PL/SQL

WHAT IS PL/SQL?

- PL/SQL stands for Procedural Language extension of SQL.
- PL/SQL is a combination of SQL along with the procedural features of programming languages.
- Oracle uses a PL/SQL engine to processes the PL/SQL statements. A PL/SQL code can be stored in the client system (client-side) or in the database (server-side).
-

A SIMPLE PL/SQL BLOCK

- Each PL/SQL program consists of SQL and PL/SQL statements which from a PL/SQL block.

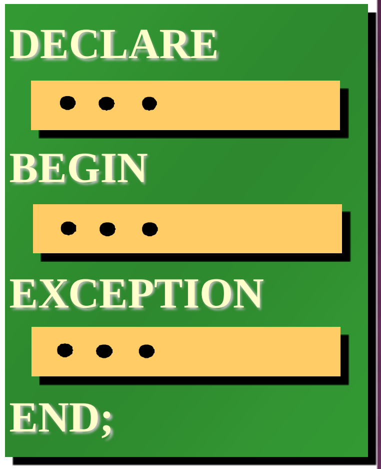
- A PL/SQL Block consists of three sections:
  - The Declaration section (optional).
  - The Execution section (mandatory).
  - The Exception (or Error) Handling section (optional).

A SIMPLE PL/SQL BLOCK

```
DECLARE
            Variable declaration;
BEGIN
            Program Execution;
EXCEPTION
            Exception handling;
END;
```

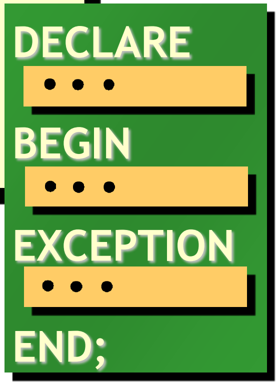
PL/SQL BLOCK STRUCTURE

- 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 BLOCK STRUCTURE

```
DECLARE
  v_variable  VARCHAR2(5);
BEGIN
  SELECT  column_name
    INTO  v_variable
    FROM  table_name;
EXCEPTION
  WHEN exception_name THEN
  ...
END;
```



DECLARATION SECTION

- Starts with the reserved keyword DECLARE.

- Used to declare any placeholders like variables, constants, records and cursors

EXECUTION SECTION

- Starts with the reserved keyword BEGIN
- Ends with END
- Program logic is written to perform any task.
- The programmatic constructs like loops, conditional statement and SQL statements form the part of execution section.

EXCEPTION SECTION

- Starts with the reserved keyword EXCEPTION.
- Any errors in the program can be handled

CONTD…

- Every statement in the three sections must end with a semicolon **;**
- PL/SQL blocks can be nested within other PL/SQL blocks.
- Comments can be used to document code.

PL/SQL PLACEHOLDERS

- Temporary storage area
- Any of Variables, Constants and Records.
- Define placeholders with a name and a datatype.
- Datatypes :-
  - Number (n,m) ,
  - Char (n) ,
  - Varchar2 (n) ,
  - Date ,
  - Long ,
  - Long raw,
  - Raw, Blob,
  - Clob,
  - Nclob,
  - Bfile

SYNTAX TO DECLARE A VARIABLE

variable_name   datatype [NOT NULL := value];

- *variable_name* is the name of the variable.
- *datatype* is a valid PL/SQL datatype.
- NOT NULL is an optional specification on the variable.
- *value* or DEFAULT *value* is also an optional specification, where you can initialize a variable.
- Each variable declaration is a separate statement and must be terminated by a semicolon.

E.G.

```
DECLARE
salary number (6);

DECLARE
salary number(4);
dept varchar2(10) NOT NULL := “HR Dept”;
```

```
Declare
   birthday          DATE;
   age               NUMBER(2) NOT NULL := 27;
   name              VARCHAR2(13) := 'Levi';
   magic             CONSTANT NUMBER := 77;
   valid             BOOLEAN NOT NULL := TRUE;
```

CONDITIONAL STATEMENTS IN PL/SQL

- PL/SQL supports programming language features like conditional statements, iterative statements.
- **IF THEN ELSE STATEMENT**

```
IF condition THEN
statement 1;
ELSE  statement 2;
END IF;
```

- Nested IF **THEN ELSE STATEMENT**

```
IF condition 1 THEN
statement 1;
ELSIF condtion2 THEN
statement 2;
ELSE
statement 3;
END IF
```

ITERATIVE STATEMENTS IN PL/SQL

- Iterative control Statements are used when we want to repeat the execution of one or more statements for specified number of times.
- **There are three types of loops in PL/SQL:**
  - Simple Loop
  - While Loop
  - For Loop

SIMPLE LOOP

- It is used when a set of statements is to be executed at least once before the loop terminates.
- An EXIT condition must be specified in the loop, otherwise the loop will get into an infinite number of iterations.
- When the EXIT condition is satisfied the process exits from the loop.
- **General Syntax to write a Simple Loop is**

```
LOOP
statements;
EXIT;
{or EXIT WHEN condition;}
END LOOP;
```

- These are the important steps to be followed while using Simple Loop.
  - Initialize a variable before the loop body.
  - Increment the variable in the loop.
  - Use a EXIT WHEN statement to exit from the Loop.
  - If you use a EXIT statement without WHEN condition, the statements in the loop is executed only once.

```
create table number_table (num NUMBER(10));
DECLARE
  i  number_table.num%TYPE := 1;
BEGIN
  LOOP
    INSERT INTO number_table VALUES(i);
    i := i + 1;
    EXIT WHEN i > 10;
  END LOOP;
END;
```

WHILE LOOP

- It is used when a set of statements has to be executed as long as a condition is true.
- The condition is evaluated at the beginning of each iteration.
- The iteration continues until the condition becomes false.
- The General Syntax to write a WHILE LOOP is:

```
WHILE <condition>
LOOP statements;
END LOOP;
```

- Important steps to follow when executing a while loop:
  - Initialize a variable before the loop body.
  - Increment the variable in the loop.

```
DECLARE
TEN number:=10;
i number_table.num%TYPE:=1;
BEGIN
  WHILE i <= TEN LOOP
     INSERT INTO number_table VALUES(i);
     i := i + 1;
  END LOOP;
END;
```

FOR LOOP

- It is used to execute a set of statements for a predetermined number of times.
- Iteration occurs between the start and end integer values given.
- The counter is always incremented by 1.
- The loop exits when the counter reaches the value of the end integer.
- The General Syntax to write a FOR LOOP is:

```
FOR counter IN val1..val2
LOOP statements;
END LOOP;
```

- Important steps to follow when executing a while loop:
  - The counter variable is implicitly declared in the declaration section, so it's not necessary to declare it explicitly.
  - The counter variable is incremented by 1 and does not need to be incremented explicitly.

```
DECLARE
 i number_table.num%TYPE;
BEGIN
 FOR i IN 1..10 LOOP
   INSERT INTO number_table VALUES(i);
 END LOOP;
END;
```

PRINTING OUTPUT

- You need to use a function in the DBMS_OUTPUT package in order to print to the output
- If you want to see the output on the screen, you must type the following (before starting):

set serveroutput on

- Then print using
  - dbms_output. put_line*(your_string);*
  - dbms_output.put(*your_string*);

INPUT AND OUTPUT EXAMPLE

```
set serveroutput on
ACCEPT high PROMPT 'Enter a number: ‘
DECLARE
   i            number_table.num%TYPE:=1;
BEGIN
   dbms_output.put_line(‘Accept Value from User…….!!!');
   WHILE    i <= &high   LOOP
                INSERT INTO number_table  VALUES(i);
                 i := i + 1;
   END LOOP;
END;
```

ASSIGN VALUES TO VARIABLES

1. variable_name:= value;

2. SELECT column_name INTO variable_name FROM table_name [WHERE condition];

SELECT STATEMENTS

```
DECLARE
  v_sname       VARCHAR2(10);
  v_rating      NUMBER(3);
BEGIN
  SELECT      sname, rating
    INTO      v_sname, v_rating
    FROM      Sailors
   WHERE      sid = '112';
END;
/
```

- INTO clause is required.
- Query must return exactly one row.
- Otherwise, a NO_DATA_FOUND or TOO_MANY_ROWS exception is thrown

E.g.: The below program will get the salary of an employee with id '1116' and display it on the screen.

```
DECLARE
   var_salary number(6);
   var_emp_id number(6) := 1116;
BEGIN
   SELECT salary  INTO var_salary
   FROM employee
   WHERE emp_id = var_emp_id;
   dbms_output.put_line(var_salary);
   dbms_output.put_line('The employee ' ||
   var_emp_id || ' has salary ' || var_salary);
END;
 /
```

```
DECLARE
   var_num1 number;
   var_num2 number;
BEGIN
   var_num1 := 100;
   var_num2 := 200;
   DECLARE
          var_mult number;
          BEGIN
                   var_mult := var_num1 * var_num2;
          END;
END;
/
```

GENERAL SYNTAX TO DECLARE A CONSTANT

constant_name CONSTANT datatype := VALUE;

```
DECLARE
   salary_increase CONSTANT number(3);
BEGIN
   salary_increase := 100;
   dbms_output.put_line (salary_increase);
END;
```

PL/SQL RECORDS

- **What are records?**
- Records are another type of datatypes which oracle allows to be defined as a placeholder.
- Records are composite datatypes, which means it is a combination of different scalar datatypes like char, varchar, number etc.
- Each scalar data types in the record holds a value.
- A record can be visualized as a row of data.
- It can contain all the contents of a row.

PL/SQL RECORDS

- The General Syntax to define a composite datatype is:
- **TYPE** record_type_name **IS RECORD** (first_col_name column_datatype, second_col_name column_datatype, ...);
- *record_type_name* – it is the name of the composite type you want to define.
- *first_col_name, second_col_name, etc*.,- *it is the* names the fields/columns within the record.
- *column_datatype* defines the scalar datatype of the fields.

```
DECLARE
TYPE employee_type IS RECORD
   (employee_id number(5),
   employee_first_name varchar2(25),
   employee_last_name employee.last_name%type,
   employee_dept employee.dept%type);
   employee_salary employee.salary%type;)
```

UPDATING DATA

- Increase the salary of all employees in the emp table who are Analysts.
- Example

```
DECLARE
  v_sal_increase   emp.sal%TYPE := 2000;
BEGIN
  UPDATE     emp
  SET        sal = sal + v_sal_increase
  WHERE      job = 'ANALYST';
END;
```

DELETING DATA

- Delete rows that belong to department 10 from the emp table.
- Example

```
DECLARE
  v_deptno   emp.deptno%TYPE := 10;
 BEGIN
  DELETE FROM   emp
  WHERE        deptno = v_deptno;
END;
```

TRAPPING EXCEPTIONS

- Define the actions that should happen when an exception is thrown.
- Example Exceptions:
  - NO_DATA_FOUND
  - TOO_MANY_ROWS
  - ZERO_DIVIDE
- When handling an exception, consider performing a rollback

```
DECLARE
   num_row  number_table%ROWTYPE;
BEGIN
   select * into num_row
   from number_table;
   dbms_output.put_line(1/num_row.num);

EXCEPTION
   WHEN NO_DATA_FOUND THEN
            dbms_output.put_line('No data!');
   WHEN TOO_MANY_ROWS THEN
            dbms_output.put_line('Too many!');
   WHEN OTHERS THEN
            dbms_output.put_line(‘Error’);
end;
```

USER-DEFINED EXCEPTION

```
DECLARE
  e_number1  EXCEPTION;
  cnt       NUMBER;
BEGIN
  select count(*) into cnt
  from number_table;

  IF cnt = 1 THEN RAISE e_number1;
  ELSE dbms_output.put_line(cnt);
  END IF;

EXCEPTION
  WHEN e_number1 THEN
        dbms_output.put_line('Count = 1');
end;
```

PL/SQL EXAMPLE

```
DECLARE
        v_first_name VARCHAR2(35);
        v_last_name VARCHAR2(35);
BEGIN
    SELECT first_name, last_name
    INTO v_first_name, v_last_name
    FROM student WHERE student_id = 123;
    DBMS_OUTPUT.PUT_LINE('Student name: '||v_first_name||'
        '||v_last_name);
EXCEPTION
        WHEN NO_DATA_FOUND THEN
             DBMS_OUTPUT.PUT_LINE('There is no student with student
        id 123');
END;
/
```

```
DECLARE
I INT;
FACT INT;
A INT;
BEGIN
FACT:=1;
A:=&A;
IF A=0 THEN
  RAISE E1;
ELSIF A<0 THEN
  RAISE E2;
ELSE
FOR I IN 1..A
LOOP
    FACT:=FACT*I;
    A:=A-1;
END LOOP;
END IF
```

```
DECLARE
I INT;
FACT INT;
A INT;
E1 EXCEPTION;
E2 EXCEPTION;
BEGIN
FACT:=1;
A:=&A;
IF A=0 THEN
  RAISE E1;
ELSIF A<0 THEN
  RAISE E2;
ELSE
FOR I IN 1..A
```

```
LOOP
    FACT:=FACT*I;
    A:=A-1;
END LOOP;
END IF;
DBMS_OUTPUT.PUT_LINE('FACT='||FACT);
EXCEPTION
WHEN E1 THEN
 DBMS_OUTPUT.PUT_LINE('FACTORIAL = 1');
WHEN E2 THEN
  DBMS_OUTPUT.PUT_LINE('FACTORIAL OF -
    VE NUMBER CAN NOT BE FOUND ');
END;
```

LAB EXERCISE

- Write a PL/SQL block to calculate factorial. Use Exception Handling.
- Write a PL/SQL block to find prime number for first 30 numbers.
- Write a PL/SQL block to find Fibonacci series for first 50 numbers.
- Write a PL/SQL block to find **a** raised to **b i.e.** $\mathbf{a}^{\mathbf{b}}$
- Write a PL/SQL block to find the grade of a student. Enter marks for 5 subjects.