PL/SQL: PROCEDURES & FUNCTIONS

Procedural languages are linear. They begin at the beginning, and end at the end. Each statement must wait for the preceding statement to complete before it can run. PL/SQL is a mix of procedural and object-oriented languages (Orcale 8i onwards).

PL/SQL

- has a tight integration with the database
- supports advanced language concepts and capabilities
- ensures a structured approach to programming, wherein for every BEGIN, there is an END and for every IF, there is an END IF.
- Virtually all SQL capabilities are possible with PL/SQL.

PL/SQL is provided by Oracle, exclusively for Oracle and is grown from Ada. It is a proprietary language not available outside the Oracle Database. It is a third-generation language (3GL) that provides programming constructs similar to other 3GL languages, including variable declarations, loops, error handling, etc. Historically, PL/SQL was procedural only.

SQL provides no ability to

- Loop through records, manipulating them one at a time.
- Keep code secure by offering encryption, and storing code permanently on the server rather than the client.
- Handle exceptions.
- Work with variables, parameters, collections, records, arrays, objects, cursors, exceptions, BFILEs, etc.

Anonymous Blocks

Anonymous blocks of code are not stored, and not named. They are executed in-session and cannot be called from another session. To execute the same code again, the anonymous block must be saved to an OS file and run, typed it in again, or included in a program that executes the block when needed.
Anonymous blocks are perfect for scripting, or activities that you do not wish to repeat frequently. Anonymous blocks are used extensively.

PL/SQL code is grouped into structures called blocks. When a stored procedure or package is created, the block of PL/SQL code is given a name. If the block of PL/SQL code is not given a name, it is called an anonymous block. The block begins with DECLARE or BEGIN and is not stored anywhere once executed. PL/SQL block contains three sections:

Section	Description
Declarations	Defines and initializes the variables and cursors used in the block.
Executable Commands	Uses flow-control commands (such as if commands and loops) to execute the commands and assign values to the declared variables .
Exception Handling	Provides customized handling of error conditions .

Within the Declarations section, the variables and cursors used by the block are defined. It starts with DECLARE keyword and ends when the Executable Commands section starts (indicated by BEGIN keyword). The Executable Commands section is followed by the Exception Handling section (indicated by EXCEPTION keyword signals the start of the Exception Handling section. The PL/SQL block is terminated by the END keyword.

PL/SQL Block Structure

```
[DECLARE
      < declaration section > ]
 BEGIN
      < executable commands >
[EXCEPTION
      < exception handling  > ]
 END;
```

To view the output of a PL/SQL block, the environment variable SERVEROUTPUT should be enabled.

```
SET SERVEROUTPUT ON
```

To display the system time ahead by 3 hours, the block can be coded as -

```
DECLARE
      V_DATE TIMESTAMP;
BEGIN
      SELECT SYSTIMESTAMP + 3/24
            INTO V_DATE FROM DUAL;
      DBMS_OUTPUT.PUT_LINE('Clock After 3 Hours:'' || V_DATE );
END;
/
```

To compute and print volume of a sphere, the block can be coded ass

```
DECLARE
      PI CONSTANT NUMBER(7,5):= 3.14159;
      V_RAD INTEGER(3);
      V_VOL NUMBER(9,2);
BEGIN
      V_RAD := 7;
      V_VOL := PI * POWER(V_RAD,3) * (4/3);
      DBMS_OUTPUT.PUT_LINE('Volume of Sphere with Radius ' || V_RAD
            || ' is ' || V_VOL || ' cu. units.');
END;
/
```

Whereas the minimum structure for a PL/SQL block is a BEGIN and an END with at least one executable command in between.

```
BEGIN
      NULL;
END;
/
```

CONDITIONALS AND LOOPING STRUCTURES

IF Statement

An IF statement has two forms: IF-THEN and IF-THEN-ELSE. The IF-THEN statement is the most basic kind of a conditional control; it has the following structure:

```
IF condition THEN
      statement_1;
      ...
      statement_n;
END IF;
```

IF-ELSE Statement

An IF-THEN-ELSE statement enables you to specify two groups of statements. One group of statements is executed when the condition evaluates to TRUE. Another group of statements is executed when the condition evaluates to FALSE. This is indicated as follows: :

```
IF condition THEN
      statement_1;
ELSE
      statement_2
END IF;
statement_3;
```

To check whether the user entered date falls on the weekend (the saturday or sunday).

```
DECLARE
      V_DATE DATE := TO_DATE('&USERDATE', 'DD-MON-YYYY');
      V_DAY  VARCHAR2(15);
BEGIN
      V_DAY := RTRIM(TO_CHAR(V_DATE, 'DAY'));
      -- IF V_DAY LIKE 'S%' THEN
      IF V_DAY IN ('SATURDAY', 'SUNDAY') THEN
            DBMS_OUTPUT.PUT_LINE (V_DATE||' falls on weekend');
      END IF;
      DBMS_OUTPUT.PUT_LINE ('DONE...');
END;
```

Remove the RTRIM function from the assignment statement for v_day as follows:

```
V_DAY := TO_CHAR(V_DATE, 'DAY');
```

and reexceute the script. Observe the changes.

ELSIF Statement

If none of the specified conditions yields TRUE, control is passed to the ELSE part of the ELSIF construct. An ELSIF statement can contain any number of ELSIF clauses.

```
IF condition_1 THEN
      statement_1;
ELSIF condition_2 THEN
      statement_2
ELSIF condition_3 THEN
      statement_3
      ...
ELSE
      statement_n
END IF;
```

To check whethe

```
DECLARE
      V_NUM NUMBER := &NUMIN;
BEGIN
      IF V_NUM < 0 THEN
            DBMS_OUTPUT.PUT_LINE('NEGATIVE NUMBER : ' || V_NUM);
      ELSIF V_NUM = 0 THEN
            DBMS_OUTPUT.PUT_LINE('NUMBER IS ZERO : ' || V_NUM);
      ELSE
            DBMS_OUTPUT.PUT_LINE('POSITIVE NUMBER : ' || V_NUM);
      END IF;
END;
```

[Use of NESTED IF]

PL/SQL block to converts the value of a temperature from one system to another. If the temperature is supplied in Fahrenheit, it is converted to Celsius, and vice versa.

```
DECLARE
      V_TEMP_IN   NUMBER := &TEMP_IN;
      V_TEMP_OU T  NUMBER;                -- := 0
      V_SCALE_IN  CHAR(1) := &SCALE_IN;
      V_SCALE_OUT CHAR(1);                -- := 'X'
BEGIN
      IF V_SCALE_IN != 'C' AND V_SCALE_IN != 'F' THEN
            DBMS_OUTPUT.PUT_LINE('.. INVALID SCALE ..');
      ELSE
            IF V_SCALE_IN = 'C' THEN
                  V_TEMP_OUT := ((9 * V_TEMP_IN)/5 ) + 32;
                  V_SCALE_OUT := 'F';
            ELSE
                  V_TEMP_OUT := ((V_TEMP_IN - 32)*5 ) / 9;
                  V_SCALE_OUT := 'C';
            END IF;
            DBMS_OUTPUT.PUT_LINE ('NEW TEMPERTAURE SCALE: '||V_SCALE_OUT);
            DBMS_OUTPUT.PUT_LINE ('NEW TEMPERATURE VALUE: '||V_TEMP_OUT);
      END IF;
END;
```

[Use of CASE Statement]

To print the day for user entered date.

```
DECLARE
      V_DATE DATE := TO_DATE('&USERDATE', 'DD-MON-YYYY');
      V_DAY CHAR(1);
BEGIN
      V_DAY := TO_CHAR(V_DATE, 'D');
      CASE V_DAY
            WHEN '2' THEN
                  DBMS_OUTPUT.PUT_LINE (V_DATE||': MONDAY');
            WHEN '3' THEN
                  DBMS_OUTPUT.PUT_LINE (V_DATE||': TUESDAY');
            WHEN '4' THEN
                  DBMS_OUTPUT.PUT_LINE (V_DATE||': WEDNESDAY');
            WHEN '5' THEN
                  DBMS_OUTPUT.PUT_LINE (V_DATE||': THURSDAY');
            WHEN '6' THEN
                  DBMS_OUTPUT.PUT_LINE (V_DATE||': FRIDAY');
            ELSE
                  DBMS_OUTPUT.PUT_LINE (V_DATE||': WEEKEND');
      END CASE;
END;
```

SIMPLE LOOP

A simple loop is the most basic kind of loop.

```
LOOP
      statement_1;
      statement_2;
      ...
      statement_n;
END LOOP;
```

Every time the loop iterates, a sequence of statements is executed, and then control is passed back to the top of the loop. The sequence of statements is executed an infinite number of times, because no statement specifies when the loop must terminate. Hence, a simple loop is called an **infinite loop** because there is no means to exit the loop. A properly constructed loop needs an exit condition that determines when the loop is complete. This exit condition has two forms: EXIT and EXIT WHEN. The reserved word LOOP marks the beginning of the simple loop.

EXIT Statement

The EXIT statement causes a loop to terminate when the EXIT condition evaluates to TRUE. The EXIT condition is evaluated with the help of an IF statement. When the EXIT condition is evaluated to TRUE, control is passed to the first executable statement after the END LOOP statement. This is indicated by the following:

```
LOOP
      statement_1;
      statement_2;
      IF condition THEN
            EXIT;
      END IF;
END LOOP;
statement_3;
```

The EXIT statement is valid only when placed inside a loop. When placed outside a loop, it causes a syntax error. To avoid this error, use the RETURN statement to terminate a PL/SQL block before its normal end is reached:

```
BEGIN
      DBMS_OUTPUT.PUT_LINE ('Termnated');
      RETURN;
      DBMS_OUTPUT.PUT_LINE ('Not Executed');
END;
```

EXIT WHEN Statement

The EXIT WHEN statement causes a loop to terminate only if the EXIT WHEN condition evaluates to TRUE. Control is then passed to the first executable statement after the END LOOP statement.

```
LOOP
      statement_1;
      statement_2;
      EXIT WHEN condition;
END LOOP;
statement_3;
```

The decision about loop termination is made inside the body of the loop, and the body of the loop, or a part of it, is always executed at least once. However, the number of iterations of the loop depends on the evaluation of the EXIT condition and is not known until the loop completes.

When the EXIT statement is used without an EXIT condition, the simple loop executes only once.

```
DECLARE
      V_KNT NUMBER := 0;
BEGIN
      LOOP
            DBMS_OUTPUT.PUT_LINE ('Count = '||V_KNT);
            EXIT;
      END LOOP;
END;
```

WHILE Loop

```
WHILE condition LOOP
      statement_1;
      statement_2;
      ...
      statement_n;
END LOOP;
```

The test condition is evaluated prior to each iteration of the loop.

```
DECLARE
      V_KNT NUMBER := &NUM; -- initial value below 10
BEGIN
      WHILE V_KNT < 10 LOOP
            DBMS_OUTPUT.PUT_LINE ('V_COUNTER = '||V_COUNTER);
            V_KNT := V_KNT - 1;
      END LOOP;
END;
```

The EXIT and EXIT WHEN statements can be used inside the body of a WHILE loop. If the EXIT condition evaluates to TRUE before the test condition evaluates to FALSE, the loop is terminated prematurely. If the test condition yields FALSE before the EXIT condition yields TRUE, there is no premature termination of the loop.

Numeric FOR Loops

A numeric FOR loop is called numeric because it requires an integer as its terminating value.

```
FOR loop_counter IN [REVERSE] lo_limit .. hi_limit LOOP
      statement_1;
      statement_2;
      ...
      statement_n;
END LOOP;
```

The reserved word FOR marks the beginning of a FOR loop construct. The variable loop_counter is an implicitly defined index variable. There is no need to define the loop counter in the declaration section of the PL/SQL block. This variable is defined by the loop construct. lo_limit and hi_limit are two integer numbers or expressions that evaluate to integer values at runtime, and the double dot (..) serves as the range operator. lo_limit and hi_limit define the number of iterations for the loop, and their values are evaluated once, for the first iteration of the loop.

The reserved word IN or IN REVERSE must be present when the loop is defined. If the REVERSE keyword is used, the loop counter iterates from the upper limit to the lower limit. However, the syntax for the limit specification does not change. The lower limit is always referenced first.

```
BEGIN
      FOR V_KNT IN 1 .. 5 LOOP
            DBMS_OUTPUT.PUT_LINE ('Count = '||V_KNT);
      END LOOP;
END;
```

The loop counter is implicitly defined and incremented when a numeric FOR loop is used. As a result, it cannot be referenced outside the body of the FOR loop. As soon as the loop completes, the

loop counter ceases to exist. Therefore, the statement,

```
V_KNT :=  V_KNT + 1;
```

results in an error.

The EXIT and EXIT WHEN statements covered in the previous labs can be used inside the body of a numeric FOR loop as well. If the EXIT condition evaluates to TRUE before the loop counter reaches its terminal value, the FOR loop is terminated prematurely. If the loop counter reaches its terminal value before the EXIT condition yields TRUE, the FOR loop doesn't terminate prematurely.

```
BEGIN
      FOR V_KNT IN 1 .. 7 LOOP
            DBMS_OUTPUT.PUT_LINE ('Count = '||V_KNT);
            EXIT WHEN V_KNT = 4;
      END LOOP;
END;
```

EXCEPTION HANDLING

Two types of errors that can be found in a program: compilation errors and runtime errors. A special section in a PL/SQL block handles runtime errors, called the exception-handling section, and in it, runtime errors are called exceptions. The exception-handling section allows programmers to specify what actions should be taken when a specific exception occurs. PL/SQL has two types of exceptions: built-in and user-defined.

The exception- handling section has the following structure:

```
EXCEPTION
      WHEN exception_name THEN
            error_processing_statements;
```

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

Let us catch the divide by zero error.

```
DECLARE
      V_NUM1 INTEGER := &NO1;
      V_NUM2 INTEGER := &NO2;
      V_OUT  NUMBER;
      BEGIN
            V_OUT := V_NUM1 / V_NUM2;
            DBMS_OUTPUT.PUT_LINE ('Result: '||V_OUT);
      EXCEPTION
            WHEN ZERO_DIVIDE THEN
                  DBMS_OUTPUT.PUT_LINE ('DIVIDE BY ZERO ERROR..');
END;
```

ZERO_DIVIDE is a system defined exception.

Advantages of Exception Handler

1. An exception-handling section allows a program to execute to completion, instead of terminating prematurely.
2. The exception-handling section isolates error-handling routines. All error-processing code for a specific block is located in a single section. As a result, the program's logic is easier to follow and understand.
3. Adding an exception-handling section enables event-driven processing of errors.

Calculate the value of the square root of a number and display it on the screen.

```
DECLARE
      V_NUM NUMBER := &NO;
BEGIN
      DBMS_OUTPUT.PUT_LINE ('SQRT('||V_NUM|| '): '||SQRT(V_NUM));
EXCEPTION
      WHEN VALUE_ERROR THEN
            DBMS_OUTPUT.PUT_LINE ('AN ERROR HAS OCCURRED');
END;
```

When an error occurs that raises a built-in exception, the exception is said to be raised implicitly. In other words, if a program breaks an Oracle rule, control is passed to the exception- handling section of the block. At this point, the error-processing statements are executed. It is important to realize that after the exception-handling section of the block has executed, the block terminates. Control does not return to the executable section of the block.

```
DECLARE
      V_NAME VARCHAR2(35);
BEGIN
      SELECT FNAME||' '||LNAME INTO V_NAME
            FROM STUDENT
            WHERE ROLL = 4001;
      DBMS_OUTPUT.PUT_LINE ('4001: '|| V_NAME);
EXCEPTION
      WHEN NO_DATA_FOUND THEN
            DBMS_OUTPUT.PUT_LINE ('NO Such Student...');
END;
```

USER-DEFINED EXCEPTIONS

A user-defined exception is defined by the programmer. As a result, before the exception can be used, it must be declared. A user-defined exception is declared in the declaration section of a PL/SQL block, catched in executable section, and handled in exception-handling section.

```
DECLARE
      exception_name EXCEPTION;
BEGIN
      ...

      IF condition THEN
            RAISE exception_name;
      ELSE
            ...
      END IF;

EXCEPTION
      WHEN exception_name THEN
            error_processing_statements;
END;
```

For example,

```
DECLARE
      V_ROLL STUDENT.ROLL%TYPE := &ROLL_NO;
      V_KOUNT NUMBER;
      E_INVALID_ID EXCEPTION;
BEGIN
      IF V_ROLL < 0 THEN
            RAISE E_INVALID_ID;
      ELSE
            SELECT COUNT(*) INTO V_KOUNT
                  FROM STUDENT
                  WHERE ROLL = V_ROLL;
            DBMS_OUTPUT.PUT_LINE ('Student Exists..') ;
      END IF;
      DBMS_OUTPUT.PUT_LINE ('No Error...');
EXCEPTION
      WHEN E_INVALID_ID THEN
            DBMS_OUTPUT.PUT_LINE ('Roll Number cannot be Negative.');
END;
```

the RAISE statement should be used in conjunction with an IF statement. Otherwise, control of the execution is transferred to the exception-handling section of the block for every execution.