Basics of PL/SQL

- PL/SQL - Introduction and advantages
  Understanding
- PL/SQL Block structure
  Fundamentals of PL/SQL Language - data types (BOOLEAN, CHAR, NUMBER, DATE, VARCHAR2), variables, constants and expressions (CASE expression)
- Operators
- Conditional statement – IF and CASE statements
- Controlling loop iterations – LOOP, EXIT, EXITWHEN, WHILE, FOR
- Sequential control statement – GOTO and NULL

INTRODUCTION TO PL / SQL

A PL / SQL are an Oracle's procedural language. It is the superset of SQL. You can use PL / SQL to implement your business rules by creating stored functions, procedures, and triggers or you can add programming logic to the SQL commands. PL / SQL code is grouped into structures called as block.

In PL / SQL there is an exception handling section which provides mechanism for handling the errors. When the PL / SQL block encounters any error the control of PL / SQL block shifts to the exception handling section. The PL / SQL provide system defined exception as well as it allows you to add your own exception (user defined exception).

If a SELECT statement is used in PL / SQL program only one row can be retriever. In Oracle the cursors are used to retrieve total number f rows by the SELECT statement.

The triggers define an action the database should take when some conditions are arise or satisfied. Triggers are executed by the database when commands like INSERT, UPDATE, and DELETE are fired on the table.

The PL / SQL also support procedures and functions. The groups of procedures, functions, variables, and other PL / SQL commands are called a package.

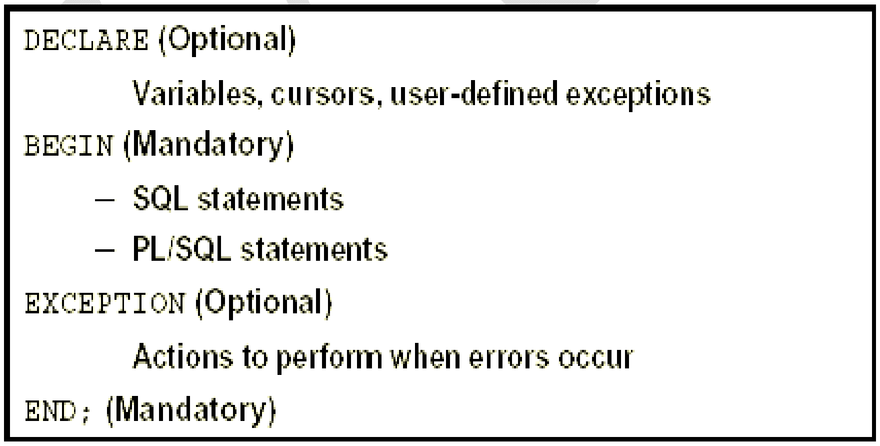
LIMITATIONS OF SQL

- SQL does not have any procedural capabilities. For Example - SQL does not provide the programming techniques such as conditional checking, looping and branching. That is critical for the data storage.
- An statements are passed to the oracle engine one by one at a time-
- Each time an SQL statement executed a call is made to the oracle engine resources.
- This adds to the traffic on the network by this processing speed decreases, especially in multi-user environment.
- While processing an oracle statement if an error occurs the oracle engine displays its own error message.
- SQL has no facility for program handling and error handling.

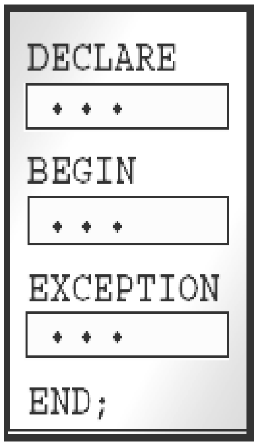
ADVANTAGES OF PL / SQL OVER SQL

- PL/SQL is a development tool that not only supports the SQL data manipulation but also providing facilities like conditional checking, looping and branching.
- PL/SQL sends an entire block of statement to the oracle engine at one time-
  1. The communication between program block and the oracle engine reduces the processing time.
  2. This reduces the network traffic.
  3. PL/SQL also permits dealing errors as required and it displays user-friendly messages when errors are encounter.
- PL/SQL allows declaration and use of variables in block of code.
  1. This variables can be used to store inter mediate result of a query, or it calculates the values and insert them into an oracle table later.
  2. PL/SQL variables can be used anywhere either in SQL or in PL/SQL.
- Through PL/SQL all the calculations can be done quickly and efficiently without the use of oracle engine. This improves the transaction performance.
- The applications written in PL/SQL are portable to any computer hardware and OS where oracle is operational.

PL / SQL BLOCK Structure: -



```
DECLARE (Optional)
        Variables, cursors, user-defined exceptions
BEGIN (Mandatory)
      – SQL statements
      – PL/SQL statements
EXCEPTION (Optional)
        Actions to perform when errors occur
END; (Mandatory)
```



PL / SQL are a block structured language. A program can be divided into logical blocks. The block structure gives modularity to a PL / SQL program and each object within a block has a scope.

A PL / SQL block consists of four sections.

A declarative section, executable section, exception - handling section, and end section.

Each of these sections explained below: -

- **DECLARE SECTION:**

  The code block starts with a declarative section in which memory variables and other Oracle object's can be declared and if require it will be initialized. Once it is declared they can be used in SQL statement for data manipulation.

- **BEGIN SECTION (Executable section):**

  It contains the set of SQL and PL / SQL statement. Actual data manipulation, retrieval, looping, and branching statements are specified in this section.

- **EXCEPTION SECTION:**

  This section deals with handling of errors that arises during execution of data manipulation statements, which makes up the PL / SQL code block. The errors can be arising due to syntax, logic, or violation of rules.

- **END SECTION:**

  This section specifies that end of PL / SQL block.

PL/SQL DATATYPES:

Both PL/SQL and Oracle have their foundations in SQL. Most PL/SQL datatypes are similar to Oracle's data dictionary. Hence there is a very easy integration of PL/SQL code with the Oracle's engine.

The default data types that can be declared in PL/SQL are:-

- Number: - It is used for storing numeric data
- Char: - It is used for storing character data
- Boolean: - It is used for storing logical values that is True or False
- Varchar and Varchar2: - It is used for storing string values of upto 255 characters
- Date: - It is used to store date
- Long: -To store variable length character strings also used to store arrays of binary data in ASCII format.

Variables and Constants:

A variable name must begin with character and can be followed by a maximum of 29 other characters.
Reserved Keywords cannot be used as variable names.
A space cannot be used in a variable name.

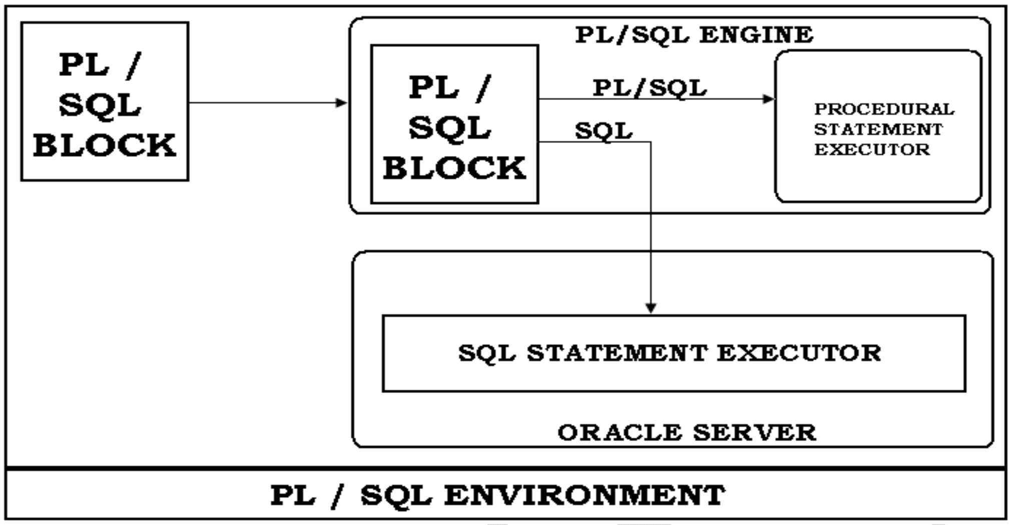
HOW TO DECLARE A VARIABLE:

DECLARE Identifier name [CONSTANT] DATATYPE [NOT NULL]

[:= / DEFAULT EXPRESSION];

The identifier is the name of variable or constant. A CONSTANT is an identifier that must be initialized and whose value cannot be change inside the program body. A NOT NULL constraint can be used for variable and they must be initialized. The DEFAULT clause or: = (assignment operator) can be used to initializes a constant or a variable through a value.

PL / SQL ENVIRONMENT:



- **How we can write PL/SQL Block?**

```
SQL>ED PROGRAM_NAME.SQL;
Example: -
SQL>ED OUTPUT.SQL;
```

- **How we can run or execute PL/SQL Block?**

```
SQL>START PROGRAM_NAME.SQL;
          OR
SQL>@ PROGRAM_NAME.SQL;
Example: -
SQL>START OUTPUT.SQL;
          OR
SQL>@ OUTPUT.SQL;
```

1. **SET SERVEROUTPUT ON / OFF:**
   The server output is an SQL * PLUS environment parameter that displays the information as a parameter to the PUT_LINE function (it is used to display the output on the screen).

2. **SET FEEDBACK ON / OFF:**
   The feedback command displays number of rows written by a query. To avoid this set feedback to off.

3. SET VERIFY OFF:

It is used to suppress (avoid) the message: OLD and: NEW when program is executed.

4. DBMS_OUTPUT.PUT_LINE:

In order to output the data from a PL / SQL block we make the use of procedure name as PUT_LINE which is the part of system package called DBMS_OUTPUT. To use this procedure we must also use another option SET SERVEROUTPUT ON.

5. COMMENT IN PL / SQL:

Single line comment	--This is PL / SQL program
Multi line comment	/*this is PL /SQL program */

Operators

Arithmetic Operators

Following table shows all the arithmetic operators supported by PL/SQL. Let us assume variable A holds 10 and variable B holds 5, then –

Operator	Description	Example
+	Adds two operands	A + B will give 15
-	Subtracts second operand from the first	A - B will give 5
*	Multiplies both operands	A * B will give 50
/	Divides numerator by de-numerator	A / B will give 2
**	Exponentiation operator, raises one operand to the	A ** B will give

	power of other	100000

Relational Operators

Relational operators compare two expressions or values and return a Boolean result. Following table shows all the relational operators supported by PL/SQL. Let us assume variable A holds 10 and variable B holds 20, then –

Operator	Description	Example
=	Checks if the values of two operands are equal or not, if yes then condition becomes true.	(A = B) is not true.
!= <> ~=	Checks if the values of two operands are equal or not, if values are not equal then condition becomes true.	(A != B) is true.
>	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.	(A > B) is not true.
<	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.	(A < B) is true.
>=	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.	(A >= B) is not true.
<=	Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.	(A <= B) is true

Comparison Operators

Comparison operators are used for comparing one expression to another. The result is always either TRUE, FALSE or NULL.

Operator	Description	Example

LIKE	The LIKE operator compares a character, string, or CLOB value to a pattern and returns TRUE if the value matches the pattern and FALSE if it does not.	If 'Zara Ali' like 'Z% A_i' returns a Boolean true, whereas, 'Nuha Ali' like 'Z% A_i' returns a Boolean false.
BETWEEN	The BETWEEN operator tests whether a value lies in a specified range. x BETWEEN a AND b means that x >= a and x <= b.	If x = 10 then, x between 5 and 20 returns true, x between 5 and 10 returns true, but x between 11 and 20 returns false.
IN	The IN operator tests set membership. x IN (set) means that x is equal to any member of set.	If x = 'm' then, x in ('a', 'b', 'c') returns Boolean false but x in ('m', 'n', 'o') returns Boolean true.
IS NULL	The IS NULL operator returns the BOOLEAN value TRUE if its operand is NULL or FALSE if it is not NULL. Comparisons involving NULL values always yield NULL.	If x = 'm', then 'x is null' returns Boolean false.

Logical Operators

Operator	Description	Examples
and	Called the logical AND operator. If both the operands are true then condition becomes true.	(A and B) is false.
or	Called the logical OR Operator. If any of the two operands is true then condition becomes true.	(A or B) is true.
not	Called the logical NOT Operator. Used to reverse the logical state of its operand. If a condition is true then Logical NOT operator will make it false.	not (A and B) is true.

CONDITIONAL STATEMENT – IF And CASE Statements

IF STATEMENTS:

The simplest form of IF statement associates a condition with a sequence of statements enclosed by the keywords THEN and END IF (not ENDIF).

The IF statement lets you execute a sequence of statements conditionally. That is, whether the sequence is executed or not depends on the value of a condition. There are Two forms of IF statements:

1. **SIMPLE IF STATEMENT:**

```
IF <CONDITION> THEN
        STATEMENTS;
ELSE
        STATEMENTS;
END IF;
```

2. **NESTED IF STATEMENT:**

```
IF <CONDITION> THEN
        STATEMENTS;
ELSIF <CONDITION> THEN
        STATEMENTS;
ELSE
        STATEMENTS;
END IF;
```

Write a program that display whether or not entered number is EVEN or ODD.

```
DECLARE
NO NUMBER (10);

BEGIN
NO: = &NO1;

IF MOD (NO, 2) = 0 THEN
        DBMS_OUTPUT.PUT_LINE ('THE NUMBER IS EVEN');
ELSE
        DBMS_OUTPUT.PUT_LINE ('THE NUMBER IS ODD');
END IF;
END;
/
```

CASE STATEMENTS:

Like the IF statement, the CASE statement selects one sequence of statements to execute. However, to select the sequence, the CASE statement uses a selector rather than multiple Boolean expressions.
A selector is an expression, whose value is used to select one of several alternatives.

```
CASE selector
   WHEN 'value1' THEN S1;
   WHEN 'value2' THEN S2;
   WHEN 'value3' THEN S3;
   ...
   ELSE Sn;  -- default case
END CASE;
```

Example :

```
DECLARE
            NO NUMBER (10);
      BEGIN
            NO: = &NO;
            CASE NO WHEN 1
            THEN  DBMS_OUTPUT.PUT_LINE ('THE NUMBER IS ONE');
            CASE NO WHEN 2
            THEN  DBMS_OUTPUT.PUT_LINE ('THE NUMBER IS TWO');
            CASE NO WHEN 3
            THEN  DBMS_OUTPUT.PUT_LINE ('THE NUMBER IS THREE');
            CASE NO WHEN 4
            THEN  DBMS_OUTPUT.PUT_LINE ('THE NUMBER IS FOUR');
            CASE NO WHEN 5
            THEN  DBMS_OUTPUT.PUT_LINE ('THE NUMBER IS FIVE');
            ELSE
            DBMS_OUTPUT.PUT_LINE ('THE NUMBER IS INVALID');
            END CASE;
      END;
      /
```

Controlling Loop iterations-Loop, Exit, Exitwhen, While, For

LOOP COMMAND:

SYNTAX : (Using exit)

```
LOOP
      <COMMANDS>;
      <COMMANDS>;
      EXIT
            <COMMANDS>;
            <COMMANDS>;
END LOOP;
```

Description:

The loop command initializes a group of statements indefinitely or until a condition forces a break from the loop.

Syntax: (Using Exit When)

```
LOOP
        <COMMANDS>;
        <COMMANDS>;
    EXIT WHEN <CONDITION>
        <COMMANDS>;
        <COMMANDS>;
END LOOP;
```

Example :

Write a program that prints 1 to 100 number using LOOP Command.

```
DECLARE
        I NUMBER (10);

BEGIN
        LOOP
                I: = I+1;
        EXIT WHEN I >= 100;
                DBMS_OUTPUT.PUT_LINE ('THE LOOP NUMBER IS   '|| I);
        END LOOP;
END;
/
```

WHILE COMMAND:

Syntax:

```
WHILE <CONDITION> LOOP
        <COMMANDS>;
END LOOP;
```

Description:

The WHILE commands is another control structure. This structure only executes the command if the condition is true. In WHILE , the condition is checked at the beginning of the command.

Example:

Write a program that prints 1 to 100 number using WHILE LOOP Command.

```
DECLARE
I NUMBER (10);
BEGIN
        WHILE I <= 100 LOOP
                DBMS_OUTPUT.PUT_LINE ('THE LOOP NUMBER IS   '|| I);
                I: = I+1;
        END LOOP;
END;
/
```

Write a program to enter data in the table using the WHILE LOOP. Use EMP table with following fields (NO, NAME, SAL, DOJ)

```
        DECLARE
        I NUMBER (4): = 1;

        BEGIN
                WHILE I <= 10 LOOP
                        INSERT INTO EMP (N0, DOJ) VALUES (I, SYSDATE);
                        I: = I+1;
                END LOOP;
        END;
        /
```

FOR LOOP: -

Syntax:

```
FOR <VARIABLE_NAME> IN [REVERSE]
<START NUMBER> .. <END NUMBER> LOOP
    <SET OF STATEMENTS>;
END LOOP;
```

Description:

Variables used as counter in the for loop need not be declared. This control structure executes the command in the loop number of times from Start number to end number. The reverse keyword is optional and we can use if we want to print the values in reverse order.

Example:

Write a program that print 1 to 100 numbers using FOR LOOP.

```
DECLARE
        I NUMBER (10);

BEGIN
        FOR I IN 1..100
        LOOP
                DBMS_OUTPUT.PUT_LINE ('THE LOOP NUMBER IS   '|| I);
        END LOOP;
END;
```

Sequential Control Statement : GOTO

GOTO COMMAND:

Syntax:

```
GOTO<CODEBLOCK NAME>;
```

Description:

The Goto statement changes the flow of control within a PL/SQL block. The statement allows the execution of a section of code, which is not the normal flow of control

Example:

Write a program to display use of GOTO statement

```
DECLARE
    a number(2) := 10;
BEGIN
    <<loopstart>>
    WHILE a < 20 LOOP
      dbms_output.put_line ('value of a: ' || a);
      a := a + 1;
      IF a = 15 THEN
        a := a + 1;
        GOTO loopstart;
      END IF;
    END LOOP;
END;
  /
```

Labeling a PL/SQL Loop

PL/SQL loops can be labeled. The label should be enclosed by double angle brackets (<< and >>) and appear at the beginning of the LOOP statement. The label name can also appear at the end of the LOOP statement. You may use the label in the EXIT statement to exit from the loop.

The following program illustrates the concept −

```
DECLARE
   i number(1);
   j number(1);
BEGIN
   << outer_loop >>
   FOR i IN 1..3 LOOP
      << inner_loop >>
      FOR j IN 1..3 LOOP
         dbms_output.put_line('i is: '|| i || ' and j is: ' || j);
      END loop inner_loop;
   END loop outer_loop;
```

```
END;
/
```

The Loop Control Statements

1. **Exit**
   The Exit statement completes the loop and control passes to the statement immediately after the END LOOP.

2. **Continue**
   Causes the loop to skip the remainder of its body and immediately retest its condition prior to reiterating.

3. **Goto**
   Transfers control to the labeled statement. Though it is not advised to use the GOTO statement in your program.