PL/SQL

What is PL/SQL

- Procedural Language – SQL
- An extension to SQL with design features of programming languages (procedural and object oriented)
- PL/SQL and Java are both supported as internal host languages within Oracle products.

Why PL/SQL

- Acts as host language for stored procedures and triggers.
- Provides the ability to add middle tier business logic to client/server applications.
- Improves performance of multi-query transactions.
- Provides error handling

PL/SQL BLOCK STRUCTURE

DECLARE (*optional*)

- variable declarations

BEGIN (*required*)

- SQL statements
- PL/SQL statements or sub-blocks

……

EXCEPTION (*optional*)

- actions to perform when errors occur

…….

END; (*required*)

PL/SQL Block Types

Anonymous

```
DECLARE
BEGIN
    -statements
EXCEPTION
END;
```

a.sql

Procedure

```
PROCEDURE <name>
IS
BEGIN
    -statements
EXCEPTION
END;
```

p.sql

Function

```
FUNCTION <name>
RETURN <datatype>
IS
BEGIN
    -statements
EXCEPTION
END;
```

f.sql

PL/SQL Variable Types

- Scalar (char, varchar2, number, date, etc)
- Composite (%rowtype)

DECLARE

Syntax

```
identifier [CONSTANT] datatype [NOT NULL]
    [:= | DEFAULT expr];
```

Examples

Notice that PL/SQL includes all SQL types, and more...

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

PL/SQL- Assignment

- All variables must be declared before their use.
- The assignment statement

  : =

  is not the same as the equality **=**(comparison) operator

  =

- All statements end with a **;** semicolon

DBMS_OUTPUT.PUT_LINE()

- **Printing on the screen**
- DBMS_OUTPUT is the package , defined with function PUT_LINE(string variable) .
- string variable value passed can be displayed on the screen.
- Before using DBMS_OUTPUT.PUT_LINE(..) , use SET SERVEROUTPUT ON at SQL prompt

Example:

```
SET SERVEROUTPUT ON
DBMS_OUTPUT.PUT_LINE(‘ HELLO ….’);
DBMS_OUTPUT.PUT_LINE(‘MY Register Number ‘ ||to_char(12345));
```

|| symbol concatenates two strings.

PL/SQL FIRST PROGRAM

```
SET SERVEROUTPUT ON
DECLARE
   message  varchar2(20):= 'Hello, World! ';
BEGIN
   dbms_output.put_line(message);
END;
/
```

PL/SQL Sample Program

```
/* Find the area of the circle*/
SET SERVEROUTPUT ON
DECLARE
   pi constant number:=3.14;
   radius number:=2;
   area number;
BEGIN
   area:=pi*radius*radius;
   dbms_output.put_line('Area of circle is:'||area);
END;
/
```

PL/SQL sample program

```
--Find the area of the circle
SET SERVEROUTPUT ON
DECLARE
   pi constant number:=3.14;
   radius number:=&radius;
   area number;
BEGIN
   area:=pi*power(radius,2);
   dbms_output.put_line('Area of circle is:'||area);
END;
/
```

Retrieving Column values into variables

SELECTING Columns Value Into Variables

```
SELECT Column1,Column2, .. INTO
Variabl1, Variabl2,.. FROM
table.. WHERE.. ;
```

Note: This select statement must retrieve one single record

Tables to use in PL/SQL Example

- Create table circle(radius number(2),area number(5,1), circum number(5,1))

- Insert into circle(radius) values(2);
- Insert into circle(radius) values(3);
- Insert into circle(radius) values(4);

Comments:

Single line comments -- This is 1st PL/SQL block

Multiline comments /* This is 1st PL/SQL block … */

%type and SELECT.. INTO..

```
DECLARE
  v_radius  circle.radius%TYPE;
  V_area circle.area%TYPE;
BEGIN
  SELECT radius INTO v_radius FROM circle WHERE
 ROWNUM = 1;
  DBMS_OUTPUT.PUT_LINE(' Radius = ' || v_radius);
 V_area:=3.142*power(v_radius,2);
 Update circle set Area=v_area where
 radius=v_radius;
END;
```

%TYPE

%TYPE is used to declare a field with the same type as that of a specified table's column:

```
DECLARE
  v_EmpName  emp.ename%TYPE;
  v_empno emp.empno%TyPE;
   v_sal  emp.sal%type;
BEGIN
 v_empno:=& v_empno;
  SELECT ename,sal INTO v_EmpName,v_sal  FROM emp WHERE empno =v_empno;
  DBMS_OUTPUT.PUT_LINE('Name = ' || v_EmpName|| '  Salary ' || v_sal);
END;
/
```

%ROWTYPE

-- %ROWTYPE is used to declare a record with the same types as found in the specified database table, view or cursor:

```
DECLARE
  v_emp emp%ROWTYPE;
BEGIN
  v_emp.empno := 10;
  v_emp.ename := 'XXXXXXX';
END;
/
```

%ROWTYPE

```
Set serveroutput on
DECLARE
  v_dept dept%rowtype;
BEGIN
  select * into v_dept
    from dept where deptno=10;

  DBMS_OUTPUT.PUT_LINE (v_dept.deptno);
  DBMS_OUTPUT.PUT_LINE (v_dept.dname);
  DBMS_OUTPUT.PUT_LINE (v_dept.loc);
END;
/
```

Example:

Assume that we have a table

STUD(<u>RegNo</u>, Name, Mark1,Mark2,Mark3)

Write a PL/SQL block to find the marks details of a student depending on the Registration Number input by the user. Also display Total and Average marks of the student.

COMMON IN-BUILT STRING FUNCTIONS

- CHR(asciivalue)
- ASCII(string)
- LOWER(string)
- SUBSTR(string,start,substrlength)
- LTRIM(string)
- RTRIM(string)
- LPAD(string_to_be_padded, spaces_to_pad, |string_to_pad_with|)
- RPAD(string_to_be_padded, spaces_to_pad, |string_to_pad_with|)
- REPLACE(string, searchstring, replacestring)
- UPPER(string)
- INITCAP(string)
- LENGTH(string)

COMMON IN-BUILT NUMERIC FUNCTIONS

- ABS(value)
- ROUND(value, precision)
- MOD(value,divisor)
- SQRT(value)
- TRUNC(value,|precision|)
- LEAST(exp1, exp2…)
- GREATEST(exp1, exp2…)

Conditional logic

Condition:

```
If <cond>
  Then <command>
Elsif <cond2>
  Then <command2>
Else
      <command3>
End if;
```

Nested conditions:

```
If <cond>
 Then
    If <cond2>
      Then
        <command1>
     End if;
 Else <command2>
 end if;
```

IF-THEN-ELSIF Statements

```
. . .
IF rating > 7 THEN
  v_message := 'You are great';
ELSIF rating >= 5 THEN
  v_message := 'Not bad';
ELSE
  v_message := 'Pretty bad';
END IF;
. . .
```

CASE.. WHEN Statement

- The CASE statement selects one sequence of statements to execute among multiple sequences.

```
CASE e
        WHEN e1 THEN r1
        WHEN e2 THEN r2
        ……................
        WHEN en THEN rn
[ ELSE r_else ]
END CASE;
```

CASE.. WHEN- Example

```
DECLARE
grade CHAR(1); BEGIN
grade := & grade;
CASE grade
    WHEN 'A' THEN  DBMS_OUTPUT.PUT_LINE('Excellent');
        DBMS_OUTPUT.PUT_LINE('A - Grade');
    WHEN 'B' THEN  DBMS_OUTPUT.PUT_LINE('Very Good');
        DBMS_OUTPUT.PUT_LINE('B - Grade');
    WHEN 'C' THEN  DBMS_OUTPUT.PUT_LINE('Good');
        DBMS_OUTPUT.PUT_LINE('C - Grade');
    WHEN 'D' THEN  DBMS_OUTPUT.PUT_LINE('Fair');
        DBMS_OUTPUT.PUT_LINE('D - Grade');
    WHEN 'F' THEN  DBMS_OUTPUT.PUT_LINE('Poor');
        DBMS_OUTPUT.PUT_LINE('F - Grade');
    ELSE  DBMS_OUTPUT.PUT_LINE('No such grade');
END CASE;
END;
```

Loops: Simple Loop

```
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;
```

Loops: FOR Loop

```
FOR counter IN [REVERSE] initial_value .. final_value
LOOP
    ……————————————
    sequence_of_statements;
    ————————————————————————
END LOOP;
```

Notice that i is incremented automatically

Example-Loops: FOR Loop

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

Notice that i is incremented automatically

Example-Loops: FOR Loop (REVERSE)

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

Notice that i is incremented automatically by 1

Example-Loops: WHILE 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;
```

Cursors

CURSORS

- A cursor is a private memory area.
- Set of records returned by Query are stored in Cursor.
- Data in Cursor is Active Data Set
- An Oracle Cursor = VB record set = JDBC Result Set
- Implicit cursors are created for every query made in Oracle
- Explicit cursors can be declared by a programmer within PL/SQL.

Implicit cursor

```
SELECT emp_no, emp_name, job, salary
FROM employee
WHERE dept = 'physics'
```

1234	A. N. Sharanu	Asst. Professor	22,000.00
1345	N. Bharath	Senior Lecturer	17,000.00
1400	M. Mala	Lab Incharge	9,000.00

- ✓ Cursor is Automatically –Opened
- ✓ All Records Retrieved.
- ✓ Cursor is Closed

Implicit Cursor Attributes

- SQL%ROWCOUNT — Rows returned so far (Number)
- SQL%FOUND — One or more rows retrieved (Boolean)
- SQL%NOT FOUND — No rows found (Boolean)
- SQL%ISOPEN — Is the cursor open (Boolean)

Implicit Cursor

```
SET SERVEROUTPUT ON
BEGIN
 update dept set loc='&location' where deptno=&dno;
 if SQL%found then
        DBMS_OUTPUT.PUT_LINE('Department Successfully
transferred');
  end if;
if SQL%notfound then
        DBMS_OUTPUT.PUT_LINE('Department not existing');
  end if;
END;
/
```

Explicit cursor

- Declare the cursor
- Open the cursor
- Fetch data from the cursor record by record
- Close the cursor

1234	A. N. Sharanu	Asst. Professor	22,000.00
1345	N. Bharath	Senior Lecturer	17,000.00
1400	M. Mala	Lab Incharge	9,000.00

Explicit cursor

ORACLE keep track of the "current status" of the cursor through- Cursor Attributes(system variables)

- **%NOTFOUND:** Evaluates to TRUE if the last fetch is failed i.e. no more rows are left. (single word)
  Syntax: cursor_name %NOTFOUND

- **%FOUND:** Evaluates to TRUE, when last fetch succeeded
  Syntax: cursor_name %FOUND

- **%ISOPEN:** Evaluates to TRUE, if the cursor is opened, otherwise evaluates to FALSE.
  Syntax: cursor_name %ISOPEN

- **%ROWCOUNT:** Returns the number of rows fetched.
  Syntax: cursor_name %ROWCOUNT

Explicit Cursor Control

- Declare the cursor — **CURSOR Cur_name IS SELECT… FROM… WHERE..;**
- Open the cursor — **OPEN Cur_name;**
- Fetch a row — **FETCH Cur_name INTO var1,var2,..**
- Test for end of cursor — **Cur_name% NOTFOUND / FOUND**
- Close the cursor — **CLOSE Cur_name;**

loop

Example-1: Write a PL/SQL Block to retrieve Employee name and their salary if salary is more than 3000

Assume the tables – EMP(Empno, Ename, Sal, Deptno)

DEPT(Deptno, Dname, Bugdet)

Example-1

```
DECLARE
   cursor c_emp is      -- Cursor Declaration
     select ename,sal  from emp  where sal>=3000;
   v_ename  emp.ename%TYPE;
   v_salary  emp.sal%TYPE;
BEGIN
open c_emp;              -- Open Cursor
loop
  fetch c_emp into v_ename,v_salary;        -- Fetch Record
  exit when c_emp%notfound;                 -- Test End of Cursor
  DBMS_OUTPUT.PUT_LINE(v_ename||' draws '||v_salary||' as salary');
end loop;
DBMS_OUTPUT.PUT_LINE('Number records : '||c_emp%rowcount);
close c_emp;        -- Close cursor
END;
```

Example-2

Assume we have two tables- **EMP(Empno, Ename, Sal, Deptno)** Deptno references DEPT & **DEPT(Deptno, Dname, Budget);**

Write a PL/SQL block to increase salary of employees by 5%, 10% 15% depending on the Budget of their Department. Salary increment criteria is as Below-

If Budget is <= 200000 , 5%

Budget >200000 and Budget < =400000 , 10%

Budget > 400000 , 15%

Example-2..

```
DECLARE
   cursor c_emp is select ename,sal,Budget  from emp ,Dept
   where  emp.deptno=dept.deptno;

 v_ename  emp.ename%TYPE;
   v_salary emp.sal%TYPE;
   v_Budget dept.budget%Type;
   updated_sal emp.sal%TYPE;

BEGIN
open c_emp;              -- Open Cursor
loop
  fetch c_emp into v_ename,v_salary,v_Budget;
  exit when c_emp%notfound;
```

..Example-2

```
  IF v_Budget<=200000 THEN
     updated_sal:= v_salary+v_salary*0.05;
  ELSIF v_Budget>200000 AND v_Budget<= 400000 THEN
            updated_sal:= v_salary+v_salary*0.1;
  ELSE
    updated_sal:= v_salary+v_salary*0.15;
  END IF;
  DBMS_OUTPUT.PUT_LINE(' Name : '||v_ename);
  DBMS_OUTPUT.PUT_LINE(' Old Salary : '||v_salary);
  DBMS_OUTPUT.PUT_LINE(' New Salary : '||updated_sal);
  DBMS_OUTPUT.PUT_LINE('=====================');
end loop;
close c_emp;        -- Close cursor
END;
/
```

Explicit Cursor- cursor for loop

Cursor for loop simplifies the usage of explicit cursor. In each cursor PL/SQL following procedure is needed-

- Opening Cursor
- Fetching record and variable to hold fetched values.
- Exiting from loop
- Closing loop

In cursor for loop, no need of writing above steps explicitly.

Explicit Cursor- cursor for loop

Write a PL/SQL block to employee name and salary information of the employees who has salary more than 3000

```
DECLARE
   cursor c_emp is Select Ename, Sal from Emp where
                        Sal>=3000;
BEGIN
for i in c_emp
loop
   DBMS_OUTPUT.PUT_LINE(i.ename||' draws '||i.sal||' as
   salary');
end loop;
END;
/
```

Parameterized Cursor

Example 3:

Write a PL/SQL block to retrieve employee name, salary information of employees in the following format depending on the department number entered by the user.

Assume we have table EMP(<u>Empno</u>, Ename, Sal, Deptno)

Expected Output

Enter value for par_dept: 10

Raghu draws 10900 as salary

Ravi draws 20000 as salary

Example 3-Parameterized Cursor

```
SET SERVEROUTPUT ON
DECLARE
CURSOR cur_emp (par_dept number) IS SELECT ename, sal FROM emp
                                   WHERE deptno = par_dept  ORDER BY ename;
v_ename  emp.ename%TYPE;
v_salary  emp.salary%TYPE;
BEGIN
  OPEN cur_emp (& par_dept);
  LOOP
        FETCH cur_emp INTO v_ename, v_salary;
        EXIT WHEN cur_emp%NOTFOUND;
        DBMS_OUTPUT.PUT_LINE(v_ename||' draws '||v_salary||' as salary');
  END LOOP;
  close cur_emp ;
  END;
  /
```

Parameterized Cursor

Example 4:

Write a PL/SQL block using Cursor for loop to retrieve employee name, salary information of employees in the following format depending on the department number entered by the user.

Assume we have table EMP(<u>Empno</u>, Ename, Sal, Deptno)

Expected Output

Enter value for par_dept: 10

Raghu draws 10900 as salary

Ravi draws 20000 as salary

Example 4

```
DECLARE
CURSOR cur_emp (par_dept number) IS SELECT ename,
sal FROM emp
        WHERE deptno = par_dept  ORDER BY ename;
BEGIN
  for emp_rec in cur_emp(&par_dept)
 LOOP
        DBMS_OUTPUT.PUT_LINE(emp_rec.ename||'
draws '||emp_rec.sal||' as salary');
 END LOOP;
 END;
 /
```

END-PLSQL & CURSOR