PL/SQL Syntax

Don't forget to write SET AUTOCOMMIT ON and SET SERVEROUTPUT ON

1) For Creating Anonymous Block

```
DECLARE
--declaration section
BEGIN
--execution section
[EXCEPTION]
--exception section
END;
```

2) To declare variable

```
DECLARE
        Variable_name datatype (size) NOT NULL | DEFAULT value [:=value];
BEGIN
END;
```

3) To declare %TYPE and %ROWTYPE variable

```
DECLARE
        Variable_name TABLENAME.COLUMNNAME%TYPE;
        Varibale_name TABLENAME%ROWTYPE;
BEGIN
END;
```

4) To display message on console

```
BEGIN
        DBMS_OUTPUT.PUT_LINE(message || Variable_name);
END;
```

5) IF THEN ELSE

```
BEGIN
        IF condition THEN
                SQL statement;
        ELSIF condition THEN
                SQL statement;
        ELSE
                SQL statement;
        END IF;
END;
```

6) SIMPLE LOOP

```
BEGIN
        LOOP
                SQL statement;
        EXIT [WHEN condition];
        END LOOP;
END;
```

7) WHILE LOOP

```
BEGIN
        WHILE condition LOOP
                SQL statement;
        END LOOP;
END;
```

8) FOR LOOP

```
BEGIN
        FOR loop_counter [REVERSE] low_bound..upper bound LOOP
                SQL statement;
        END LOOP;
END;
```

9) CASE STRUCTURE

```
BEGIN
        CASE variable
        WHEN expression||value THEN
                SQL statement;
        WHEN expression||value THEN
                SQL statement;
        ELSE
                SQL statement;
        END CASE;
END;
```

10) DYNAMIC SQL

```
BEGIN
        EXECUTE IMMEDIATE ‘DDL | DML STATEMENT’;
END;
```

11) Select INTO Statement

```
DECLARE
        Variablename datatype;
BEGIN
        SELECT COLUMNNAME INTO Variablename FROM TABLE WHERE <condition>;
        DBMS_OUTPUT.PUT_LINE(Variablename);
END;
```

12) Explicit cursor

```
DECLARE
        CURSOR NAMEOFCURSOR IS SELECT STATEMENT;
BEGIN
        OPEN MAMEOFCURSOR;
                FETCH NAMEOFCURSOR INTO Variable or RowtypeVariable;
        CLOSE NAMEOFCURSOR;
END;
```

13) <u>Implicit cursor</u>

```
DECLARE
BEGIN
        SELECT COLUMN INTO VARIABLE FROM TABLE…;
END;
```

14) <u>Parameterized cursor</u>

```
DECLARE
        CURSOR CURSORNAME (PAR1 DATATYPE) IS SELECT STATEMENT;
BEGIN
        OPEN CURSORNAME (VALUE);
        FETCH CURSORNAME INTO Variable or Rowtype;
        CLOSE CURSORNAME;
END;
```

15) <u>To check procedure / function code</u>

```
SELECT TEXT, LINE
FROM ALL_SOURCE
WHERE NAME=<NAMEOFSUBPROGRAM>;
```

16) <u>To delete procedure / function / trigger</u>

```
DROP PROCEDURE NAMEOFPROCEDURE;

DROP FUNCTION NAMEOFFUNCTION;

DROP TRIGGER NAMEOFTRIG;
```

For loop cursor

Prof. Kirtankumar Rathod
ADBMS PRACTICAL [IMCA0207 / IMSC0207]
Dept. of Computer Science
Indus University

For loop cursor

- There is an alternative way to handle cursors.
- It is called the **cursor FOR loop** because of the simplified syntax that is used.
- With a cursor FOR loop, the process of opening, fetching, and closing is handled implicitly.
- Use the cursor FOR loop if you need to FETCH and PROCESS every record from a cursor until you want to stop processing and exit the loop.

Syntax of For loop cursor:

```
DECLARE
        CURSOR <cursor_name> IS <SELECT statement>;
BEGIN
        FOR I IN <cursor_name>
        LOOP
        .
        .
        END LOOP;
END;
```

- In the above syntax, the declaration part contains the declaration of the cursor.
- The cursor is created for the 'SELECT' statement that is given in the cursor declaration.
- In execution part, the declared cursor is setup in the FOR loop and the loop variable 'I' will behave as cursor variable in this case.

Syntax of For loop cursor:

```
FOR record IN cursor_name
LOOP
    process_record_statements;
END LOOP;
```

Here, The record is the name of the index that the cursor FOR LOOP statement declares implicitly as a %ROWTYPE record variable of the type of the cursor. he cursor_name is the name of an explicit cursor that is not opened when the loop starts.

cursor_name contain select statement.

Example of For loop cursor....1

```
DECLARE
        CURSOR c_product  IS SELECT product_name, list_price
        FROM products ORDER BY list_price DESC;
BEGIN
        FOR r_product IN c_product
        LOOP
                dbms_output.put_line( r_product.product_name || ': Rs.' ||  r_product.list_price );
        END LOOP;
END;
```

in this example c_product is cursor and r_product is loop variable, so all the values of c_product will be display using r_product variable inside the loop.

Example of For loop cursor....2

```
BEGIN
          FOR r_product IN ( SELECT product_name, list_price
                                 FROM products ORDER BY list_price DESC )
          LOOP
                    dbms_output.put_line( r_product.product_name || ': Rs.' ||  r_product.list_price );
          END LOOP;
END;
```

in this example there is no cursor and r_product is loop variable, so all the values select statment will be display using r_product variable inside the loop.

Example of Cursor For Loop

```
SQL> DECLARE

CURSOR C1 IS SELECT STNAME

FROM TBL_STUD;

BEGIN

FOR I IN C1 LOOP

 DBMS_OUTPUT.PUT_LINE( I.STNAME );

 END LOOP;

 END;

 /
riya
kkkk
yash
raj shah
mahesh parekh

PL/SQL procedure successfully completed.

Commit complete.
```

In this example, C1 is cursor which will select all record of student name from tbl_stud. Here, I is for loop variable which will implicitly fetch record from cursor and display using column name with variable I like I.STNAME I dbms_output.put_line().

Example of Cursor For Loop

```
SQL> DECLARE
  2
  3  BEGIN
  4
  5  FOR I IN (SELECT STCITY FROM TBL_STUD) LOOP
  6
  7  DBMS_OUTPUT.PUT_LINE(I.STCITY);
  8
  9  END LOOP;
 10
 11   END;
 12
 13  /
delhi
ahmedabad
surat
Delhi
Ahmedabad

PL/SQL procedure successfully completed.

Commit complete.
SQL>
```

In this example, without cursor declared, directly select statement is used in the for loop, so it will become for loop cursor. Here, using I variable value of city column will be display in output.

Error Handling & Exception - PL/SQL

Prof. Kirtankumar Rathod
ADBMS PRACTICAL [IMCA0207 / IMSC0207]
Dept. of Computer Science
Indus University

Exception-Handling Concepts and Terminology

- In the PL/SQL language, errors of any kind are treated as exceptions—situations that should not occur—in your program.
  - An error generated by the system (such as “out of memory” or “duplicate value in index”).
  - An error caused by a user action.
  - A warning issued by the application to the user.
- The exception handler mechanism allows you to cleanly separate your error-processing code from your executable statements.

Exception-Handling

- When an error occurs in PL/SQL, whether it’s a system error or an application error, an exception is raised.
- The processing in the current PL/SQL block’s execution section halts, and control is transferred to the separate exception section of the current block,
- if one exists, to handle the exception. You cannot return to that block after you finish handling the exception. Instead, control is passed to the enclosing block, if any.

Syntax of Exception in PL/SQL block

```
DECLARE
----
BEGIN
------
EXCEPTION
        WHEN EXCEPTION_NAME THEN
                ERROR-PROCESSING STATEMENTS;
END;
```

EXPLANATION of Exception

- The exception-handling section is placed after the executable section of the block.

- An exception-handling section allows a program to execute to completion, instead of terminating prematurely.

- All error-processing code for a specific block is located in a single section.

Example : 1

```
SET SERVEROUTPUT ON;
    DECLARE
    v_num NUMBER := &v_num;
    BEGIN
    DBMS_OUTPUT.PUT_LINE ('Square root of '||v_num||' is '||SQRT(v_num));
    EXCEPTION
        WHEN VALUE_ERROR THEN
            DBMS_OUTPUT.PUT_LINE ('An error has occurred');
    END;
```

Explanation of example

```
SET SERVEROUTPUT ON;
        DECLARE
        v_num NUMBER := &v_num;
        BEGIN
        DBMS_OUTPUT.PUT_LINE ('Square root of '||v_num||' is '||SQRT(v_num));
        EXCEPTION -- exception keyword to define its section
                WHEN VALUE_ERROR THEN  -- VALUE_ERROR is in-built exception
                        DBMS_OUTPUT.PUT_LINE ('An error has occurred');
        END;
-- if user input number 4 it will give output 2 but, if we input -4 it will give error message written in the exception section.
```

The following list describes some commonly used predefined exceptions and how they are raised:

1. **NO_DATA_FOUND:** This exception is raised when a SELECT INTO statement that makes no calls to group functions, such as SUM or COUNT, does not return any rows.

2. **TOO_MANY_ROWS:** This exception is raised when a SELECT INTO statement returns more than one row.

3. **ZERO_DIVIDE:** This exception is raised when a division operation is performed in the program and a divisor is equal to 0.

4. **VALUE_ERROR:** This exception is raised when a conversion or size mismatch error occurs.

Example : 2

```
DECLARE
        v_student_id NUMBER := &sv_student_id;
        v_enrolled VARCHAR2(3) := 'NO';
BEGIN
        DBMS_OUTPUT.PUT_LINE ('Check if the student is enrolled');
        SELECT 'YES' INTO v_enrolled FROM enrollment WHERE student_id = v_student_id;
        DBMS_OUTPUT.PUT_LINE ('The student is enrolled into one course');
EXCEPTION
        WHEN NO_DATA_FOUND THEN
                DBMS_OUTPUT.PUT_LINE ('The student is not enrolled');
        WHEN TOO_MANY_ROWS THEN
                DBMS_OUTPUT.PUT_LINE ('The student is enrolled in too many courses');
END;
```

Explanation of example : 2

- This example contain two exception:
    - **NO_DATA_FOUND exception will raise if no record exist for the particular student id.**
    - **TOO_MANY_ROWS exception will raise if more than one record exist for the particular student id.**

Use of OTHERS in exception section...

- OTHERS exception will raise for all pre-defined ORACLE errors.

```
DECLARE
        v_instructor_id NUMBER := &sv_instructor_id;
        v_instructor_name VARCHAR2(50);
BEGIN
        SELECT first_name||' '||last_name INTO v_instructor_name FROM instructor
        WHERE instructor_id = v_instructor_id;
                DBMS_OUTPUT.PUT_LINE ('Instructor name is '||v_instructor_name);
        EXCEPTION
                WHEN OTHERS THEN
                        DBMS_OUTPUT.PUT_LINE ('An error has occurred');
END;
```

Example of Exception: 1 [TOO_MANY_ROWS]

```
SQL> DECLARE
  2
  3  V_NAME TBL_STUD.STNAME%TYPE;
  4  V_CITY TBL_STUD.STCITY%TYPE;
  5
  6  BEGIN
  7
  8  SELECT STNAME,STCITY INTO V_NAME,V_CITY FROM TBL_STUD;
  9
 10  DBMS_OUTPUT.PUT_LINE(V_NAME||'  '||V_CITY);
 11
 12  EXCEPTION
 13
 14  WHEN TOO_MANY_ROWS THEN
 15
 16  DBMS_OUTPUT.PUT_LINE('CURSOR C1 IS HAVING MORE THAN 1 RECORDS...');
 17
 18  END;
 19
 20  /
CURSOR C1 IS HAVING MORE THAN 1 RECORDS...

PL/SQL procedure successfully completed.

Commit complete.
```

In this example, select statement will have more than 1 record so all records cannot be stored in two variables v_name and v_city, so, exception will be raise and user defined error message will be display on the screen...

Example of Exception: 2 [USER DEFINED EXCEPTION]

```
SQL> DECLARE
  2
  3  V_NUM1 NUMBER;
  4  V_NUM2 NUMBER;
  5  V_NUM3 NUMBER;
  6
  7  MY_EXCEPTION EXCEPTION;
  8
  9  BEGIN
 10
 11  V_NUM1:=&V_NUM1;
 12
 13  V_NUM2:=&V_NUM2;
 14
 15  V_NUM3:=V_NUM1*V_NUM2;
 16
 17  IF V_NUM3 = 0 THEN
 18
 19  RAISE MY_EXCEPTION;
 20
 21  ELSE
 22
 23  DBMS_OUTPUT.PUT_LINE('ANSWER IS '||V_NUM3);
 24
 25  END IF;
 26
 27  EXCEPTION
 28
 29  WHEN MY_EXCEPTION THEN
 30
 31  DBMS_OUTPUT.PUT_LINE('PLEASE ENTER VALUE GREATER THAN 0');
 32
 33  END;
 34
 35  /
Enter value for v_num1: 5
old  11: V_NUM1:=&V_NUM1;
new  11: V_NUM1:=5;
Enter value for v_num2: 5
old  13: V_NUM2:=&V_NUM2;
new  13: V_NUM2:=5;
ANSWER IS 25

PL/SQL procedure successfully completed.

Commit complete.
SQL> /
Enter value for v_num1: 7
old  11: V_NUM1:=&V_NUM1;
new  11: V_NUM1:=7;
Enter value for v_num2: 0
old  13: V_NUM2:=&V_NUM2;
new  13: V_NUM2:=0;
PLEASE ENTER VALUE GREATER THAN 0

PL/SQL procedure successfully completed.

Commit complete.
```

In this example, my_exception is user defined exception, so whenever user enter value equal to 0 than answer of multiplication will be 0 and exception will be raised and error message will be display accordingly.

Example of Exception: 3 [ZERO_DIVIDE EXCEPTION]

```
--------------------------------------------------------
SQL> DECLARE
  2  V_NUM1 NUMBER;
  3  V_NUM2 NUMBER;
  4  V_NUM3 NUMBER;
  5
  6  BEGIN
  7
  8  V_NUM1:=&V_NUM1;
  9  V_NUM2:=&V_NUM2;
 10  V_NUM3:=V_NUM1/V_NUM2;
 11
 12  DBMS_OUTPUT.PUT_LINE('ANSWER IS '||V_NUM3);
 13
 14  EXCEPTION
 15
 16  WHEN ZERO_DIVIDE THEN
 17  DBMS_OUTPUT.PUT_LINE('SORRRYYYYY.....DIVISION IS NOT POSSIBLE....');
 18  END;
 19
 20  /
Enter value for v_num1: 5
old   8: V_NUM1:=&V_NUM1;
new   8: V_NUM1:=5;
Enter value for v_num2: 5
old   9: V_NUM2:=&V_NUM2;
new   9: V_NUM2:=5;
ANSWER IS 1
PL/SQL procedure successfully completed.
Commit complete.
SQL> /
Enter value for v_num1: 8
old   8: V_NUM1:=&V_NUM1;
new   8: V_NUM1:=8;
Enter value for v_num2: 0
old   9: V_NUM2:=&V_NUM2;
new   9: V_NUM2:=0;
SORRRYYYYY.....DIVISION IS NOT POSSIBLE....
```

```
PL/SQL procedure successfully completed.

Commit complete.
SQL> /
Enter value for v_num1: 25
old   8: V_NUM1:=&V_NUM1;
new   8: V_NUM1:=25;
Enter value for v_num2: 5
old   9: V_NUM2:=&V_NUM2;
new   9: V_NUM2:=5;
ANSWER IS 5
PL/SQL procedure successfully completed.
Commit complete.
SQL>
```

In this example, built-in exception is called whenever user input 0 in the second variable, so division will not be possible and zero-divide will be raised and error message will be display on the screen.
