
| / | Run PL/SQL block. |

The block executed successfully. 

**Step 5:** Declaration part is also optional, remove **DECLARE** keyword.

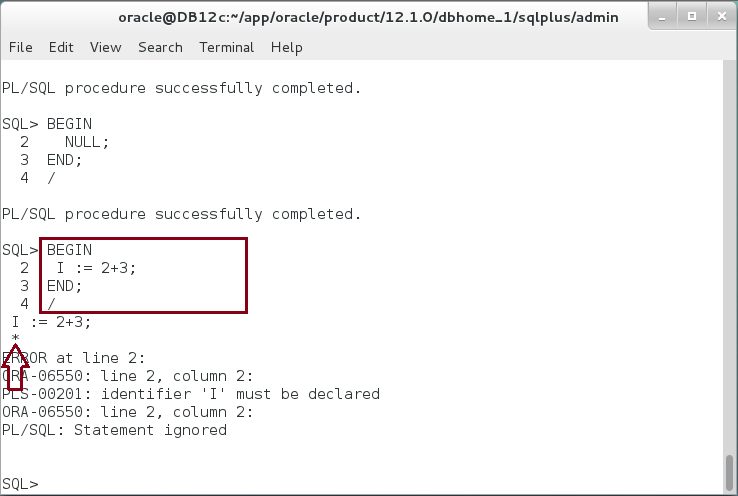
|  |  |
| --- | --- |
| Command | Description |
| begin | Start Executable Part |
| null; | 'null;' is an executable statement that do nothing. Every executable statement ends with semicolon (;) |
| End; | Ends PL/SQL block |
| / | Run PL/SQL block. |



**Step 6:** Try to execute an arithmetic operation as the following.

|  |  |
| --- | --- |
| Command | Description |
| begin | Start Executable Part |
| I := 2+3; | Add 2 to 3 and **assign** '**:=**' the value to '**I**' variable. Don't forget a semicolon **(;)** at the end. |
| End; | Ends PL/SQL block |
| / | Run PL/SQL block. |

The execution failed because the variable 'I' is used but it is not defined. In this case, adding a Declaration part is mandatory.



**Step 7:** Add a Declaration part and declare the variable **'i**' as the following.

|  |  |
| --- | --- |
| Command | Description |
| DECLARE | Start Declaration Part |
| i integer; | Define a variable '**i**' as '**integer**'. |
| begin | Start Executable Part |
| i := 2+3; | Add 2 to 3 and **assign** '**:=**' the value to '**i**' variable. Don't forget a semicolon **(;)** at the end. |
| End; | Ends PL/SQL block |
| / | Run PL/SQL block. |

The execution succeeded but the value of 'i' is not shown in SQL\*Plus console. 

**Step 8:** To view the value of the variable uses Oracle pre-defined package 'dbms\_output' as show below:

|  |  |
| --- | --- |
| Command | Description |
| DECLARE | Start Declaration Part |
| i integer; | Define a variable '**i**' as '**integer**'. |
| begin | Start Executable Part |
| dbms\_output.put\_line(' i ='||i); | Print 'i' value before assignment |
| i := 2+3; |  |
| dbms\_output.put\_line('\*i ='||i); | Print 'i' value after assignment |
| End; | Ends PL/SQL block |
| / | Run PL/SQL block. |

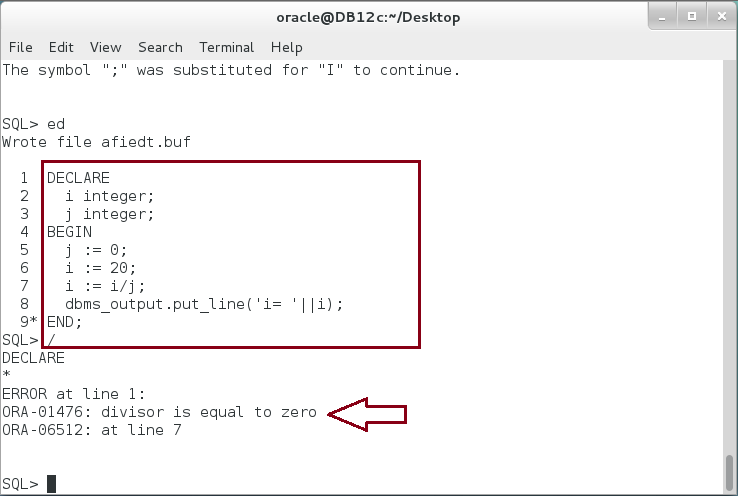


The value of 'i' before assignment is 'null' and after assignment becomes 5.

**Please note**: You will **not** see the output if you don't run ' SET serveroutput on' command in SQL\*Plus. It is highly recommended to execute [**step 7**](#_Using_SQL*Plus) in the previous section to run this command automatically on login to SQL\*Plus.

**Step 9:** Let's try other arithmetic operators. Execute the following PL/SQL block:

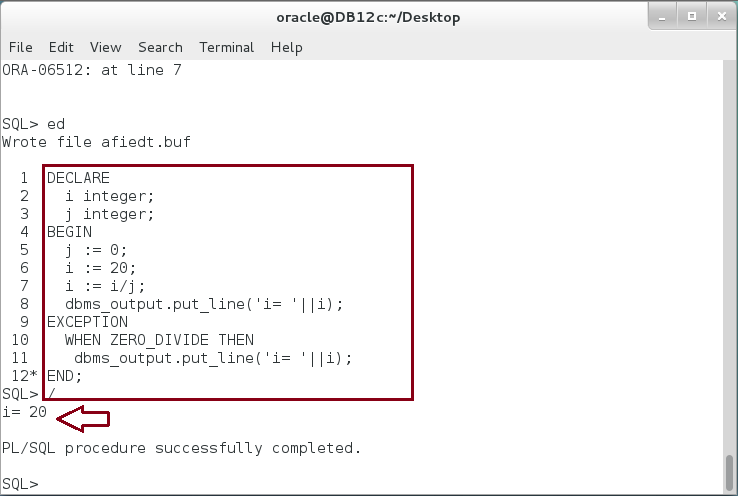
|  |  |
| --- | --- |
| Command | Description |
| DECLARE | Start Declaration Part |
| i integer; | Define a variable 'i' as 'integer'. |
| j integer; |  |
| BEGIN | Start Executable Part |
| j := 0; | Assign 0 to j |
| i := 20; | Assign 20 to i |
| i := i/j; | Assign i/j to i |
| dbms\_output.put\_line('i= '||i); | Print the value of i |
| END; | End PL/SQL block |
| / | Run PL/SQL block. |



The PL/SQL block failed to execute, even though there is no syntax error in this block and it was compiled successfully. The error raised here is a runtime error. More precisely, it is a ZERO\_DIVIDE exception.

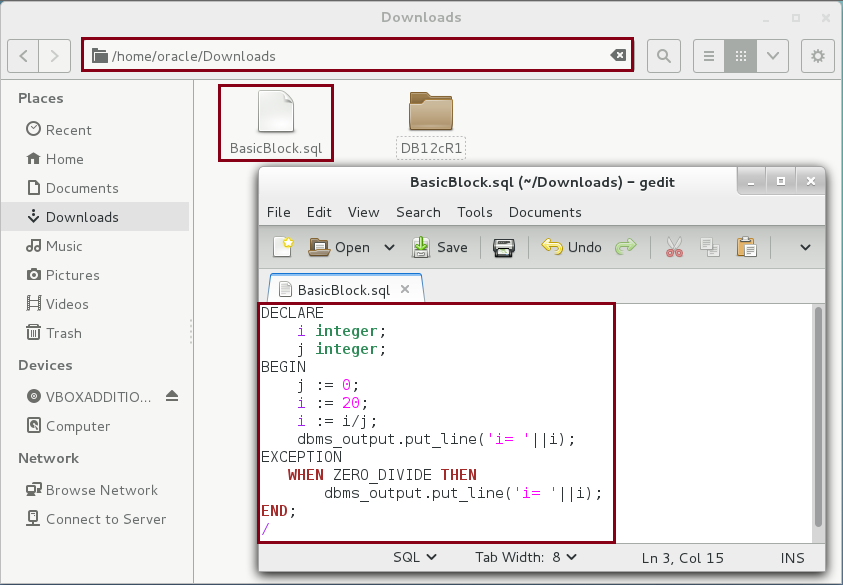
**Step 10:** To handle this exception, Exception-Handling part should be added to the block as shown:

|  |  |
| --- | --- |
| Command | Description |
| DECLARE | Start Declaration Part |
| i integer; | Define a variable 'i' as 'integer'. |
| j integer; |  |
| BEGIN | Start Executable Part |
| j := 0; | Assign 0 to j |
| i := 20; | Assign 20 to i |
| i := i/j; | Assign i/j to i |
| dbms\_output.put\_line('i= '||i); | Print the value of i |
| EXCEPTION | Start Exception-Handling Part |
| WHEN ZERO\_DIVIDE THEN | Handle ZERO\_DIVIDE application error. |
| dbms\_output.put\_line('i= '||i); | Print **'i**' value. |
| END; | End PL/SQL block |
| / | Run PL/SQL block. |



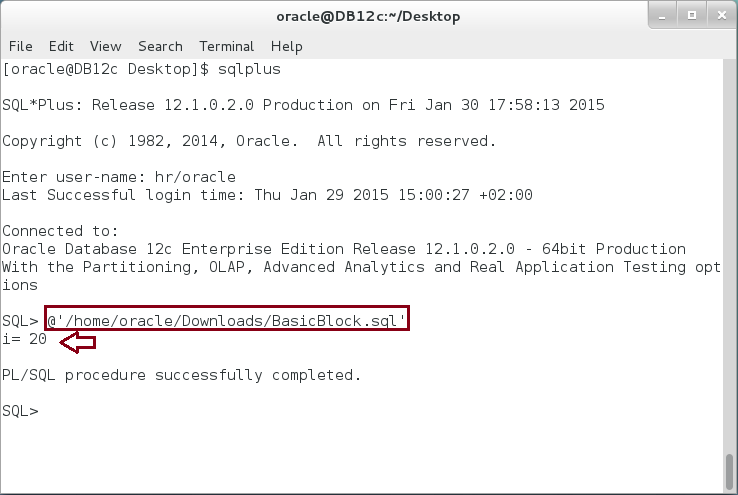
**Note**: the value of '**i**' is **20**. This is the value of '**i**' exactly before executing **'i=i/j;**' statement.

**Step 11:** Save the previous PL/SQL block as text file in **"/home/oracle/Downloads**" directory. Name the file as "**BasicBlock.sql**"



**Step 12:** You can re-run the saved block from SQL\*Plus console as the following:

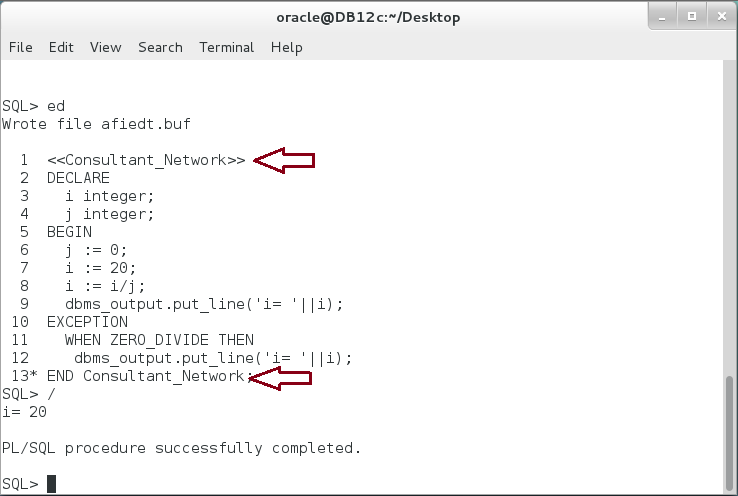
|  |  |
| --- | --- |
| Command | Description |
| @'/home/oracle/Downloads/BasicBlock.sql' | Run SQL script |

****

## Labels and Comments

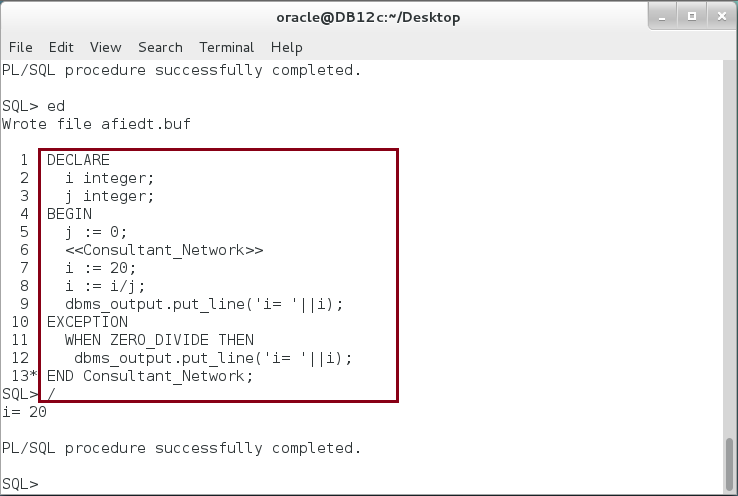
**Step 1:** Sometimes, it is very useful to add label for a block specially when there are many nested blocks in your code. Thus, it will be much easier when re-reading the code again. In the previous code, add a label for the block as the following:

|  |  |
| --- | --- |
| Command | Description |
| **<<Consultant\_Network>>** | Label the next block as "**Consultant\_Network**". |
| DECLARE | Start Declaration Part |
| i integer; | Define a variable 'i' as 'integer'. |
| j integer; | Define a variable 'i' as 'integer'. |
| BEGIN | Start Executable Part |
| j := 0; | Assign 0 to j |
| i := 20; | Assign 20 to i |
| i := i/j; | Assign i/j to i |
| dbms\_output.put\_line('i= '||i); | Print the value of i |
| EXCEPTION | Start Exception-Handling Part |
| WHEN ZERO\_DIVIDE THEN | Handle ZERO\_DIVIDE application error. |
| dbms\_output.put\_line('i= '||i); | Print **'i**' value. |
| END **Consultant\_Network**; | End **Consultant\_Network** block |
| / | Run PL/SQL block. |

****

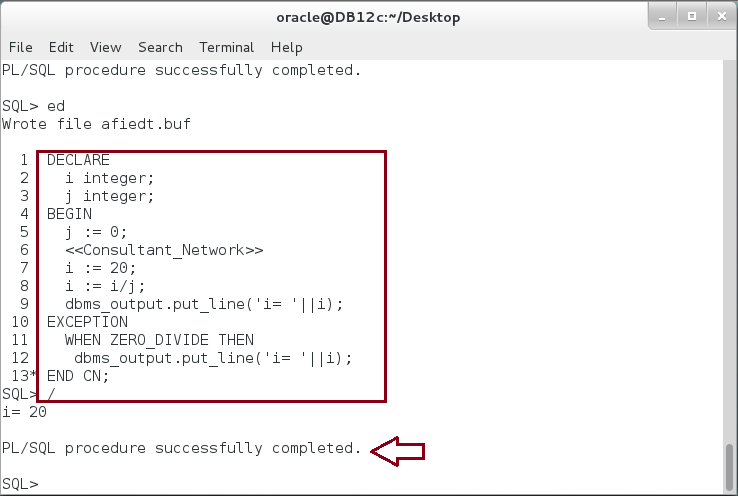
**Step 2:** Labels can also be used inside the block itself. Move the label "Consultant\_Network" inside the block as the following:

|  |  |
| --- | --- |
| Command | Description |
| DECLARE | Start Declaration Part |
| i integer; | Define a variable 'i' as 'integer'. |
| j integer; | Define a variable 'i' as 'integer'. |
| BEGIN | Start Executable Part |
| j := 0; | Assign 0 to j |
| **<<Consultant\_Network>>** | The label here does not refer to the Anonymous block. |
| i := 20; | Assign 20 to i |
| i := i/j; | Assign i/j to i |
| dbms\_output.put\_line('i= '||i); | Print the value of i |
| EXCEPTION | Start Exception-Handling Part |
| WHEN ZERO\_DIVIDE THEN | Handle ZERO\_DIVIDE application error. |
| dbms\_output.put\_line('i= '||i); | Print 'i' value. |
| END Consultant\_Network; | End "Consultant\_Network" block |
| / | Run PL/SQL block. |



**Step 3:** Be careful when using labels. Sometimes, labels may mislead the developer. For example, the label next to "END" keyword has nothing to do with any labels appear elsewhere before or in the block. Modify the block as shown below:

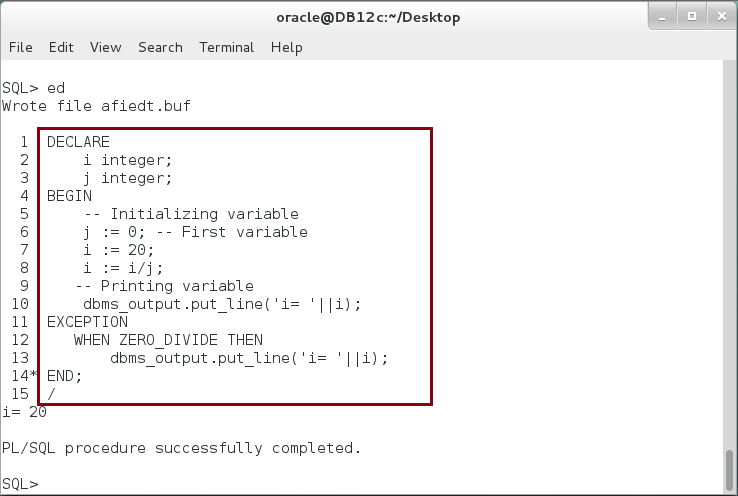
|  |  |
| --- | --- |
| Command | Description |
| DECLARE | Start Declaration Part |
| i integer; | Define a variable 'i' as 'integer'. |
| **<<Consultant\_Network>>** | Block label |
| j integer; | Define a variable 'i' as 'integer'. |
| BEGIN | Start Executable Part |
| j := 0; | Assign 0 to j |
| i := 20; | Assign 20 to i |
| i := i/j; | Assign i/j to i |
| dbms\_output.put\_line('i= '||i); | Print the value of i |
| EXCEPTION | Start Exception-Handling Part |
| WHEN ZERO\_DIVIDE THEN | Handle ZERO\_DIVIDE application error. |
| dbms\_output.put\_line('i= '||i); | Print 'i' value. |
| END **CN**; | End CN; |
| / | Run PL/SQL block. |



**Please note**: In the previous block, the label "**CN**" does not exist anywhere but next to "END" keyword. The block was compiled and ran successfully! Unfortunately, Oracle PL/SQL compiler does not raise error for this code, and the developer should carefully name labels.

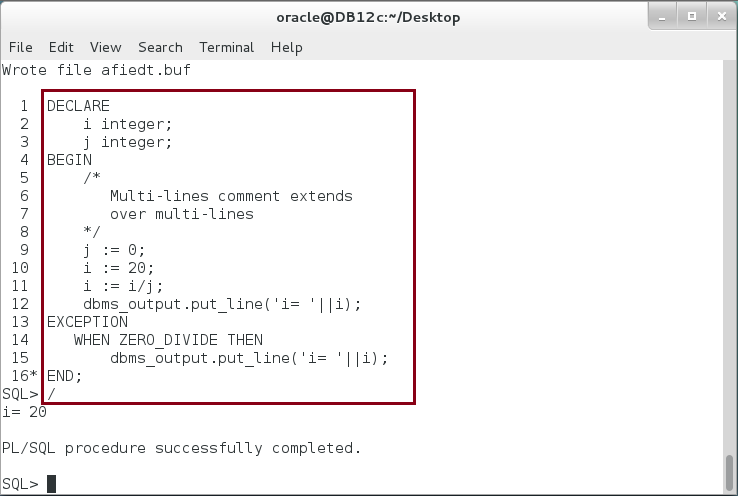
**Step 4:** It is useful to write comments on your code. Oracle PL/SQL support a single line comment by adding '--'. Any string following '--' until the end of the line will considered as comment.

|  |  |
| --- | --- |
| Command | Description |
| DECLARE | Start Declaration Part |
| i integer; | Define a variable 'i' as 'integer'. |
| j integer; | Define a variable 'i' as 'integer'. |
| BEGIN | Start Executable Part |
| **-- Initializing variable** | Single line comment |
| j := 0; **-- First variable** | Assign 0 to j |
| i := 20; | Assign 20 to i |
| i := i/j; | Assign i/j to i |
| **-- Printing variable** | Single line comment |
| dbms\_output.put\_line('i= '||i); | Print the value of i |
| EXCEPTION | Start Exception-Handling Part |
| WHEN ZERO\_DIVIDE THEN | Handle ZERO\_DIVIDE application error. |
| dbms\_output.put\_line('i= '||i); | Print 'i' value. |
| END; | End block |
| / | Run PL/SQL block. |

****

**Step 5:** Multi-line comment can be used as the following:

|  |  |
| --- | --- |
| Command | Description |
| DECLARE | Start Declaration Part |
| i integer; | Define a variable 'i' as 'integer'. |
| j integer; | Define a variable 'i' as 'integer'. |
| BEGIN | Start Executable Part |
| **/\*** | Start Multi-line comment |
| **Multi-lines comment extends** | Write your comments |
| **over multi-lines** |  |
| **\*/** | End Multi-line comment |
| j := 0; | Assign 0 to j |
| i := 20; | Assign 20 to i |
| i := i/j; | Assign i/j to i |
| dbms\_output.put\_line('i= '||i); | Print the value of i |
| EXCEPTION | Start Exception-Handling Part |
| WHEN ZERO\_DIVIDE THEN | Handle ZERO\_DIVIDE application error. |
| dbms\_output.put\_line('i= '||i); | Print 'i' value. |
| END; | End block |
| / | Run PL/SQL block. |

****

# SUMMARY

Basic PL/SQL block consists of merely executable part, which starts of "BEGIN" keyword and ends with "END" keyword. The executable part is the only mandatory part in PL/SQL block and can't be empty. Other PL/SQL parts, such as Declaration and Exception-Handling parts are optional. However, the developer should use them when he/she wants to use variables and handle a specific exception. Every executable statement in PL/SQL block should ends with (;). Oracle PL/SQL block may have labels either before, inside or next to END keyword. Labels are useful, however, developers should be careful when use them. PL/SQL supports single and multi-lines comments.

After completing this lab exercise, you should be able to write a very simple PL/SQL block.

# REFERENCES

* https://docs.oracle.com/database/121/LNPLS/fundamentals.htm
* http://www.tutorialspoint.com/plsql/plsql\_basic\_syntax.htm

# INDEX

Anonymous block 3, 22

Declarative part 2

Exception-handling part 2, 11

Executable part 2, 11

PL/SQL 2, 27