Table of Contents

[CHAPTER 9: CONTROL STATEMENTS 2](#_Toc416705340)

[Theory 2](#_Toc416705341)

[AIM 5](#_Toc416705342)

[Lab Exercise 9: CONTROL STATEMENTS 6](#_Toc416705343)

[1. Basic LOOP 7](#_Toc416705344)

[2. WHILE LOOP 17](#_Toc416705345)

[3. FOR LOOP 19](#_Toc416705346)

[4. IF Statements 25](#_Toc416705347)

[5. CASE Statements 30](#_Toc416705348)

[6. TCL Statements 36](#_Toc416705349)

[SUMMARY 41](#_Toc416705350)

[REFERENCES 42](#_Toc416705351)

[INDEX 43](#_Toc416705352)

# CHAPTER 9: CONTROL STATEMENTS

## Theory

In this chapter, you will explore the control statements and Transaction Control Language (TCL). Some of control statements already shown in the previous chapters, but in this chapter, you would examine various options of control statements.

PL/SQL has three types of control statements:

* Loop statements

**Loop iteration control** using: CONTINUE and EXIST

* + Basic LOOP statement
  + WHILE LOOP statement
  + FOR LOOP statement
* Conditional Statements
  + IF statement
  + CASE statement

PL/SQL also has four TCL statements:

* COMMIT
* ROLLBACK
* SAVEPOINT
* SET TRANSACTION

In general, all loop statements repeat/iterate through a set of statements until a pre-defined condition raise. The basic LOOP statement is a more generic and flexible statement. It may substitute all other loop statements but it requires more coding. For example, it is the only way you can implement "do .. while" loop known in other programming languages. The basic LOOP syntax is as follow:

LOOP

......

EXIT;

......

END LOOP;

.......

<<lable>>

LOOP

......

EXIT WHEN condition;

......

END LOOP <<lable>>;

.....

<<outer>> LOOP

........

<<inner>> LOOP

..........

EXIT inner WHEN condition;

EXIT outer WHEN condition;

...........

END LOOP <<inner>>;

.........

..........

END LOOP <<outer>>;

...........

You must include EXIT statement with either WHEN condition or without to exit the loop to avoid entering in infinite loop. The EXIT statement moves the execution pointer to the first statement after the loop END. Loop statements can be nested. You need labels in this case to be able to exist the outer loop from the inner loop; otherwise, you can't exit the outer loop from **inside** the inner loop. The EXIT statement can be placed anywhere inside the loop. Thus, if you put it exactly after LOOP statement, then it mimics WHILE LOOP and FOR LOOP statement. If you put it to be the last statement in loop, it mimics "do-while" syntax known in other programming languages.

You may also use "CONTINUE" or "CONTINUE WHEN" keywords to escape the current iteration and move the execution pointer to the LOOP statement.

<<lable>>

LOOP

......

CONTINUE WHEN condition;

......

END LOOP <<lable>>;

.....

WHILE LOOP and FOR LOOP are special cases of the basic LOOP statement. WHILE LOOP iterates until its condition turns to FALSE. Therefore, you must insure that the condition turns to FALSE in the content of the loop; otherwise, WHILE LOOP will iterate forever. The typical WHILE LOOP looks like the following figure:

The FOR LOOP statement runs one or more statements while the loop index is in a specified range. FOR LOOP has the structure show above. The "index" is implicit variable which is declared automatically. Its scope is limited inside the FOR LOOP content. The index is automatically increased by 1 for each iteration from "min" to "max", if "REVERSE" keyword does not exist. If "REVERSE" keyword exists, the "index" automatically decreased by 1 for each iteration from "max" to "min".

<<lable>>

FOR index IN [REVERSE] min .. max LOOP

......

...

END LOOP <<lable>>;

<<lable>>

WHILE condition LOOP

......

...

END LOOP <<lable>>;

A transaction is a sequence of one or more SQL statements that Oracle Database treats as a unit: either all of the statements are performed, or none of them is. Transaction Control Language (TCL) helps you to control the transactions inside a PL/SQL block. Thus, multiple users can work on the database concurrently, each user can see a consistent version of data and that all changes are applied in the right order. In general, a PL/SQL block does not commit the data unless you execute COMMIT statement. PL/SQL has also SAVEPOINT keyword that marks the current point in the processing of a transaction. Savepoints let you roll back part of a transaction instead of the whole transaction. The ROLLBACK statement ends the current transaction and undoes any changes made during that transaction. If you specify the name of a SAVEPOINT in a ROLLBACK statement, you will undo only the changes made after that save point.

BEGIN

......

...

..

ROLLBACK;

......

...

END;

BEGIN

......

SAVEPOINT a;

....

...

ROLLBACK TO a;

......

END;

BEGIN

......

...

COMMIT;

......

...

....

END;

## AIM

The AIM of the following exercise is to demonstrate how to use control statements in PL/SQL block.

The steps involved will include:

* Basic LOOP
* WHILE LOOP
* FOR LOOP
* IF Statements
* CASE Statements
* TCL Statements

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:

35 minutes

# Lab Exercise 9: CONTROL STATEMENTS

|  |
| --- |
|  |

## Basic LOOP

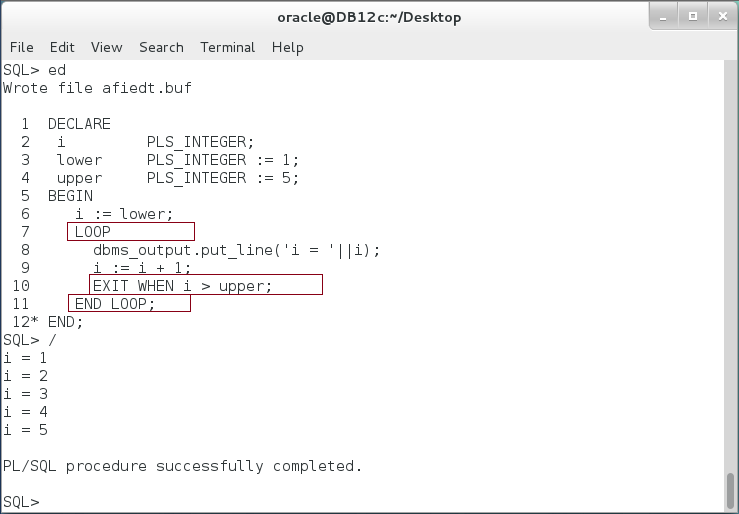
**Step 1:** Open the Terminal, open SQL\*Plus console and connect to hr schema.

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

****

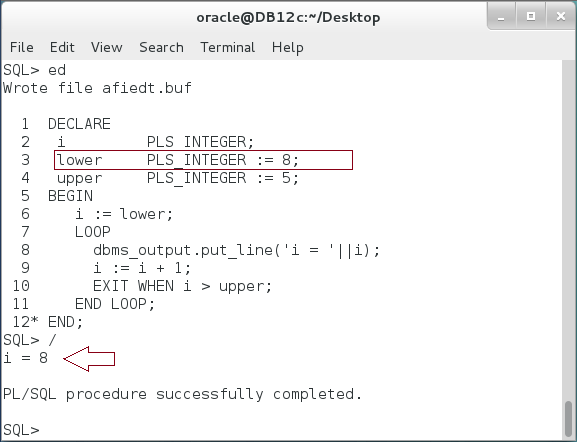
**Step 2:** Execute the following block:

|  |  |
| --- | --- |
| Command | Description |
| DECLARE |  |
| i PLS\_INTEGER; |  |
| lower PLS\_INTEGER := 1; |  |
| upper PLS\_INTEGER := 5; |  |
| BEGIN |  |
| i := lower; | Loop from lower value =1 to  upper value = 5  Note where "EXIT" statement is placed. |
| **LOOP** |
| dbms\_output.put\_line('i = '||i); |
| i := i + 1; |
| **EXIT WHEN i > upper;** |
| **END LOOP;**  END; |  |
|  |
| / |  |

****

**Step 3:** Modify the previous block as the following:

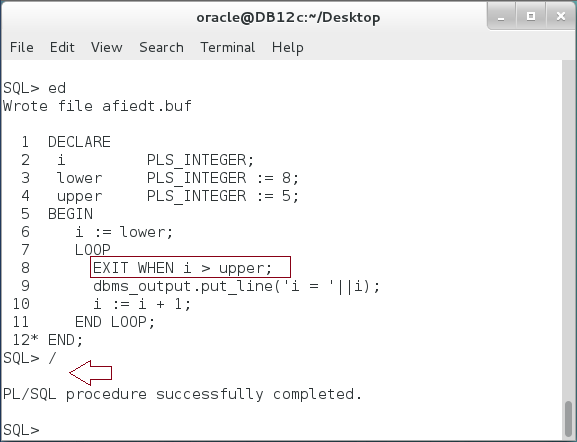
|  |  |
| --- | --- |
| Command | Description |
| DECLARE |  |
| i PLS\_INTEGER; |  |
| **lower PLS\_INTEGER := 8**; | Make the lower value greater than the upper. |
| upper PLS\_INTEGER := 5; |
| BEGIN |  |
| i := lower; | Although the lower value is greater, the loop must be executed at least once. |
| LOOP |
| dbms\_output.put\_line('i = '||i); |
| i := i + 1; |
| EXIT WHEN i > upper; |
| END LOOP;  END; |  |
|  |
| / |  |



**Please note:** By placing "EXIT" statement at the end of a LOOP, you mimics a "do-while" structure in which the loop must be executed at least once.

**Step 4:** Change the previous block again as shown below:

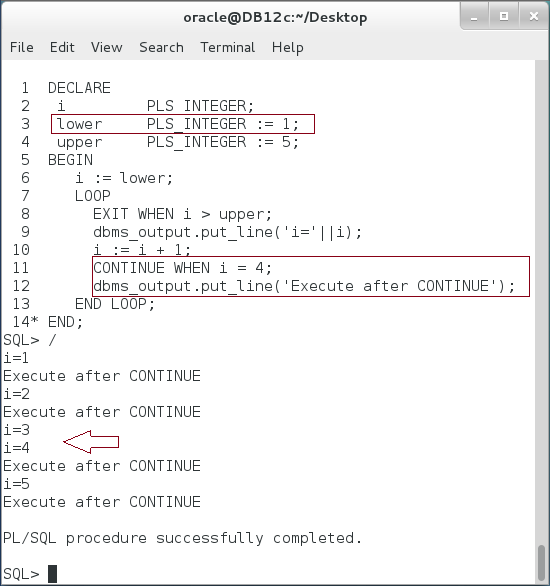
|  |  |
| --- | --- |
| Command | Description |
| DECLARE |  |
| i PLS\_INTEGER; |  |
| lower PLS\_INTEGER := 8; |  |
| upper PLS\_INTEGER := 5; |  |
| BEGIN |  |
| i := lower; | When "EXIT" statement is placed first, it prevents executing the body of the loop when the condition is not met. |
| LOOP |
| **EXIT WHEN i > upper;** |
| dbms\_output.put\_line('i = '||i); |
| i := i + 1; |
| END LOOP;  END; |  |
|  |
| / |  |

****

**Please note:** Using this structure, basic LOOP mimics other loop statements like WHILE and FOR LOOP.

**Step 5:** Modify a previous block as the following:

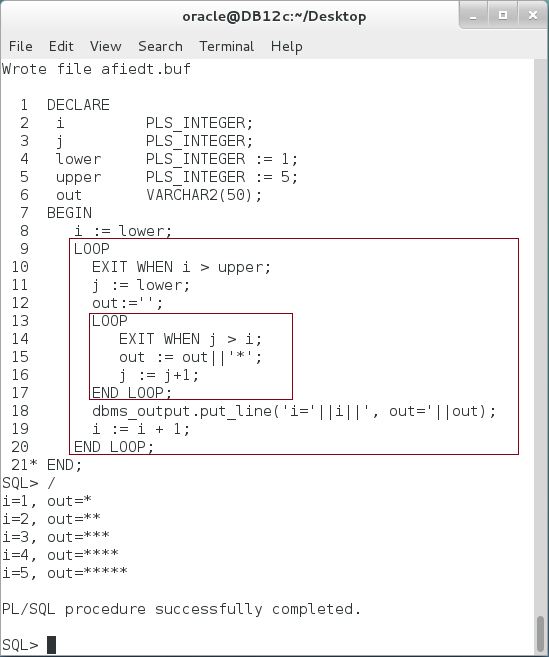
|  |  |
| --- | --- |
| Command | Description |
| DECLARE |  |
| i PLS\_INTEGER; |  |
| lower PLS\_INTEGER **:= 1**; |  |
| upper PLS\_INTEGER := 5; |  |
| BEGIN |  |
| i := lower; |  |
| LOOP |  |
| EXIT WHEN i > upper; |  |
| dbms\_output.put\_line('i='||i); | CONTINUE will escape executing the remainder of the LOOP when the condition met. |
| i := i + 1; |
| **CONTINUE WHEN i = 4;** |
| **dbms\_output.put\_line('Execute after CONTINUE');** |
| END LOOP; |
| END; |
| / |



**Please note**: The output statement after CONTINUE will be executed for all iterations except when i=4. **Explain the output?**

**Step 6:** Modify the previous block as shown below:

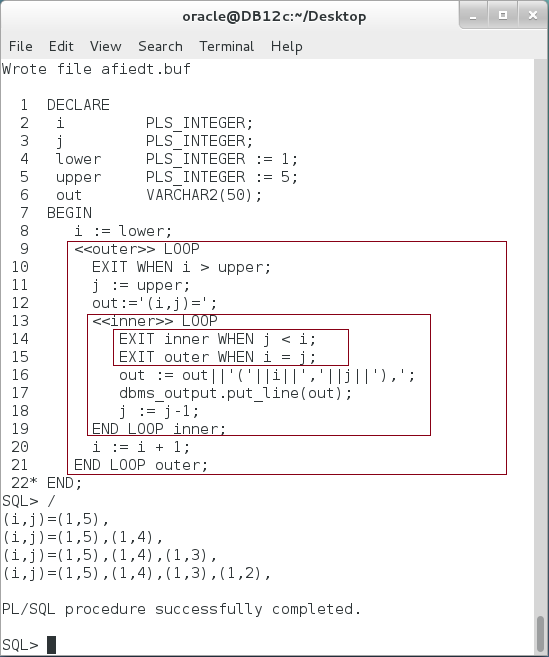
|  |  |
| --- | --- |
| Command | Description |
| DECLARE |  |
| i PLS\_INTEGER; |  |
| j PLS\_INTEGER; |  |
| lower PLS\_INTEGER := 1; |  |
| upper PLS\_INTEGER := 5; |  |
| out VARCHAR2(50); |  |
| BEGIN |  |
| i := lower; |  |
| **LOOP** | Basic LOOPs can be nested. |
| EXIT WHEN i > upper; |
| j := lower; |
| out:=''; |
| **LOOP** |
| EXIT **WHEN j > i;** |
| out := out||'\*'; |
| j := j+1; |
| **END LOOP;** |
| dbms\_output.put\_line('i='||i||', out='||out); |
| i := i + 1; |
| **END LOOP;** |
| END; |  |
| / |  |

****

**Please note:** The inner LOOP runs until it reach the outer LOOP's index value "i". **What if you want to stop the outer LOOP from inside the inner LOOP?**

**Step 7:** If you want to stop the outer loop from the inner loop, you must "label" the outer loop. In addition, you may want to navigate a loop in reverse. In this case, run the iteration from the "upper" to the "lower" as shown in the following block:

|  |  |
| --- | --- |
| Command | Description |
| DECLARE |  |
| i PLS\_INTEGER; |  |
| j PLS\_INTEGER; |  |
| lower PLS\_INTEGER := 1; |  |
| upper PLS\_INTEGER := 5; |  |
| out VARCHAR2(50); |  |
| BEGIN |  |
| i := lower; |  |
| **<<outer>> LOOP** |  |
| EXIT WHEN i > upper; |  |
| j := upper; |  |
| out:='(i,j)='; |  |
| **<<inner>> LOOP** |  |
| **EXIT inner WHEN j < i;** | Exit the outer LOOP from inner LOOP. |
| **EXIT outer WHEN i = j;** |
| out := out||'('||i||','||j||'),'; |  |
| dbms\_output.put\_line(out); |  |
| j := j-1; |  |
| **END LOOP inner;** |  |
| i := i + 1; |  |
| **END LOOP outer;** |  |
| END; |  |
| / |  |

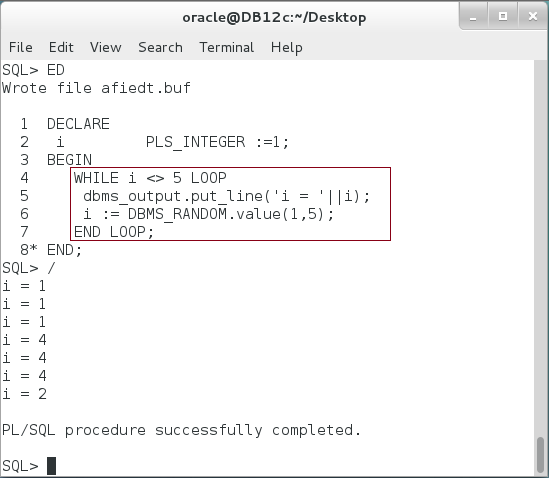


**Please note:** Basic LOOP can be create a very complex algorithm because of its flexibility. **Try to implement a Bubble Sort algorithm using Basic LOOP.**

## WHILE LOOP

**Step 1:** WHILE LOOP is a special case of basic LOOP in which a condition is checked before entering a LOOP. You must ensure that the condition turns to FALSE somewhere inside WHILE LOOP body to avoid infinite loop. Execute the following block:

|  |  |
| --- | --- |
| Line | Description |
| DECLARE |  |
| i PLS\_INTEGER :=1; |  |
| BEGIN |  |
| **WHILE i <> 5 LOOP** | **Loop until i =5** |
| dbms\_output.put\_line('i = '||i); |  |
| **i := DBMS\_RANDOM.value(1,5);** | Get a random value between 1 and 5. |
| **END LOOP;** |  |
| END; |  |
| / |  |

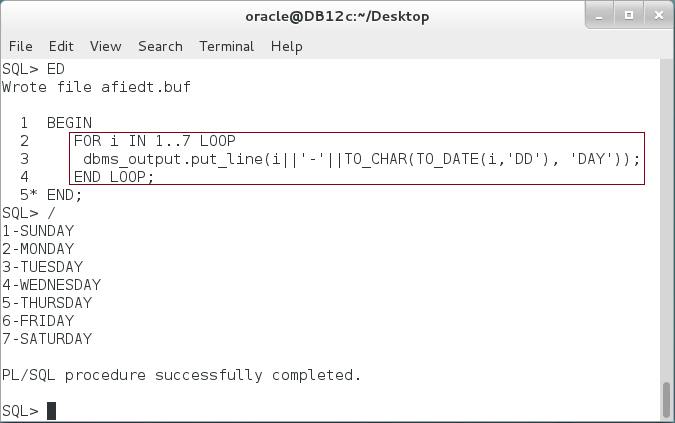


**Please note:** The upper limit of iterations is not determined in advance. WHILE LOOP may loop forever if you don't violate the condition.

## FOR LOOP

**Step 1:** The record is a composite data type which consists of one or more fields. The fields may consist of scalar data type as well as other composite data types. In this block, you will create a simple record type "address" which consists of scalar data types. Execute the following block:

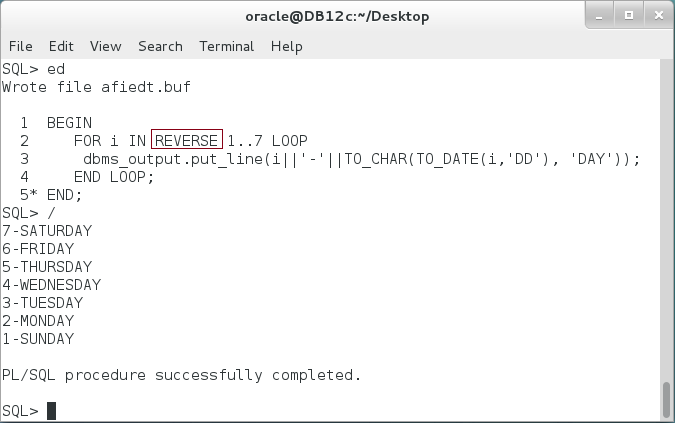
|  |  |
| --- | --- |
| Line | Description |
| BEGIN |  |
| FOR i IN 1..7 LOOP | FOR LOOP automatically runs implicit variable from 1 to 7 |
| dbms\_output.put\_line  (i||'-'||TO\_CHAR(TO\_DATE(i,'DD'), 'DAY')); | The scope of **i** is inside the FOR LOOP body. |
| END LOOP; |  |
| END; |  |
| / |  |

****

**Please note:** FOR LOOP simplifies the iterate makes you focus on the purpose of the loop. You don't need to define, increase and check the loop index "i". All of that are done for you automatically.

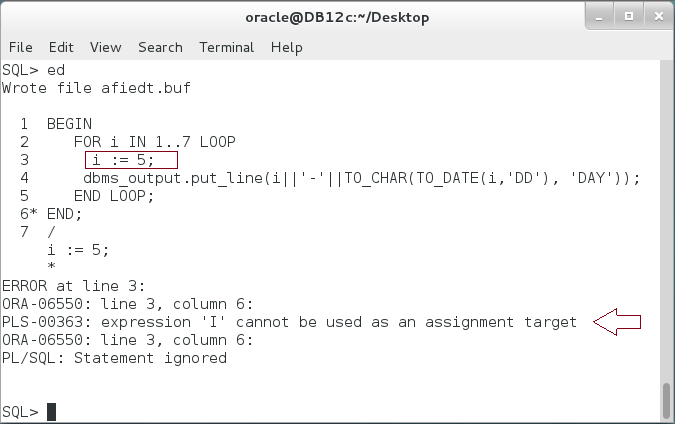
**Step 2:** Extend the previous block as the following:

|  |  |
| --- | --- |
| Line | Description |
| BEGIN |  |
| FOR i IN **REVERSE** 1..7 LOOP | FOR LOOP automatically runs implicit variable from 7 to 1 |
| dbms\_output.put\_line  (i||'-'||TO\_CHAR(TO\_DATE(i,'DD'), 'DAY')); |
| END LOOP; |  |
| END; |  |
| / |  |



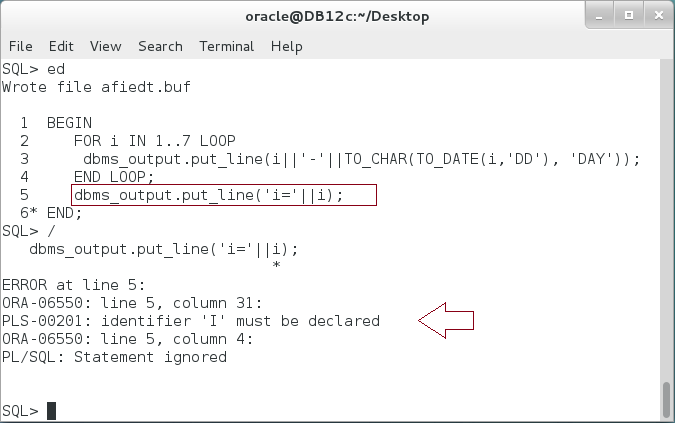
**Step 3:** Modify the previous block as the following:

|  |  |
| --- | --- |
| Line | Description |
| BEGIN |  |
| FOR i IN 1..7 LOOP | You can't change the implicit variable value. It is a **READ ONLY** variable. |
| **i := 5;** |
| dbms\_output.put\_line  (i||'-'||TO\_CHAR(TO\_DATE(i,'DD'), 'DAY')); |
| END LOOP; |  |
| END; |  |
| / |  |

****

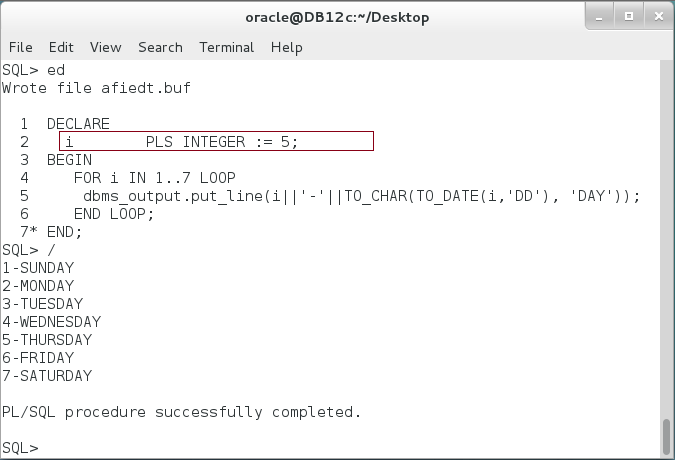
**Step 4:** The scope of implicit variable is the FOR LOOP body. Modify the previous block as the following:

|  |  |
| --- | --- |
| Line | Description |
| BEGIN |  |
| FOR i IN 1..7 LOOP |  |
| dbms\_output.put\_line  (i||'-'||TO\_CHAR(TO\_DATE(i,'DD'), 'DAY')); |  |
| END LOOP; |  |
| **dbms\_output.put\_line('i='||i);** | Try to access "i" variable outside the FOR LOOP |
| END; |  |
| / |  |

****

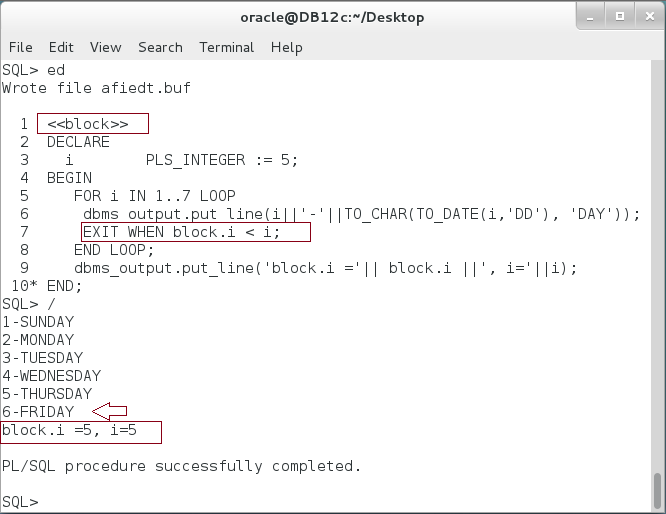
**Step 5:** If a variable with same name as the implicit variable is defined in a BLOCK level, then the LOOP will use the implicit variable .Modify the previous block as the following:

|  |  |
| --- | --- |
| Line | Description |
| **DECLARE** |  |
| **i PLS\_INTEGER := 5;** | Define "i" in a block level. |
| BEGIN |  |
| FOR i IN 1..7 LOOP | FOR LOOP uses implicit variable. |
| dbms\_output.put\_line  (i||'-'||TO\_CHAR(TO\_DATE(i,'DD'), 'DAY')); |
| END LOOP; |  |
| END; |  |
| / |  |

****

**Step 6:** You may access the block level variable using block labeling. Modify the previous block as the following:

|  |  |
| --- | --- |
| Line | Description |
| **<<block>>** |  |
| DECLARE |  |
| i PLS\_INTEGER := 5; |  |
| BEGIN |  |
| FOR i IN 1..7 LOOP | Implicit variable "i" reach 6. |
| dbms\_output.put\_line  (i||'-'||TO\_CHAR(TO\_DATE(i,'DD'), 'DAY')); |
| EXIT WHEN block.i < i; |  |
| END LOOP; |  |
| dbms\_output.put\_line  ('block.i='||block.i||', i='||i); | **Explain why block.i = i?** |
| END; |  |
| / |  |

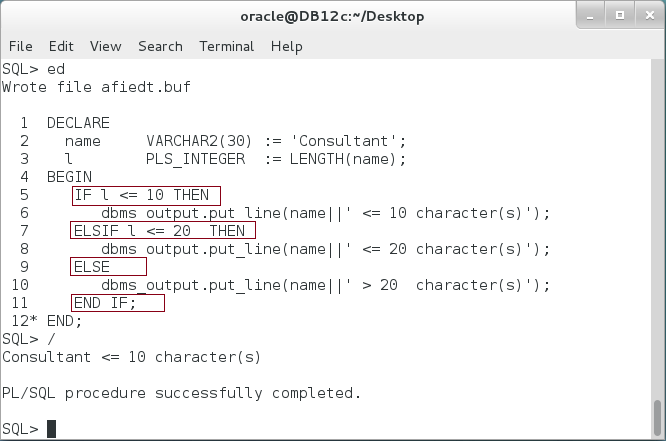
****

**Please remember:** Outside the loop, the implicit variable is **not defined.**

## IF Statements

**Step 1:** Execute the following block:

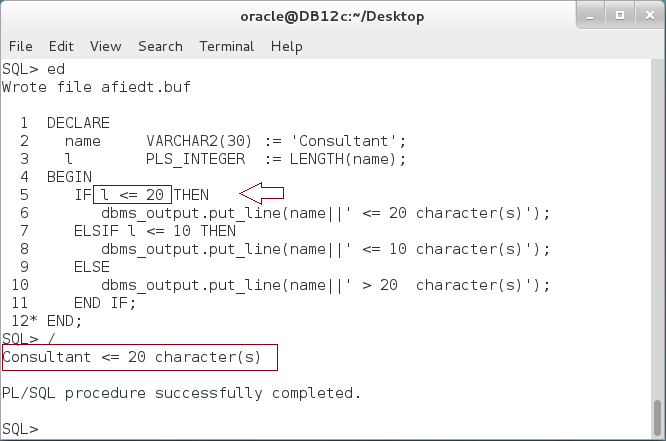
|  |  |
| --- | --- |
| Line | Description |
| DECLARE |  |
| name VARCHAR2(30) := 'Consultant'; |  |
| l PLS\_INTEGER := LENGTH(name); |  |
| BEGIN |  |
| **IF l <= 10 THEN** | Mandatory "**IF**" clause. |
| dbms\_output.put\_line(name||' <= 10 character(s)'); |
| **ELSIF l <= 20 THEN** | Optional "**ELSIF**" clause. |
| dbms\_output.put\_line(name||' <= 20 character(s)'); |
| **ELSE** | Optional "**ELSE**" clause. |
| dbms\_output.put\_line(name||' > 20 character(s)'); |
| **END IF;** | Mandatory "**END IF**" clause. |
| END; |  |
| / |  |



**Please note:** The "IF" conditions run from top to bottom. Be sure that the later condition is not included in the upper condition.

**Step 2:** Modify the previous block as the following:

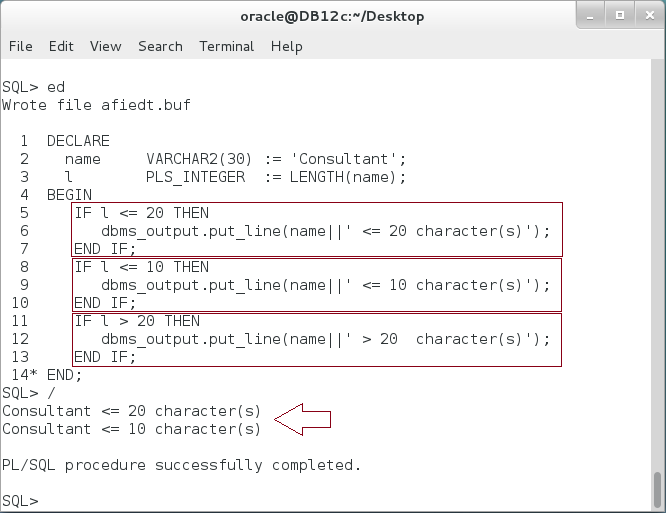
|  |  |
| --- | --- |
| Line | Description |
| DECLARE |  |
| name VARCHAR2(30) := 'Consultant'; |  |
| l PLS\_INTEGER := LENGTH(name); |  |
| BEGIN |  |
| **IF l <= 20 THEN** |  |
| dbms\_output.put\_line(name||' <= 20 character(s)'); |
| **ELSIF l <= 10 THEN** | Never reach this condition. **WHY?** |
| dbms\_output.put\_line(name||' <= 10 character(s)'); |
| ELSE |  |
| dbms\_output.put\_line(name||' > 20 character(s)'); |
| END IF; |  |
| END; |  |
| / |  |



**Please note:** In the previous block, the condition "l <= 10" is included in the preceding condition "l <=20". Since all names' length less than 10 is also less than 20, then the condition "l<=10" will never be satisfied.

**Step 3:** Modify the previous block as the following:

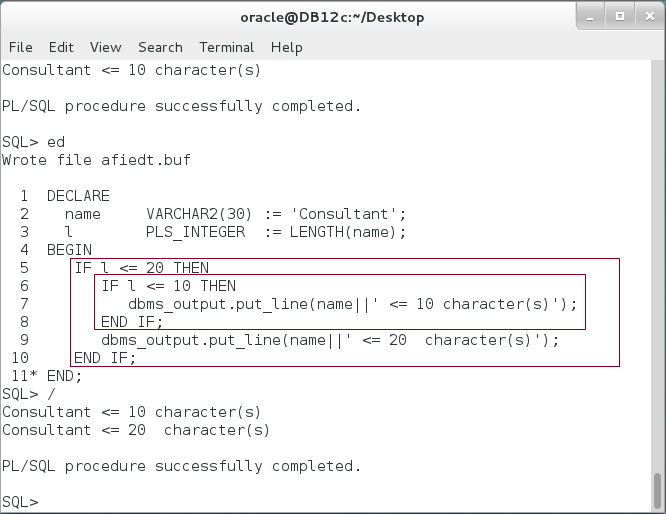
|  |  |
| --- | --- |
| Line | Description |
| DECLARE |  |
| name VARCHAR2(30) := 'Consultant'; |  |
| l PLS\_INTEGER := LENGTH(name); |  |
| BEGIN |  |
| **IF l <= 20 THEN** | One IF-ELSIF-ELSE statement is **not equal** to three different IF statements. |
| dbms\_output.put\_line(name||' <= 20 character(s)'); |
| **END IF;** |
| **ELSIF l <= 10 THEN** |
| dbms\_output.put\_line(name||' <= 10 character(s)'); |
| **END IF;** |
| **IF l > 20 THEN** |
| dbms\_output.put\_line(name||' > 20 character(s)'); |
| **END IF;** |  |
| END; |  |
| / |  |



**Please note:** In the previous block, the condition "l <= 10" and "l <=20" are both satisfied no matter the order of IF statements is.

**Step 3:** IF statements can be nested into other IF statements. Modify the previous block as the following:

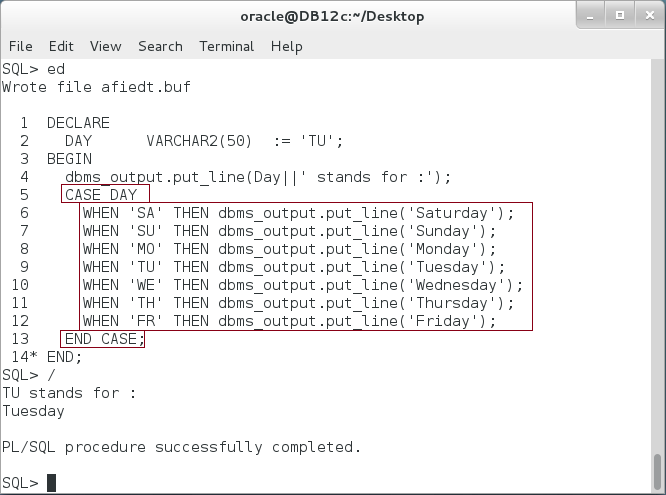
|  |  |
| --- | --- |
| Line | Description |
| DECLARE |  |
| name VARCHAR2(30) := 'Consultant'; |  |
| l PLS\_INTEGER := LENGTH(name); |  |
| BEGIN |  |
| **IF l <= 20 THEN** | Nested IF Statements. |
| **IF l <= 10 THEN** |
| dbms\_output.put\_line(name||' <= 10 character(s)'); |
| **END IF;** |
| dbms\_output.put\_line(name||' <= 20 character(s)'); |
| **END IF;** |  |
| END; |  |
| / |  |

****

## CASE Statements

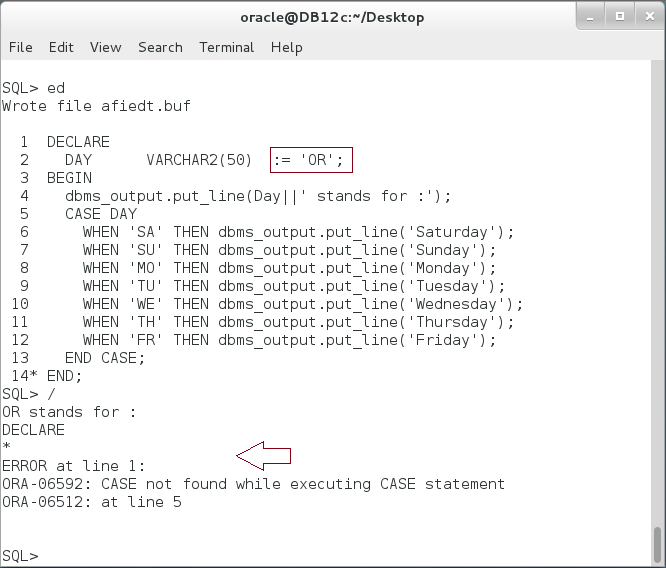
**Step 1:** You can see a CASE statement as a special case of IF statement. Execute the following block:

|  |  |
| --- | --- |
| Line | Description |
| DECLARE |  |
| DAY VARCHAR2(50) := 'TU'; |  |
| BEGIN |  |
| dbms\_output.put\_line(Day||' stands for :'); |  |
| **CASE** DAY |  |
| **WHEN** 'SA' **THEN** dbms\_output.put\_line('Saturday'); | Use WHEN THEN to check the value of "DAY". |
| **WHEN** 'SU' **THEN** dbms\_output.put\_line('Sunday'); |
| WHEN 'MO' THEN dbms\_output.put\_line('Monday'); |
| WHEN 'TU' THEN dbms\_output.put\_line('Tuesday'); |
| WHEN 'WE' THEN dbms\_output.put\_line('Wednesday'); |
| WHEN 'TH' THEN dbms\_output.put\_line('Thursday'); |
| WHEN 'FR' THEN dbms\_output.put\_line('Friday'); |
| **END CASE;** |  |
| END; |  |
| / |  |

****

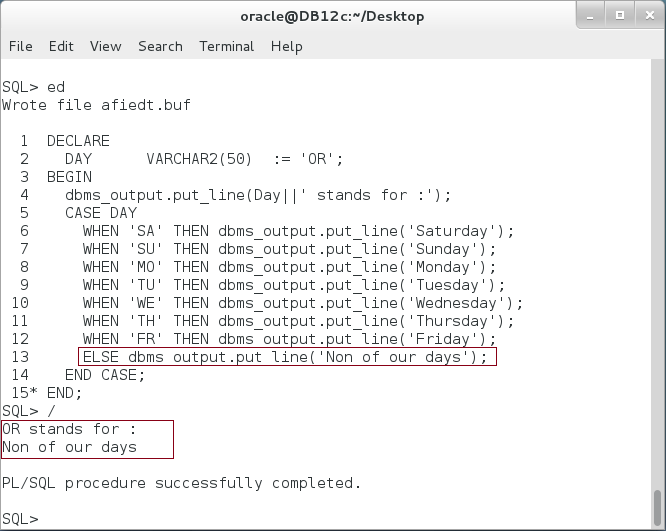
**Step 2:** The simple CASE statement runs the first statements for which value equals selector. Remaining conditions are not evaluated. If no value equals selector, Oracle raises exception. Execute the following block:

|  |  |
| --- | --- |
| Line | Description |
| DECLARE |  |
| DAY VARCHAR2(50) **:= 'OR';** |  |
| BEGIN |  |
| dbms\_output.put\_line(Day||' stands for :'); |  |
| CASE DAY |  |
| WHEN 'SA' THEN dbms\_output.put\_line('Saturday'); | Use WHEN THEN to check the value of "DAY". |
| WHEN 'SU' THEN dbms\_output.put\_line('Sunday'); |
| WHEN 'MO' THEN dbms\_output.put\_line('Monday'); |
| WHEN 'TU' THEN dbms\_output.put\_line('Tuesday'); |
| WHEN 'WE' THEN dbms\_output.put\_line('Wednesday'); |
| WHEN 'TH' THEN dbms\_output.put\_line('Thursday'); |
| WHEN 'FR' THEN dbms\_output.put\_line('Friday'); |
| END CASE; |  |
| END; |  |
| / |  |

****

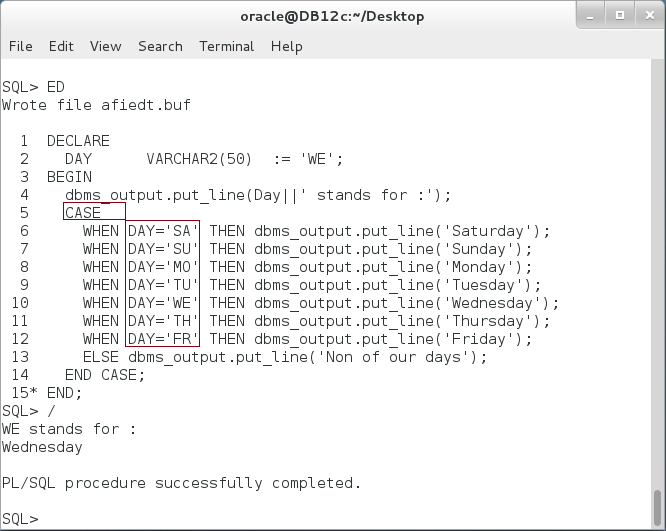
**Step 3:** To avoid this un-handled exception, use ELSE keyword at the end to handle all other values not listed above. Modify the previous block as shown below:

|  |  |
| --- | --- |
| Line | Description |
| DECLARE |  |
| DAY VARCHAR2(50) := 'OR'; |  |
| BEGIN |
| dbms\_output.put\_line(Day||' stands for :'); |
| CASE DAY |  |
| WHEN 'SA' THEN dbms\_output.put\_line('Saturday'); |  |
| WHEN 'SU' THEN dbms\_output.put\_line('Sunday'); |  |
| WHEN 'MO' THEN dbms\_output.put\_line('Monday'); |  |
| WHEN 'TU' THEN dbms\_output.put\_line('Tuesday'); |  |
| WHEN 'WE' THEN dbms\_output.put\_line('Wednesday'); |  |
| WHEN 'TH' THEN dbms\_output.put\_line('Thursday'); |  |
| WHEN 'FR' THEN dbms\_output.put\_line('Friday'); |
| **ELSE dbms\_output.put\_line('Non of our days');** |
| END CASE; |
| END; |
| / |  |

****

**Step 4:** Another way to implement CASE statement is to use "Searched CASE". . Modify the previous block as shown below:

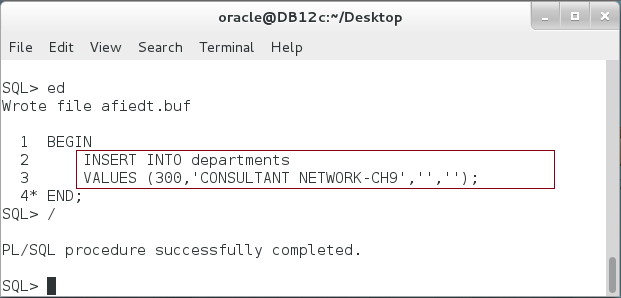
|  |  |
| --- | --- |
| Line | Description |
| DECLARE |  |
| DAY VARCHAR2(50) := 'WE'; |  |
| BEGIN |  |
| dbms\_output.put\_line(Day||' stands for :'); |  |
| **CASE** |  |
| WHEN **DAY='SA'** THEN dbms\_output.put\_line('Saturday'); |  |
| WHEN **DAY='SU'** THEN dbms\_output.put\_line('Sunday'); |  |
| WHEN DAY='MO' THEN dbms\_output.put\_line('Monday'); |  |
| WHEN DAY='TU' THEN dbms\_output.put\_line('Tuesday'); |  |
| WHEN DAY='WE' THEN dbms\_output.put\_line('Wednesday'); |  |
| WHEN DAY='TH' THEN dbms\_output.put\_line('Thursday'); |  |
| WHEN DAY='FR' THEN dbms\_output.put\_line('Friday'); |
| ELSE dbms\_output.put\_line('Non of our days'); |
| END CASE; |
| END; |
| / |  |

****

## TCL Statements

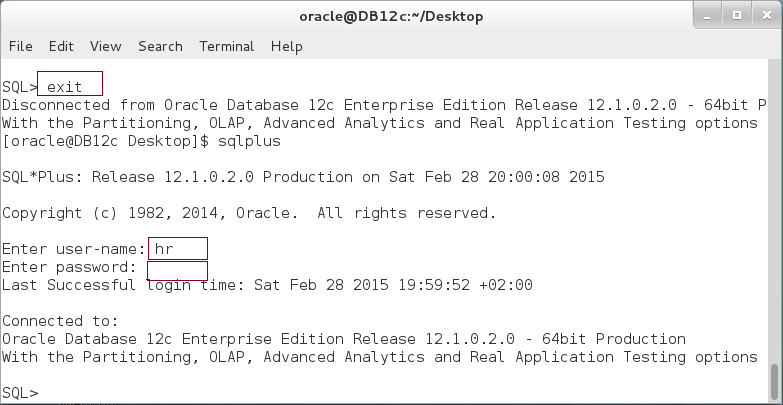
**Step 1:** PL/SQL block does not commit a transaction unless you explicitly specify. Execute a block that inserts a single record in DEPARTMENTS table.

|  |  |
| --- | --- |
| Line | Description |
| BEGIN |  |
| INSERT INTO departments |  |
| VALUES (301,'CONSULTANT NETWORK-CH9','',''); |  |
| END; |  |

****

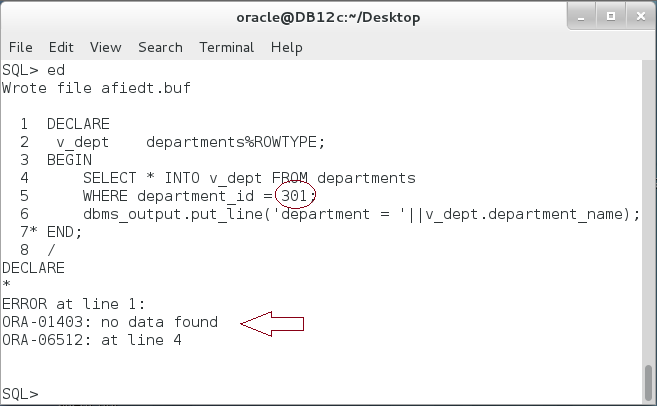
**Step 2:** Exist from SQL\*Plus session and reconnect again.

|  |  |
| --- | --- |
| Line | Description |
| exit | Exit the current session |
| sqlplus | Start another session |
| user-name | Connect to "**hr**" |
| password | Enter password "**oracle**" |

****

**Step 3:** Try to query the same query using the following block:

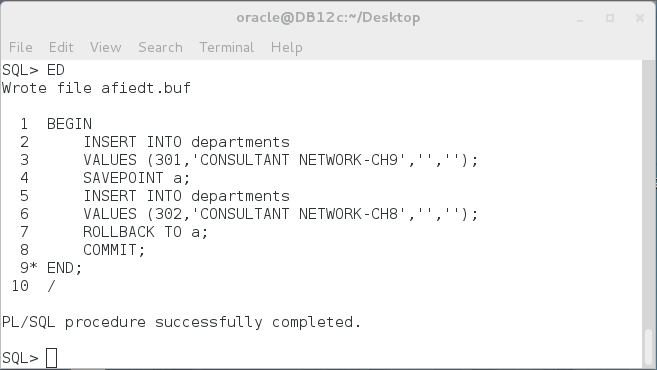
|  |  |
| --- | --- |
| Line | Description |
| DECLARE |  |
| v\_dept departments%ROWTYPE; |  |
| BEGIN |  |
| **SELECT \* INTO v\_dept FROM departments** |  |
| **WHERE department\_id = 301;** |  |
| dbms\_output.put\_line('dept='||v\_dept.department\_name); |  |
| END; |  |

****

**Please note:** The record is not exits. You must commit the transaction explicitly in PL/SQL block.

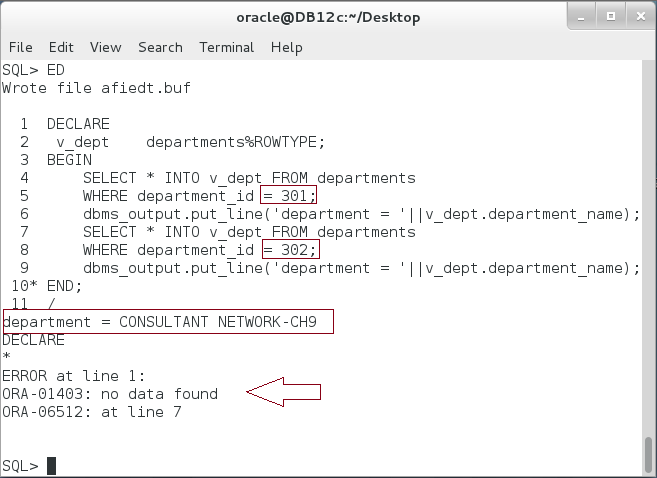
**Step 4:** Execute the following block.

|  |  |
| --- | --- |
| Line | Description |
| BEGIN |  |
| **INSERT INTO** departments |  |
| VALUES (301,'CONSULTANT NETWORK-CH9','',''); |  |
| **SAVEPOINT a;** |  |
| **INSERT INTO** departments |  |
| VALUES (302,'CONSULTANT NETWORK-CH8','',''); |  |
| **ROLLBACK TO a;** |  |
| **COMMIT;** |  |
| END; |  |

****

**Step 5:** Query the DEPARTMENTS table.

|  |  |
| --- | --- |
| Line | Description |
| DECLARE |  |
| v\_dept departments%ROWTYPE; |  |
| BEGIN |  |
| **SELECT \* INTO** v\_dept FROM departments |  |
| WHERE department\_id = 301; |  |
| dbms\_output.put\_line('dept= '||v\_dept.department\_name); |  |
| **SELECT \* INTO** v\_dept FROM departments |  |
| WHERE department\_id = 302; |  |
| dbms\_output.put\_line('dept= '||v\_dept.department\_name); |  |
| END; |  |

****

**Please note:** The first record exits while the second is not. **Explain the output?**

# SUMMARY

As any programming language, PL/SQL supports control statements. LOOP statements include the basic which requires more code but it more flexible. WHILE LOOP and FOR LOOP are very useful to shorten the code and focus more in the loop purpose. FOR LOOP automatically defines, increases/decreases, and checks the index of the loop. If the same variable name exists, the implicit variable takes the precedence. The conditional statements such as IF-ELSIF-ELSE and CASE are commonly used to carry the business logic. The CASE statement is a special case of IF statement. For all types of condition and loop statements, Oracle supports a nested coding: nested IF, nested LOOP. Oracle also supports a Transaction Control Language for PL/SQL. This may help you to rollback part of the transaction as shown in the last example.

After completing this lab exercise, you should be able to use all kind of conditional and loop statements in PL/SQL.

# REFERENCES

* http://docs.oracle.com/database/121/LNPLS/controlstatements.htm#LNPLS004

# INDEX

CASE statement 2, 25, 26, 28, 34

FOR LOOP 2, 3, 4, 5, 10, 15, 16, 18, 19, 34

IF statement 2, 25, 34

ROLLBACK 2, 4, 31

SAVEPOINT 2, 4, 31

TCL 2, 4, 5, 29

WHILE LOOP 2, 3, 5, 14, 15, 34