Table of Contents

[CHAPTER 25: COMPILER 2](#_Toc417468646)

[Theory 2](#_Toc417468647)

[AIM 4](#_Toc417468648)

[Lab Exercise 25: COMPILER 5](#_Toc417468649)

[1. Compilation Parameters 6](#_Toc417468650)

[2. Native and Interpreted Code 18](#_Toc417468651)

[3. Conditional Compilation 27](#_Toc417468652)

[SUMMARY 38](#_Toc417468653)

[REFERENCES 39](#_Toc417468654)

[INDEX 40](#_Toc417468655)

# CHAPTER 25: COMPILER

## Theory

In the early chapters of this book, we cover the concept of PL/SQL Engine. PL/SQL Engine is responsible of compiling and running PL/SQL blocks (Anonymous or named). Therefore, the compiler is part of PL/SQL Engine. When a user submits a PL/SQL block text to the PL/SQL Engine, the Engine parse, compile and execute the block. If the text is a call to a pre-compiled object, then the Engine will ignore the compilation step.

PL/SQL compilation can be done either implicitly or explicitly. For example, when you create new FUNCTION, PROCEDURE, PACKAGE, TYPE, or TRIGGER, the compiler implicitly compile the object before storing it in the database. You can also do a compilation explicitly by using ALTER command. ALTER command follows the syntax shown for all Object Type.

**ALTER** *Object\_Type Object\_Name* **COMPILE;**

While compiling PL/SQL block, the compiler uses a set of variables to determine how to compile the object correctly. You can set these variables into three levels:

1. **SYSTEM**: use this option to assign the default values of the compiler variables.
2. **SESSION**: use this option when you want a single schema to have different variables values. This option overrides the previous option.
3. **OBJECT**: you can dictate the compiler to use specific variables values when compiling a single object. This option overrides all the previous options.

Some of the compiler's variables are predefined by Oracle. You allowed to use not to assign value for them. Beside that, you can define, use and assign your own variables.

One of the compiler variables is PLSQL\_CODE\_TYPE which determines if the compiler will generate intermediate code or native code. By default, the PL/SQL statements are compiled into an intermediate form, system code, which is stored in the catalog and interpreted at run time. This intermediate form is platform independent. Oracle, however, allows you to compile PL/SQL statements into native code (platform dependent) and store it into catalog. The native code need not to be interpreted at run time, so it runs faster. Please remember, this is applied only for PL/SQL statements; other SQL statements will not be affected.

In addition to the compiler parameters or variables, Oracle supports a set of compiler conditional directives: IF ELSE statements. Before compiling the object, the compiler reads these directives to determine which code is included and should be compiled; and which code is not. This option is good to customize the functionality of a PL/SQL application without removing source text. For example, if you have to write a single function that may run in different Oracle versions: then you may direct the compiler to compile and use the code that best run for the version selected.

## AIM

The AIM of the following exercise is to use PL/SQL compiler.

The steps involved will include:

* Compilation Parameters
* Native and Interpreted Code
* Conditional Compilation

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

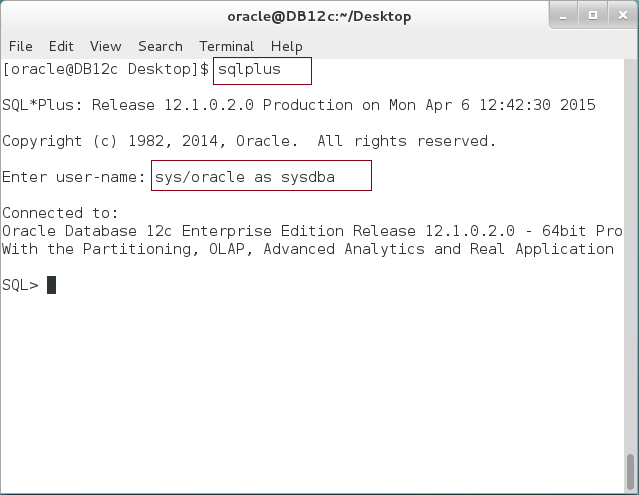
# Lab Exercise 25: COMPILER

|  |
| --- |
|  |

## Compilation Parameters

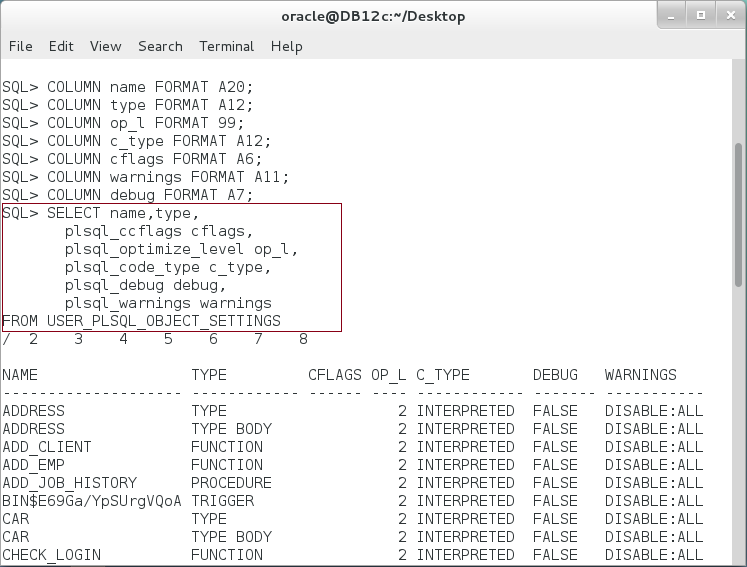
**Step 1:** Open the Terminal, open SQL\*Plus console and connect to hr schema.

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

****

**Step 2:** You can query the parameters in Object level using the following query:

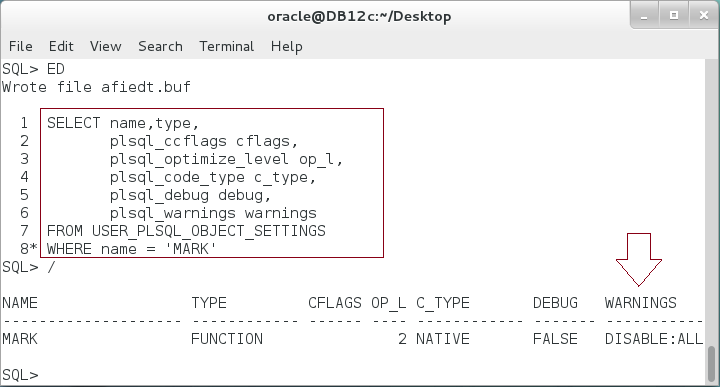
|  |  |
| --- | --- |
| Command | Description |
| DESC ALL\_PLSQL\_OBJECT\_SETTINGS |  |
|  |  |
| COLUMN name FORMAT A20; | Format columns |
| COLUMN type FORMAT A12; |
| COLUMN op\_l FORMAT 99; |
| COLUMN c\_type FORMAT A12; |
| COLUMN cflags FORMAT A6; |
| COLUMN warnings FORMAT A11; |
| COLUMN debug FORMAT A7; |
|  |  |
| SELECT name,type, | Query all "HR" objects with their compilation parameters. |
| plsql\_ccflags cflags, |
| plsql\_optimize\_level op\_l, |
| plsql\_code\_type c\_type, |
| plsql\_debug debug, |
| plsql\_warnings warnings |
| FROM **USER\_PLSQL\_OBJECT\_SETTINGS** |
| / |  |

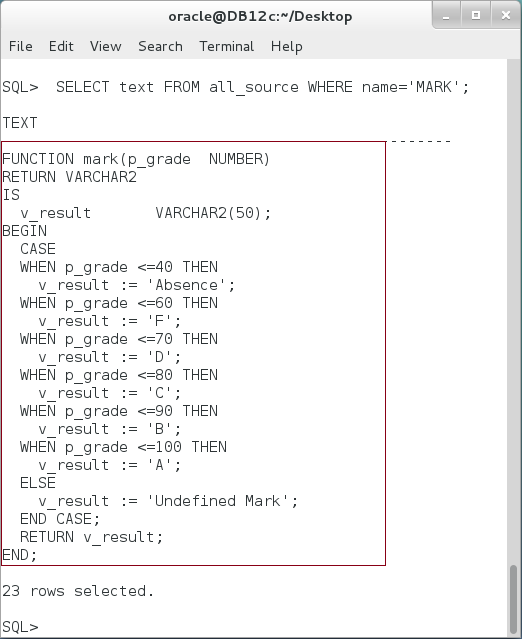
****

**Please note:** every object has its own compiler parameters.

**Step 3:** We are going to select one function, MARK. Execute the following queries:

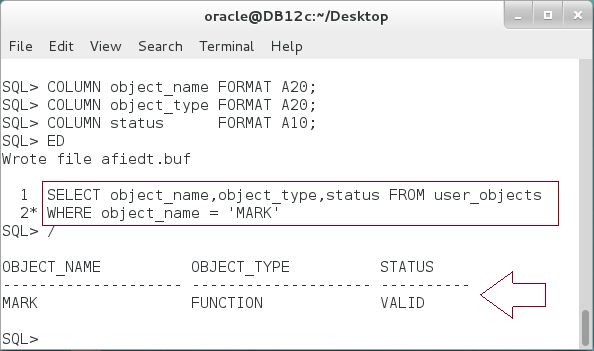
|  |  |
| --- | --- |
| Command | Description |
| SELECT name,type, | Return an employees' information |
| plsql\_ccflags cflags, |
| plsql\_optimize\_level op\_l, |
| plsql\_code\_type c\_type, | View compiler parameters of MARK function |
| plsql\_debug debug, |
| plsql\_warnings warnings |
| FROM USER\_PLSQL\_OBJECT\_SETTINGS |  |
| **WHERE name = 'MARK'** |  |
| / |  |
| SELECT text FROM all\_source WHERE name=**'MARK'** | View "**MARK**" function source code. |
| / |

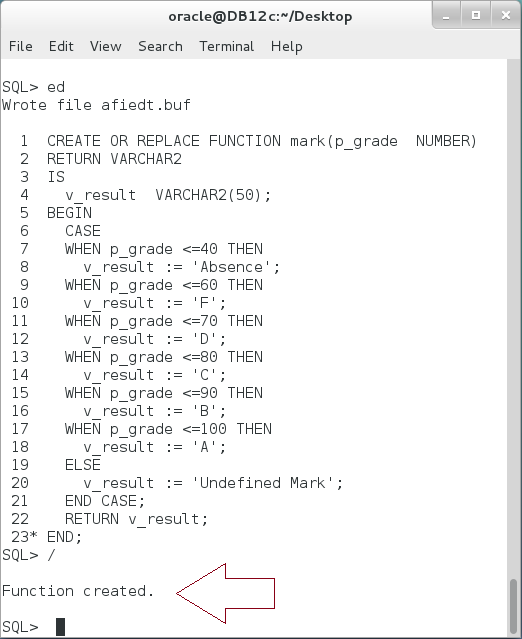
****

****

**Step 4:** MARK function was created and compiled previously. Its status is VALID. If you re-created the function, it created and compiled without error. :

|  |  |
| --- | --- |
| Command | Description |
| COLUMN object\_name FORMAT A20; |  |
| COLUMN object\_type FORMAT A20; |
| COLUMN status FORMAT A10; |
|  |
| **SELECT** object\_name,object\_type,status **FROM user\_objects** |  |
| WHERE object\_name = 'MARK' |  |
| / |  |
|  |  |
| **CREATE OR REPLACE FUNCTION mark(p\_grade NUMBER)** |
| **RETURN VARCHAR2** |
| IS |
| v\_result VARCHAR2(50); |  |
| BEGIN |
| CASE |
| WHEN p\_grade <=40 THEN |  |
| v\_result := 'Absence'; |  |
| WHEN p\_grade <=60 THEN |  |
| v\_result := 'F'; |  |
| WHEN p\_grade <=70 THEN |  |
| v\_result := 'D'; |  |
| WHEN p\_grade <=80 THEN |  |
| v\_result := 'C'; |  |
| WHEN p\_grade <=90 THEN |  |
| v\_result := 'B'; |  |
| WHEN p\_grade <=100 THEN |  |
| v\_result := 'A'; |  |
| ELSE |  |
| v\_result := 'Undefined Mark'; |  |
| END CASE; |  |
| RETURN v\_result; |  |
| END; |  |
| / |  |





**Step 5:** If you want to compile the function explicitly, execute the following DDL:

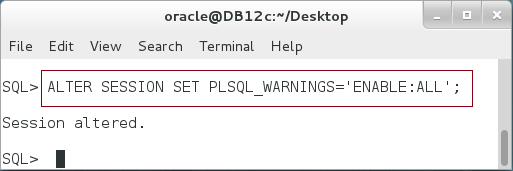
|  |  |
| --- | --- |
| Command | Description |
| ALTER FUNCTION mark COMPILE |  |
| / |  |

****

**Please note:** The function is compiled without error.

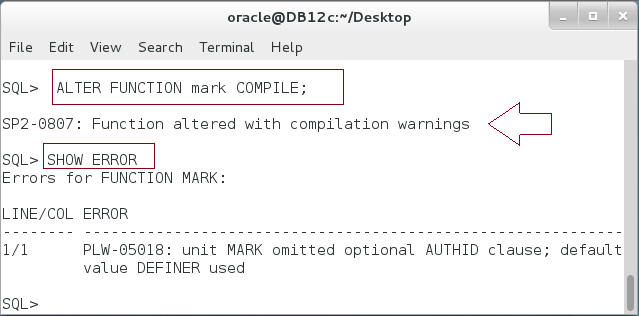
**Step 6:** Now, enable the compilation parameter PLSQL\_WARNINGS. We will change the parameter in session level. Execute the following DDL:

|  |  |
| --- | --- |
| Command | Description |
| ALTER SESSION SET PLSQL\_WARNINGS=**'ENABLE:ALL'** |  |
| / |  |

****

**Step 7:** Re-compile "MARK" function:

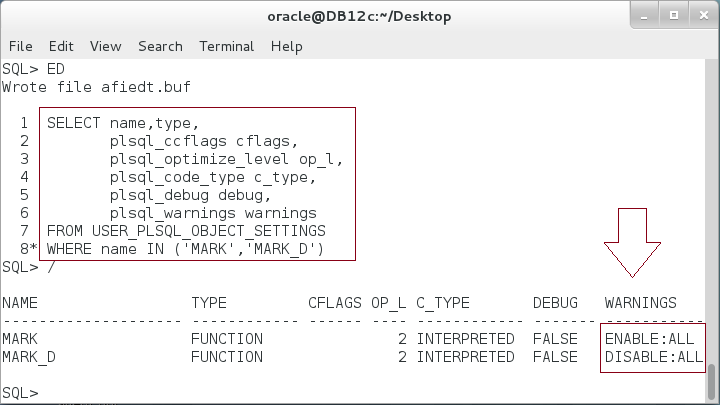
|  |
| --- |
| Command |
| ALTER FUNCTION mark COMPILE |
| / |
| SHOW ERROR |

****

**Please note:** When you re-compile the function, MARK, the function compiled but with compilation warnings. To show the workings, execute SHOW ERROR command.

**Step 8:** View the compilation parameters of MARK and MARK\_D functions as shown belwo:

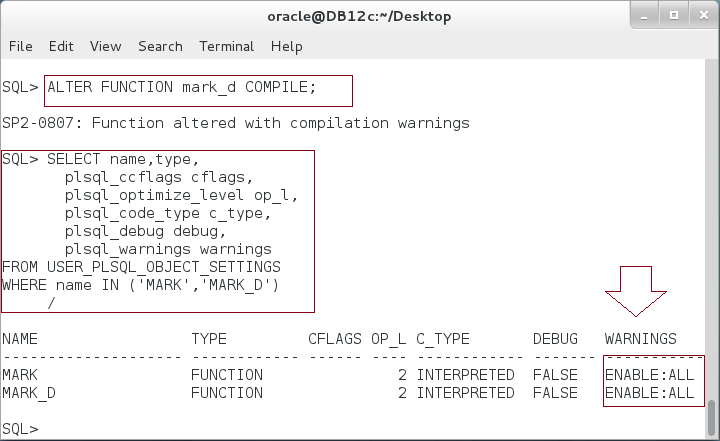
|  |
| --- |
| Command |
| ALTER FUNCTION mark COMPILE |
| SELECT name,type, |
| plsql\_ccflags cflags, |
| plsql\_optimize\_level op\_l, |
| plsql\_code\_type c\_type, |
| plsql\_debug debug, |
| plsql\_warnings warnings |
| FROM USER\_PLSQL\_OBJECT\_SETTINGS |
| **WHERE name IN ('MARK','MARK\_D')** |
| / |

****

**Please note:** MARK function parameter is now enabled but MARK\_D is still disabled. All function remain as they were until you recompile them.

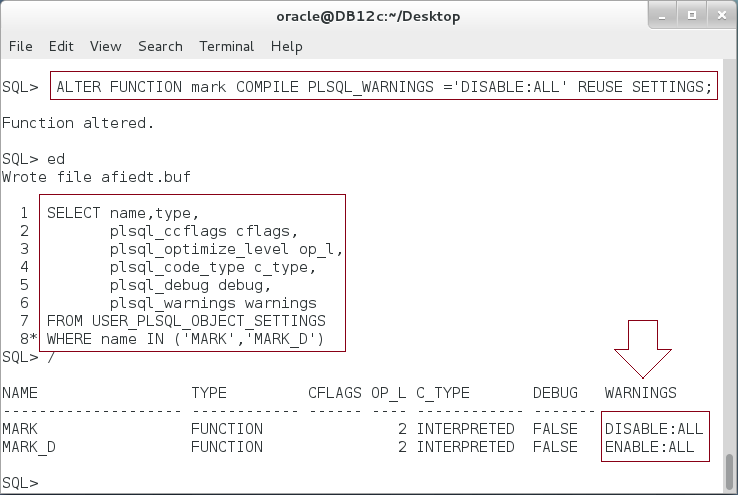
**Step 9:** Re-compile "MARK\_D" function and notice its parameters as shown below:

|  |
| --- |
| Command |
| ALTER FUNCTION **mark\_d** COMPILE |
| / |
| SELECT name,type, |
| plsql\_ccflags cflags, |
| plsql\_optimize\_level op\_l, |
| plsql\_code\_type c\_type, |
| plsql\_debug debug, |
| plsql\_warnings warnings |
| FROM USER\_PLSQL\_OBJECT\_SETTINGS |
| **WHERE name IN ('MARK','MARK\_D')** |
| / |

****

**Step 10:** You can compile a specific function with compilation parameter, it will override the session level compilation parameter. Execute the following DDL and query:

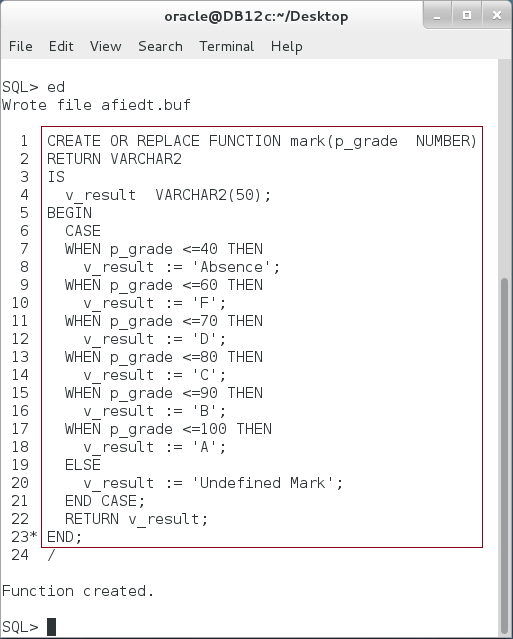
|  |
| --- |
| Command |
| ALTER FUNCTION mark COMPILE **PLSQL\_WARNINGS ='DISABLE:ALL' REUSE SETTINGS** |
| / |
| SELECT name,type, |
| plsql\_ccflags cflags, |
| plsql\_optimize\_level op\_l, |
| plsql\_code\_type c\_type, |
| plsql\_debug debug, |
| plsql\_warnings warnings |
| FROM USER\_PLSQL\_OBJECT\_SETTINGS |
| WHERE name IN ('MARK','MARK\_D') |
| / |

****

## Native and Interpreted Code

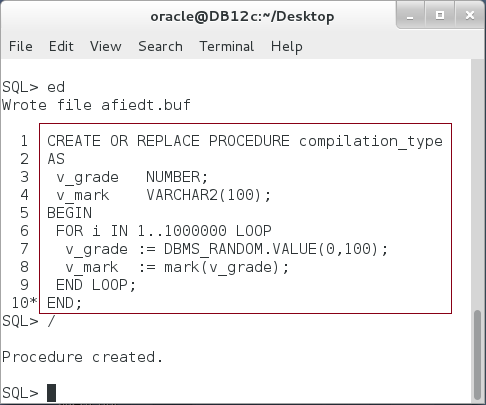
**Step 1:** We will use MARK function. The function has no SQL commands. It is merely PL/SQL control statement:

|  |
| --- |
| Command |
| **CREATE OR REPLACE FUNCTION mark(p\_grade NUMBER)** |
| **RETURN VARCHAR2** |
| IS |
| v\_result VARCHAR2(50); |
| BEGIN |
| CASE |
| WHEN p\_grade <=40 THEN |
| v\_result := 'Absence'; |
| WHEN p\_grade <=60 THEN |
| v\_result := 'F'; |
| WHEN p\_grade <=70 THEN |
| v\_result := 'D'; |
| WHEN p\_grade <=80 THEN |
| v\_result := 'C'; |
| WHEN p\_grade <=90 THEN |
| v\_result := 'B'; |
| WHEN p\_grade <=100 THEN |
| v\_result := 'A'; |
| ELSE |
| v\_result := 'Undefined Mark'; |
| END CASE; |
| RETURN v\_result; |
| END; |
| / |

****

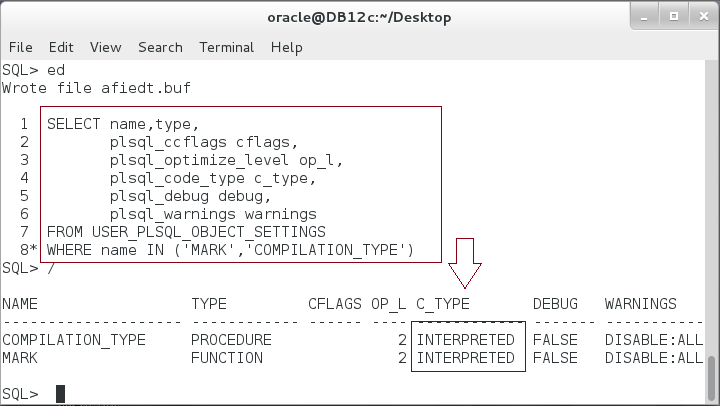
**Step 2:** We will create a function "compilation\_type" that loop through 1 million iteration to calculate the mark:

|  |  |
| --- | --- |
| Command | Description |
| **CREATE OR REPLACE PROCEDURE compilation\_type** |  |
| AS |  |
| v\_grade NUMBER; |  |
| v\_mark VARCHAR2(100); |  |
| BEGIN |  |
| **FOR i IN 1..1000000 LOOP** |  |
| v\_grade := DBMS\_RANDOM.VALUE(0,100); |  |
| v\_mark := mark(v\_grade); |  |
| **END LOOP;** |
| END; |
| / |  |



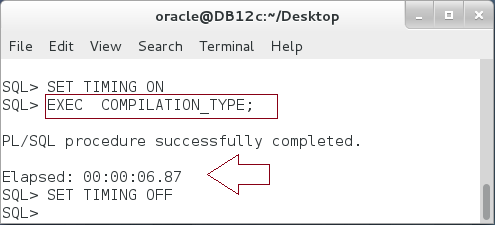
**Step 3:** By default, all objects are compiled into intermediate code, **INTERPRETED**:

|  |  |
| --- | --- |
| Command | Description |
| SELECT name,type, |  |
| plsql\_ccflags cflags, |
| plsql\_optimize\_level op\_l, |  |
| plsql\_code\_type c\_type, |  |
| plsql\_debug debug, |  |
| plsql\_warnings warnings |  |
| FROM USER\_PLSQL\_OBJECT\_SETTINGS |  |
| **WHERE name IN ('MARK','COMPILATION\_TYPE')** |  |
| / |  |

****

**Step 4:** Test the time of INTERPRETED functions as shown below:

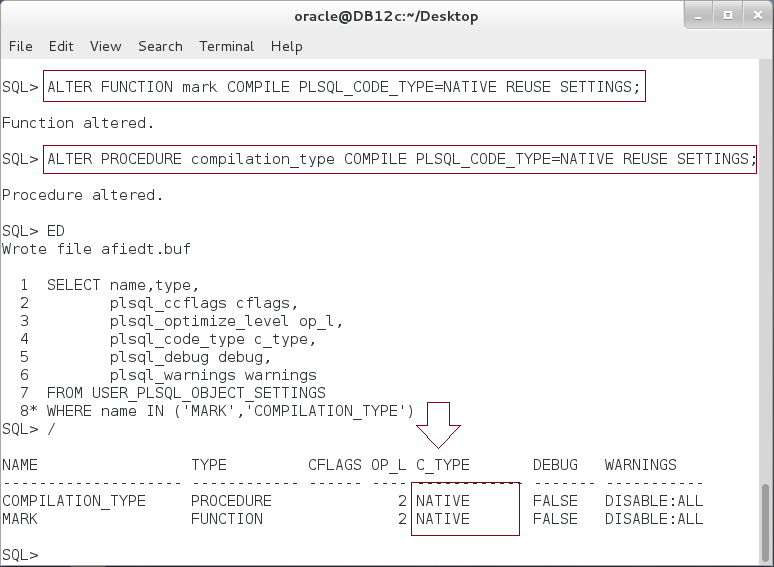
|  |  |
| --- | --- |
| Command | Description |
| SET TIMING ON |  |
| EXEC COMPILATION\_TYPE; |  |
| SET TIMING OFF |  |



**Please note:** You may need to execute the function twice.

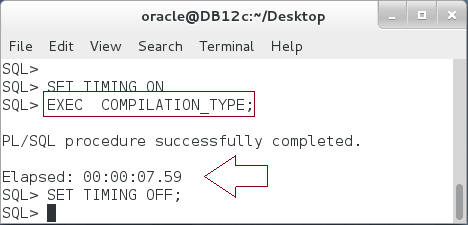
**Step 5:** Now, re-compile the methods to be NATIVE:

|  |
| --- |
| Line |
| **ALTER FUNCTION mark COMPILE PLSQL\_CODE\_TYPE=NATIVE REUSE SETTINGS** |
| / |
| **ALTER PROCEDURE compilation\_type COMPILE PLSQL\_CODE\_TYPE=NATIVE REUSE SETTINGS** |
| / |
| SELECT name,type, |
| plsql\_ccflags cflags, |
| plsql\_optimize\_level op\_l, |
| plsql\_code\_type c\_type, |
| plsql\_debug debug, |
| plsql\_warnings warnings |
| FROM USER\_PLSQL\_OBJECT\_SETTINGS |
| **WHERE name IN ('MARK','COMPILATION\_TYPE')** |
| / |



**Step 6:** Test the time required to execute NATIVE function:

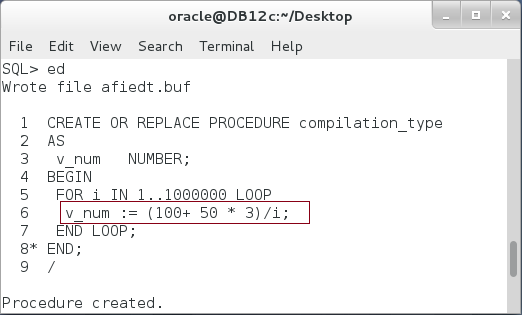
|  |  |
| --- | --- |
| Line |  |
| SET TIMING ON |  |
| **EXEC COMPILATION\_TYPE;** |  |
| SET TIMING OFF |  |



**Please note:** the result is not expected. The native code require somehow similar to the interpreted code!

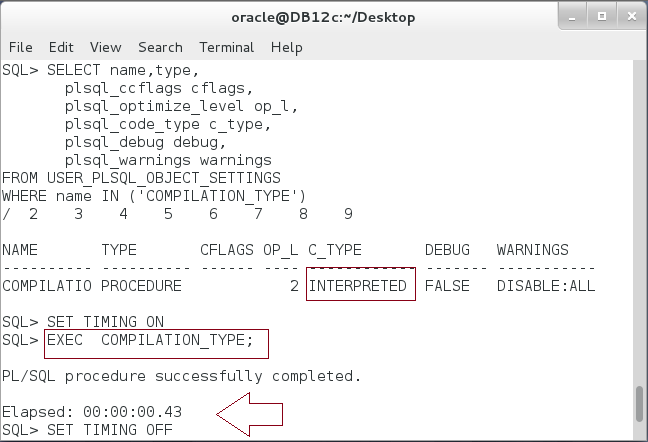
**Step 7:** Even when you repeat the test with a simple iteration method:

|  |
| --- |
| Line |
| CREATE OR REPLACE PROCEDURE compilation\_type |
| AS |
| v\_num NUMBER; |
| BEGIN |
| FOR i IN 1..1000000 LOOP |
| **v\_num := (100+ 50 \* 3)/i;** |
| END LOOP; |
| END; |
| / |



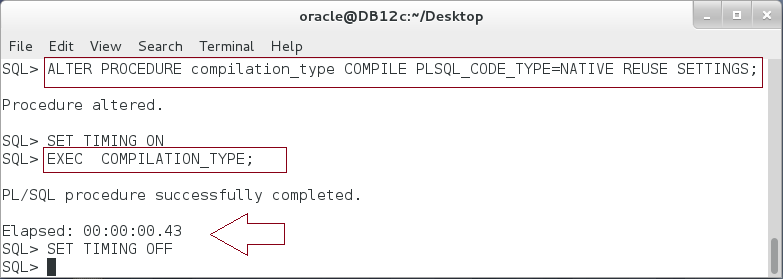
**Step 8:** Test the timing of interpreted method:

|  |
| --- |
| Line |
| SELECT name,type, |
| plsql\_ccflags cflags, |
| plsql\_optimize\_level op\_l, |
| plsql\_code\_type c\_type, |
| plsql\_debug debug, |
| plsql\_warnings warnings |
| FROM USER\_PLSQL\_OBJECT\_SETTINGS |
| WHERE name IN ('COMPILATION\_TYPE') |
| / |
| SET TIMING ON |
| **EXEC COMPILATION\_TYPE;** |
| SET TIMING OFF |



**Step 9:** Test the timing of native method:

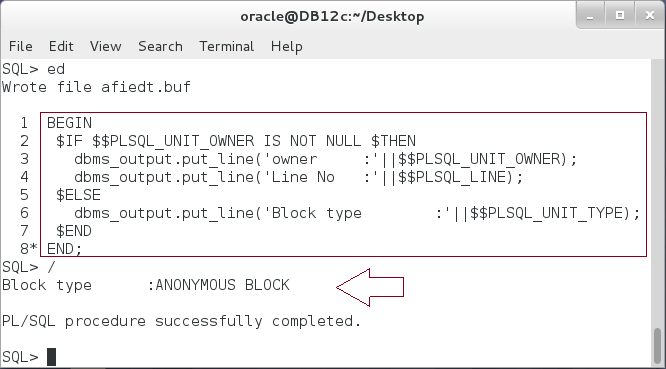
|  |
| --- |
| Line |
| **ALTER PROCEDURE compilation\_type COMPILE PLSQL\_CODE\_TYPE=NATIVE REUSE SETTINGS** |
| / |
| SET TIMING ON |
| **EXEC COMPILATION\_TYPE;** |
| SET TIMING OFF |

**Please note:** again, the time required for native and interpreted code is similar!

## Conditional Compilation

**Step 1:** PL/SQL compiler considers the conditional directives before compiling the PL/SQL block. Execute the following:

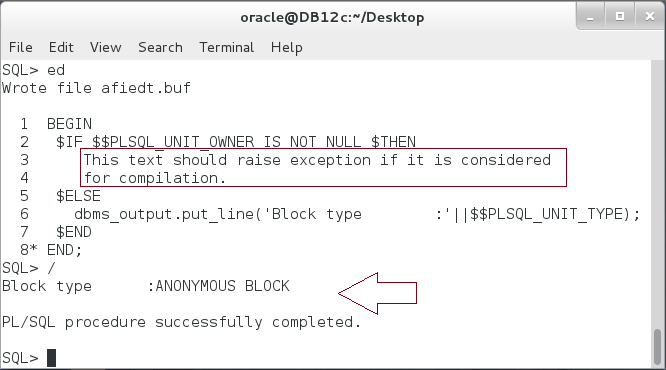
|  |  |
| --- | --- |
| Line | Description |
| BEGIN | The directive keywords start with **$.**  The directive variables start with **$$** |
| **$IF** **$$PLSQL\_UNIT\_OWNER** IS NOT NULL **$THEN** |
| dbms\_output.put\_line('owner :'||**$$PLSQL\_UNIT\_OWNER**); |
| dbms\_output.put\_line('Line No:'||**$$PLSQL\_LINE**); |
| **$ELSE** |
| dbms\_output.put\_line('Block type:'||**$$PLSQL\_UNIT\_TYPE**); |
| **$END** |
| END; |
| / |



**Please note:** In the previous block, what is really consider for compiling is determined based on $$PLSQL\_UNIT\_OWNER value. Since it is NULL, then any text written between $IF and $ELSE will not considered for compilation.

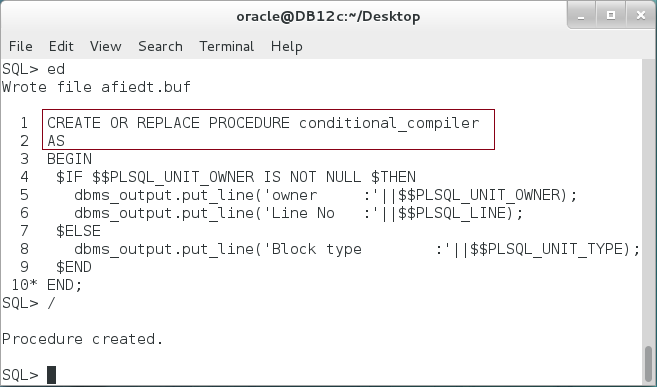
**Step 2:** To test the previous note, write any text between $IF and $ELSE as shown below:

|  |  |
| --- | --- |
| Line | Description |
| BEGIN |  |
| $IF $$PLSQL\_UNIT\_OWNER IS NOT NULL $THEN |
| **This text should raise exception if it is considered** |
| **for compilation.** |
| $ELSE |
| dbms\_output.put\_line('Block type:'||$$PLSQL\_UNIT\_TYPE); |
| $END |
| END; |
| / |

****

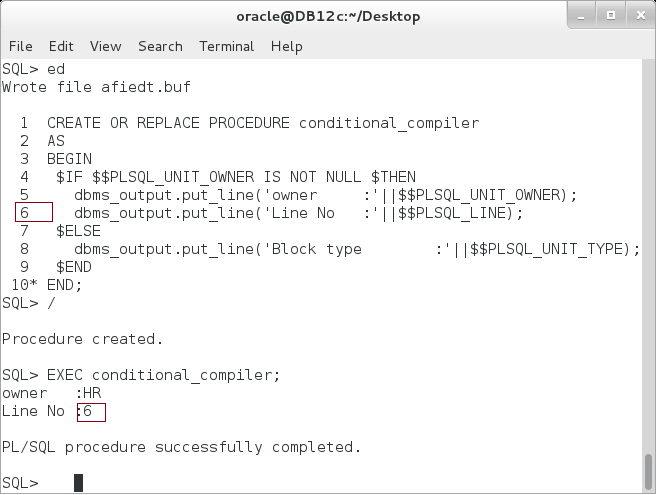
**Step 3:** Create the previous block as a stored procedure:

|  |
| --- |
| Line |
| **CREATE OR REPLACE PROCEDURE conditional\_compiler** |
| **AS** |
| BEGIN |
| $IF $$PLSQL\_UNIT\_OWNER IS NOT NULL $THEN |
| dbms\_output.put\_line('owner :'||$$PLSQL\_UNIT\_OWNER); |
| dbms\_output.put\_line('Line No :'||$$PLSQL\_LINE); |
| $ELSE |
| dbms\_output.put\_line('Block type :'||$$PLSQL\_UNIT\_TYPE); |
| $END |
| END; |
| / |

****

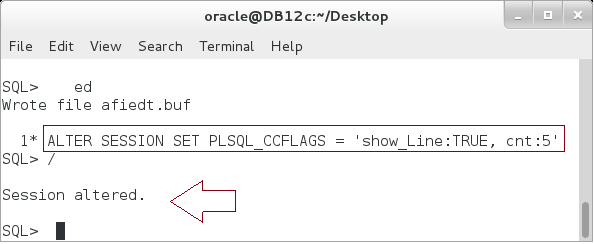
**Step 4:** You can test the method as shown below:

|  |
| --- |
| Line |
| EXEC conditional\_compiler |
| / |



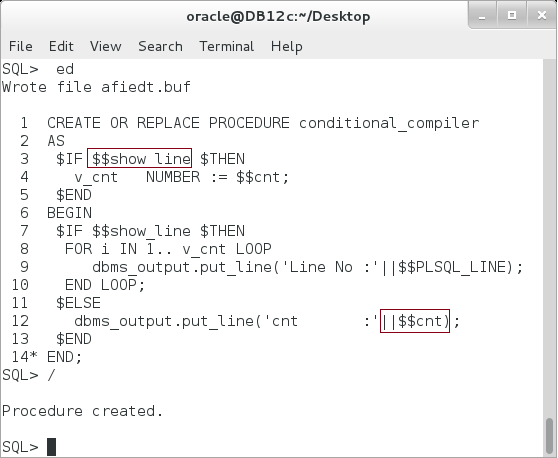
**Step 5:** You can also define your directive session variables: **show\_line**, and **cnt**:

|  |  |
| --- | --- |
| Line | Description |
| ALTER SESSION SET PLSQL\_CCFLAGS = **'show\_Line:TRUE, cnt:5**' |  |
| / |



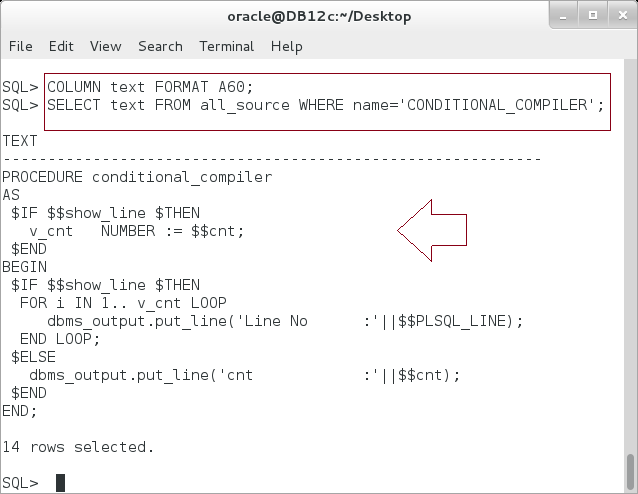
**Step 6:** You can know use these variables to determine the compiler behavior as shown below:

|  |
| --- |
| Line |
| CREATE OR REPLACE PROCEDURE conditional\_compiler |
| AS |
| $IF **$$show\_line** $THEN |
| v\_cnt NUMBER := **$$cnt**; |
| $END |
| BEGIN |
| $IF $$show\_line $THEN |
| FOR i IN 1.. v\_cnt LOOP |
| dbms\_output.put\_line('Line No :'||$$PLSQL\_LINE); |
| END LOOP; |
| $ELSE |
| dbms\_output.put\_line('cnt :'**||$$cnt**); |
| $END |
| END; |
| / |



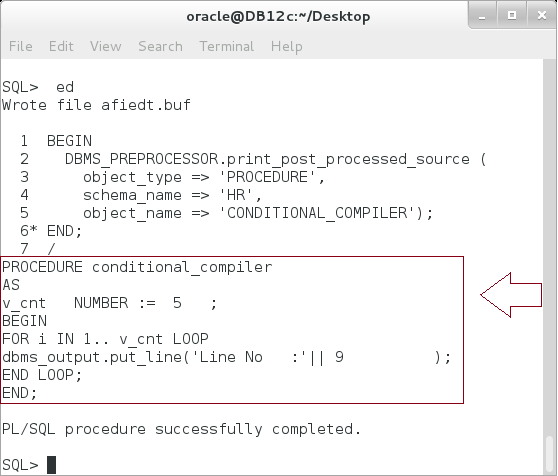
**Step 7:** If you search for the source code of the previous method, you will still find the compilation directives:

|  |  |
| --- | --- |
| Line | Description |
| COLUMN text FORMAT A60; |  |
| SELECT text FROM all\_source WHERE name='CONDITIONAL\_COMPILER' |  |
| / |  |



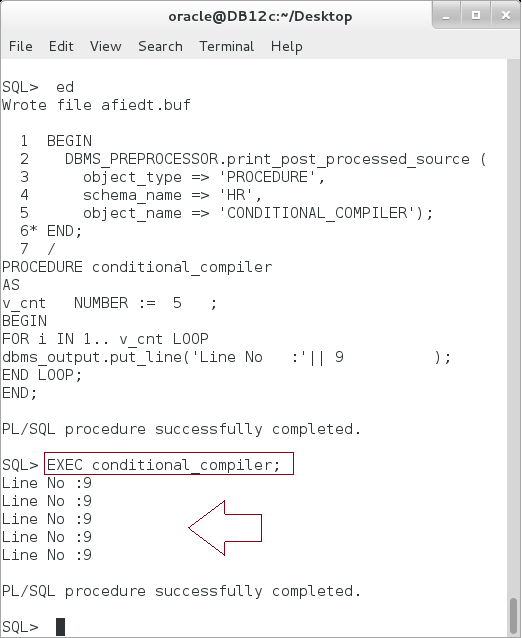
**Step 8:** To see what the method looks like just before compilation, use DBMS\_PREPROCESSOR package as shown below:

|  |
| --- |
| Line |
| BEGIN |
| **DBMS\_PREPROCESSOR.print\_post\_processed\_source** ( |
| object\_type => 'PROCEDURE', |
| schema\_name => 'HR', |
| object\_name => 'CONDITIONAL\_COMPILER'); |
| END; |
| / |



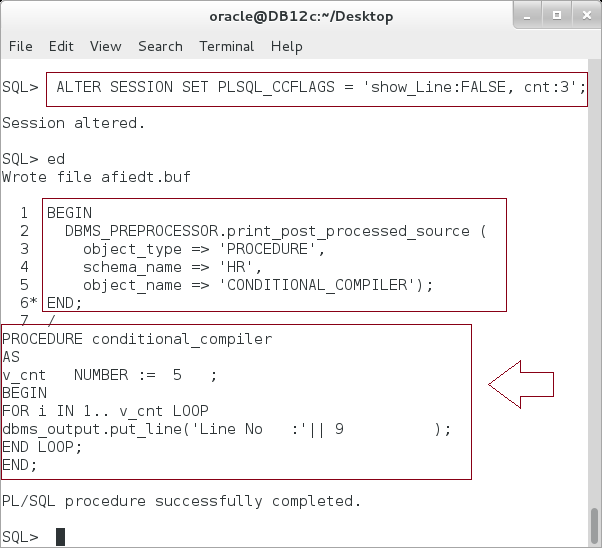
**Step 9:** Test the previous method:

|  |
| --- |
| Line |
| EXEC conditional\_compiler; |



**Step 10:** Alter the session variable and check the method text:

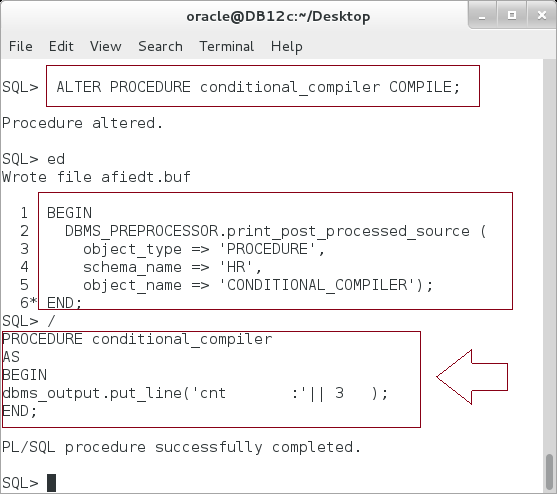
|  |
| --- |
| Line |
| **ALTER SESSION SET PLSQL\_CCFLAGS** = 'show\_Line:FALSE, cnt:3' |
| / |
| BEGIN |
| **DBMS\_PREPROCESSOR.print\_post\_processed\_source** ( |
| object\_type => 'PROCEDURE', |
| schema\_name => 'HR', |
| object\_name => 'CONDITIONAL\_COMPILER'); |
| END; |
| / |



**Please note:** The text is still as it was before changing the variables! **Why?**

**Step 11:** Re-compile the method and check its source code again:

|  |
| --- |
| Line |
| **ALTER PROCEDURE conditional\_compiler COMPILE** |
| / |
| BEGIN |
| **DBMS\_PREPROCESSOR.print\_post\_processed\_source** ( |
| object\_type => 'PROCEDURE', |
| schema\_name => 'HR', |
| object\_name => 'CONDITIONAL\_COMPILER'); |
| END; |
| / |



# SUMMARY

PL/SQL compiler is part of PL/SQL Engine. Its main duty is to take the source code and transform it to either intermediated byte code, or native code. The intermediated code is platform independent and should be then interpreted to platform machine language at runtime. While native code does not need to be interpreted. However, this is applied only for PL/SQL code; other SQL commands do not affected. In our lab, the time difference between interpreted and native code is not significant. You can change the way the compiler acts by changing the compilation parameters: a set of variable that attached to every compiled object. You can change these parameters on three levels: SYSTEM, SESSION and the Object itself by ALTER statement. Finally, the compiler can be directed to compile some code and ignore other code using a set of compiler variables and directive command; IF ELSE. This feature is useful to make the same code runs on different Oracle versions.

After completing this lab exercise, you should be able to use PL/SQL compiler and its features.

# REFERENCES

* http://docs.oracle.com/database/121/LNPLS/tuning.htm#LNPLS914
* http://oracle-base.com/articles/10g/conditional-compilation-10gr2.php
* http://docs.oracle.com/database/121/LNPLS/fundamentals.htm#LNPLS294
* http://docs.oracle.com/database/121/REFRN/refrn10253.htm#REFRN10253
* http://docs.oracle.com/database/121/LNPLS/overview.htm#LNPLS0121
* http://oracle-base.com/articles/9i/plsql-native-compilation-9i.php

# INDEX

compiler parameters 3, 7, 8

intermediate code 2, 20

native code 2, 23, 38

PL/SQL Engine 2, 38