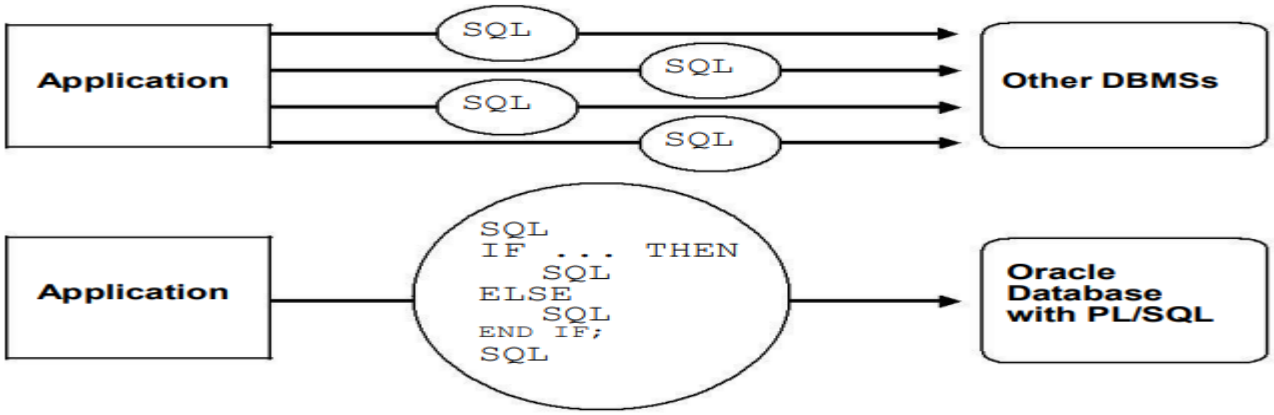
Unit 5 PL/SQL

What is PL/SQL

- Procedural Language – SQL
- An extension to SQL with design features of programming language (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 ;

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:

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;
/
```

- 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);

Retrieving Column values into variables

SELECTING Columns Value Into Variables

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

%type

```
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;
```

Example

Consider the table :

Emp(<u>Empno</u>,EmpName,Salary,deptno)

Dept(<u>Dno</u>,Dname,location)

%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 dno='D1';

  DBMS_OUTPUT.PUT_LINE (v_dept.dno);
  DBMS_OUTPUT.PUT_LINE (v_dept.dname);
  DBMS_OUTPUT.PUT_LINE (v_dept.location);
END;
/
```

COMMON PL/SQL 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 PL/SQL NUMERIC FUNCTIONS

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

Create a Table EMPSAL with fields-Empno, Empname, Sal, HRA, DA, Gross Salary, PF, Net Salary (assume appropriate datatype and size).

Write a PL/SQL block to accept an employee number existing in EMP table and calculate HRA, DA, Gross Salary, PF, Net_Salary of that employee.

Insert the Empno, Empname, Sal, HRA, DA, Gross Salary, PF, Net Salary into the table EMPSAL:

Use the following formula to calculate salary components:

HRA=50% of Sal

DA=20% of Sal

PF=12% of Sal.

Gross_sal= Sal+ HRA+DA

Net_Sal= Gross_sal-PF

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’);
        WHEN 'B' THEN DBMS_OUTPUT.PUT_LINE('Very Good’);
        WHEN 'C' THEN DBMS_OUTPUT.PUT_LINE('Good');
        WHEN 'D' THEN DBMS_OUTPUT.PUT_LINE('Fair');
        WHEN 'F' THEN DBMS_OUTPUT.PUT_LINE('Poor’);
        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

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

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

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 set of records
- An Oracle Cursor = VB recordset = JDBC ResultSet
- Implicit cursors are created for every query made in Oracle
- Explicit cursors can be declared by a programmer within PL/SQL.

Implicit Cursor Attributes

- SQL%ROWCOUNT Rows returned so far
- SQL%FOUND One or more rows retrieved
- SQL%NOTFOUND No rows found
- SQL%ISOPEN Is the cursor open

Loops: For Cursor Loops

```
DECLARE
  cursor c is select * from number_table;

BEGIN
   for num_row in c loop
       insert into doubles_table
                        values(num_row.num*2);
   end loop;
END;
/
```

Notice that a lot is being done implicitly: declaration of num_row, open cursor, fetch cursor, the exit condition

Implicit Cursor

```
SET SERVEROUTPUT ON
BEGIN
 update dept set location='&location' where dno='&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 Control

- Declare the cursor
- Open the cursor
- Fetch a row
- Test for end of cursor
- Close the cursor

Explicit Cursor Attributes

• cursorname%ROWCOUNT	Rows returned so far
• cursorname%FOUND	One or more rows retrieved
• cursorname%NOTFOUND	No rows found
• Cursorname%ISOPEN	Is the cursor open

Sample Program

```
DECLARE
   cursor c_emp is
     select ename,salary  from emp  where salary>30000;
   v_ename  emp.ename%TYPE;
   v_salary  emp.salary%TYPE;
BEGIN
open c_emp;
loop
  fetch c_emp into v_ename,v_salary;
  exit when c_emp%notfound;
    DBMS_OUTPUT.PUT_LINE(v_ename||' draws '||v_salary||' as salary');
end loop;
close c_emp;
END;
```

Explicit Cursor

```
DECLARE
   cursor c_emp is      select ename,salary
          from emp     where salary>30000;
BEGIN
for i in c_emp
loop
    DBMS_OUTPUT.PUT_LINE(i.ename||' draws '||i.salary||'
as salary');
end loop;
END;
/
```

Parameterized Cursor

```
SET SERVEROUTPUT ON
DECLARE
CURSOR cur_emp (par_dept VARCHAR2) IS SELECT ename, salary 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;
 END;
 /
```