Sub Programs
Functions & Procedures

By
Parteek Bhatia
Assistant Professor
Dept of Comp Sc & Engg
Thapar University
Patiala

Subprograms

- Subprograms are named PL/SQL blocks that can take parameters and can be invoked. Subprograms allow decomposition of a program into logical units.

PL/SQL has two types of subprograms:

- Procedures
- Functions

They can be further classified as:

- Local subprograms
- Stored Subprograms

Local and Stored Subprograms

- **Local Subprograms**

  Such procedures and functions are local to the PL/SQL module, which contain it. They can be created in the declarative section of PL/SQL module, local to the module. The local module can be called anywhere in the module's execution section.
- **Stored Subprograms**

  A stored procedure or function is a named PL/SQL code block that have been compiled and stored in one of the Oracle engine's system tables.They are invoked or called by any the PL/SQL block that appear within an application.

Difference between Procedure and Function

The comparison between procedure and function is indicated in the following table:

Procedure	Function
A Procedure is a subprogram that can accept parameters, perform an action and return parameters.	A Function is a subprogram that can accept parameters, compute a value and return that value to caller.
A procedure may return no value.	A Function must return only one value.
A procedure cannot return a value as direct output.	A Function can return a value as direct output of the function call.
Generally, we use a procedure to perform an action.	Generally, we use a function to compute a value.

Local Procedure

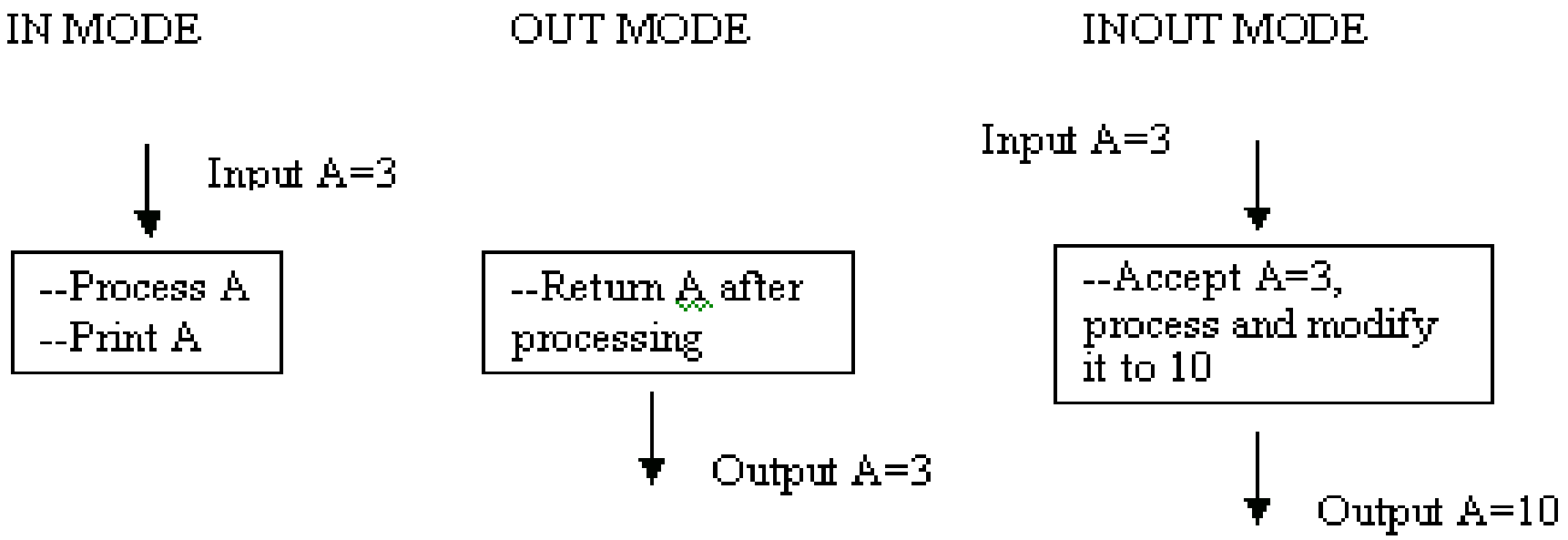
```
DECLARE
      --Declaration of Global variables
      PROCEDURE name
       [(argument {IN, OUT, IN OUT} data type, ....)]  {IS | AS}
        [local declarations]
      BEGIN
          PL/SQL subprogram body;
      [EXCEPTION
        exception handlers]
      END [name];
BEGIN
          --Executable code
          --Code to call procedure
  [EXCEPTION
          exception handlers for main block]
END;
```

Arguments

IN	OUT	IN OUT
It is the default.	It must be specified.	It must be specified.
It passes values to a subprogram.	It returns values to the caller.	It passes initial values to a subprogram and returns updated values to the caller.
It is a formal parameter acts like a constant.	It is a formal parameter acts like an uninitialized variable.	It is a formal parameter acts like an initialized variable.

IN	OUT	IN OUT
It is a formal parameter cannot be assigned a value.	It is a formal parameter cannot be used in an expression and must be assigned a value.	It is a formal parameter should be assigned a value.
It can be a constant, initialized variable, literal, or expression.	It must be a variable.	It must be a variable.

Table 20.2



Consider a procedure that accepts two numbers and return addition, subtraction, multiplication and division of two numbers or in other words a procedure to return multiple values through arguments.

```
DECLARE
        A NUMBER;
        B NUMBER;
        C NUMBER;
        D NUMBER;
        E NUMBER;
        F NUMBER;
        PROCEDURE PROCESS( A IN NUMBER, B IN NUMBER ,C OUT
        NUMBER, D OUT NUMBER, E OUT NUMBER, F OUT NUMBER)
        IS
        BEGIN
                C=A+B;
                D=A-B;
                E=A*B;
                F=A/B;
        END;
```

```
BEGIN
        A=&FIRSTNUMBER;
        B=&SECONDNUMBER;
        PROCESS(A,B,C,D,E,F);
        DBMS_OUTPUT.PUT_LINE('ADDITION IS'||C);
        DBMS_OUTPUT.PUT_LINE('SUBTRACTION IS'||D);
        DBMS_OUTPUT.PUT_LINE('MUTIPLICATION IS'||E);
        DBMS_OUTPUT.PUT_LINE('DIVISION IS'||F);
END;
```

Stored Procedure

```
CREATE OR REPLACE PROCEDURE procedurename
        (argument {IN, OUT, IN OUT} datatype,....) {IS | AS}
        [local declarations];
BEGIN
        PL/SQL subprogram body;
[EXCEPTION
        Exception PL/SQL block;]
END;
```

Note:
If an error occurs during compilation of the procedure or function an invalid procedure or function is created. These errors can be viewed using the select statement:
SELECT *FROM USER_ERRORS;
Or
Show errors;

A Stored procedure that accepts two numbers and return addition, subtraction, multiplication and division of two numbers or in other words a stored procedure to return multiple values through arguments.

```
CREATE OR REPLACE PROCEDURE PROCESS( A IN NUMBER, B IN
NUMBER, C OUT NUMBER, D OUT NUMBER, E OUT NUMBER, F OUT
NUMBER) IS


BEGIN
        C:=A+B;
        D:=A-B;
        E:=A*B;
        F:=A/B;
END;
```

A PL/SQL code to call the procedure PROCESS

```
DECLARE
        A NUMBER;
        B NUMBER;
        C NUMBER;
        D NUMBER;
        E NUMBER;
        F NUMBER;
BEGIN
        A:=&FIRSTNUMBER;
        B:=&SECONDNUMBER;
        PROCESS(A,B,C,D,E,F);
        DBMS_OUTPUT.PUT_LINE('ADDITION IS'||C);
        DBMS_OUTPUT.PUT_LINE('SUBTRACTION IS'||D);
        DBMS_OUTPUT.PUT_LINE('MUTIPLICATION IS'||E);
        DBMS_OUTPUT.PUT_LINE('DIVISION IS'||F);
END;
```

A stored procedure fire_employee to delete employee on the basis of employee number.

```
CREATE PROCEDURE fire_employee (emp_id NUMBER)
AS
BEGIN
DELETE FROM emp WHERE empno = emp_id;
END;
```

A PL/SQL code to call the procedure fire_employee

```
DECLARE
e number;
BEGIN
e:=&empno;
fire_employee(e);
END;
```

Local Function

```
DECLARE
        FUNCTION function_name
        [(argument IN data type, ...)] RETURN datatype {IS | AS}
        [local declarations]
        BEGIN
        PL/SQL subprogram body;
        [EXCEPTION
        exception handlers]
        END [name];
BEGIN
--PL/SQL code to call function
[EXCEPTION
--calling program exception handler]
END;
```

A PL/SQL code that calls a function to add two numbers.

```
DECLARE

        A NUMBER;
        B NUMBER;
        C NUMBER;

        FUNCTION ADDN ( A IN NUMBER, B IN NUMBER) RETURN
        NUMBER IS
        BEGIN
                C:=A+B;
                RETURN(C);
        END;

BEGIN

        A:=&FIRSTNUMBER;
        B:=&SECONDNUMBER;
        C:=ADDN(A,B);
        DBMS_OUTPUT.PUT_LINE('ADDITION IS'||C);

END;
```

Stored Function

```
CREATE OR REPLACE FUNCTION function_name
        (argument IN DATATYPE,...)
        RETURN datatype {IS | AS}
        [local declarations]

BEGIN
        PL/SQL subprogram body;

[EXCEPTION
        exception PL/SQL block;]

END;
```

A Stored function that accepts two numbers and return addition of passed values.

```
CREATE OR REPLACE FUNCTION ADDN( A IN NUMBER, B IN NUMBER)
      RETURN NUMBER IS
      C NUMBER;

BEGIN

      C:=A+B;
      RETURN(C);
END;
```

A PL/SQL code to call the function ADDN

```
DECLARE
      A NUMBER;
      B NUMBER;
      C NUMBER;
BEGIN
    A:=&FIRSTNUMBER;
    B:=&SECONDNUMBER;
     C:=ADDN(A,B);
    DBMS_OUTPUT.PUT_LINE('ADDITION IS'||C);
END;
```

A Stored function that accepts department number and return total salary of that department.

```
CREATE OR REPLCAE FUNCTION SALARY (dept NUMBER)
RETURN NUMBER IS
        s NUMBER;

BEGIN
        Select sum(sal) into s from emp where deptno=dept;
        RETURN(s);

END;
```

A PL/SQL code to call the function SALARY

```
DECLARE

      D NUMBER;
BEGIN

      D:=&DEPTNO;
      SAL:=SALARY(D);
      DBMS_OUTPUT.PUT_LINE('salary of department number' || d || 'is'
      || sal);

END;
```

Dropping a Procedure/ Function

Syntax:

- DROP PROCEDURE procedure_name;
- DROP FUNCTION function_name;

Examples:

- DROP PROCEDURE process;
- DROP FUNCTION addn;

Thanks

Lets Implement it in Lab Session