PL/SQL has two types of subprograms:

- PROCEDURES
- FUNCTIONS

A named PL/SQL block which contain set of statement to perform specific task is called stored Procedure.

OR

A named PL/SQL program which is stored or created under database that performs one or more actions is called as procedure an executable PL/SQL statement. It also known as a subprogram to perform a task or set of tasks. You can pass information or parameters to a procedure using IN, OUT and INOUT mode. Procedures and Functions are the subprograms which can be created and saved in the database as database objects.

A procedure should not return any value to calling area or object. Whenever we are use create or replace keyword in front of procedure those procedures automatically permantaly stored in database. These types of procedures are also called as stored procedures.

- It can be stored as precompiled object.
- It will be complied once and executed any number of times.
- Procedure provides reusability.
- Procedure can be easily enhansible for future requirement.
- Procedures are explicitly executed by user.

Procedures are two types:

- **Static Procedure:**
  It will not contain any arguments (parameters) variable and it won't take any values from the user. Always display same output.
- **Dynamic Procedure:**
  A procedure with parameters is known as dynamic procedure. It will be take runtime values from the user and display different output values from different user.

Every procedure having two parts:

- ***Procedure Specification:*** It contains the name of the procedure and the parameters or variables passed to the procedure.
- ***Procedure Body:*** It contains a declaration section, execution section and exception
  section similar to a general PL/SQL block. Here we are solving actual task.

Syntax:

Create [or replace] Procedure <proc_name> (parameter_name [mode]datatype)
IS / AS (Formal Parameters)
<Variable declaration, cursors, user-defined exceptions>;
BEGIN
<Executable statements>;
<Data Processing Statements>;

<Output Statements>;
EXCEPTION
<Exception statements>;
END <Procedure_name>;

Note: You can also end your procedure without procedure name just use end keyword for procedure close.

Subprograms have three parts:

- **The declarative part** contains declarations of types, cursors, constants, variables,
  exceptions, and nested subprograms. These objects are local and cease to exist when exited from the subprogram.
- **The executable part** contains statements that assign values, control execution, and manipulate ORACLE data.
- **The exception handling part** contains exception handlers, which deal with exceptions raised during exception.

How to execute a procedure

Method 1: execute/exec <procedure_name> (actual parameter);
Method 2: Using anonymous block

```
Begin
Procedure_name(actual parameter);
End;
/
```

Method 3: call Procedure_name(actual parameter);

Example: Write a pl/sql stored procedure for passing empno as a parameter display name of employee and his salary.

```
Create or replace procedure proc_ename(veno number)
Is
vename varchar2(10);
vsal number(10);
begin
select ename,sal into vename,vsal from emp where eno=veno;
dbms_output.put_line('Name of emp:'|| vename ||    'salary: '|| vsal);
end proc_ename;
/
```

Method 1:

```
set serveroutput on
exec proc_ename> (2);
```

Method 2: Using anonymous block a PL/SQL program to call procedure

```
set serveroutput on
        Begin
        proc_ename (2);
        End;
        /
```

Method 3: set serveroutput on
call proc_ename (2);

Note: set serveroutput on command for display result on the screen otherwise result can't be display.

All stored procedures information stored under user_source data dictionary.

```
sql>  desc user_source;
sql>  select text from user_source where NAME='proc_enmae';
```

- **Two types of Procedures:**
  - Static Procedure
  - Dynamic Procedure

1) Static Procedure

```
create procedure proc_ename
is
veno int;
vename varchar2(20);
begin
veno:=103;
select ename into vename from emp where eno=veno;
dbms_output.put_line('Name of emp number: '|| veno || 'is :' || vename);
end proc_ename;
/
```

2) Dynamic Procedure

Example: Write a pl/sql stored procedure for passing empno as a parameter display name of employee.

```
create or replace procedure proc_ename (veno int)
is
vename varchar2(20);
begin
select ename into vename from emp where eno=veno;
dbms_output.put_line('Name of emp number:    '|| veno ||   'is : '     || vename);
end proc_ename;
/
```

Example: Write a pl/sql stored procedure display area, radius using for loop.

First create a tble:
Create table circle (radius number, pi number(3,2),area number);

```
create or replace procedure proc_circle
is
a number;
pi constant number:=3.14;
begin
for r in 1..20
loop
a:=pi*r*r;
insert into circle values(r,pi,a);
end loop;
dbms_output.put_line('Insertion Completed');
end proc_circle;
/
```

Select * from circle --Run this query before procedure is executed.
After executing procedure records are inserted into circle table.

Conditional Statement using Procedure
Example: Write a pl/sql stored procedure for passing empno as a parameter display employee number, salary, bonus and his final salary.

```
create or replace procedure proc_salbonus(veno emp.eno%type)
is
vsal number;
b number;
fsal number;
begin
select sal into vsal from emp where eno=veno;
dbms_output.put_line('-----------------------------------------------');
dbms_output.put_line('Given emp number: '  || veno);
dbms_output.put_line('Salary :  ' || vsal);
dbms_output.put_line('-----------------------------------------------');
if (vsal>=5000) then
b:=0.20*vsal;
else
b:=0.10*vsal;
end if;
fsal:=b+vsal;
```

```
dbms_output.put_line('Bonus Amount: '  || b);
dbms_output.put_line(' Final Salary :  ' || fsal);
end proc_salbonus;
/
```

- **Parameter used in Procedures:**

Procedures are used to pass the values into procedure and also return values from the procedure. We must use two types of parameters.

- Formal Parameter
- Actual Parameter

- Formal Parameter: Formal parameters are defined in procedure specification. In formal parameter we are defining parameter name and mode of the parameter.

There are three types of modes supported by oracle.

- IN Mode
- OUT Mode
- INOUT Mode

Syntax: parameter_name [mode] datatype

Note: Whenever we are using procedures, functions, Cursor. We are not allowed to use data type size in formal parameter declaration.

- **IN Mode :**
  - By default procedure parameter having IN Mode. It takes input values at runtime, you specific this mode or not.
  - IN Mode used to pass the values into procedure body.
  - This Mode acts like a 'constant' in procedure body.
  - We can also pass default values using default or := operator.
  - It is a read-only variable inside the subprograms; their values cannot be changed inside the subprogram.

Example: Write a pl/sql stored procedure to insert a record into dept table using IN parameter.

```
Create or replace procedure proc_dept(vdno in number, vdname in
varchar2, vloc varchar2)
is
begin
insert into dept values(vdno, vdname, vloc);
dbms_output.put_line('Record inserted through Procedure');
end;
/
```

- **OUT Mode :**
  - You want return any values from procedure body then declare out parameter.
  - Parameter values return to calling area or object (program or function).

- OUT Mode acts like an uninitialized variable in procedure body.
- We must specify OUT keyword explicitly.
- It is a read-write variable inside the subprograms, their values can be changed inside the subprograms.

Syntax: parameter_name OUT datatype

Example:

```
create or replace procedure square(n in number,result OUT number)
as
  begin
     result:=n*n;
 end;

declare
 ans number;
begin
  square(12,ans);
  dbms_output.put_line('Square of number is '||ans);
end;
```

- **IN OUT Mode :**
  - Used to for both giving input and return output values from the subprograms.
  - You can change values within procedure. It accept runtime input values.
  - It internally acts like a constant, initialized variable.
  - We must specify IN OUT keyword explicitly.
  - It is a read-write variable inside the subprograms, their values can be changed inside the subprograms.

Syntax: parameter_name OUT datatype

Example: Write a pl/sql stored procedure to display factorial number given number using IN OUT parameter.

```
create or replace procedure FACT(N IN OUT NUMBER)
 IS
 F NUMBER:=1;
 BEGIN
 FOR I IN 1..N LOOP
 F:=F*I;
 END LOOP;
 N:=F;
 END;
 /
 DECLARE
 N NUMBER:=&N;
 BEGIN
 DBMS_OUTPUT.PUT_LINE('GIVEN VALUE:'||N);
```

```
FACT(N);
DBMS_OUTPUT.PUT_LINE('FACTORIAL:'||N);
END;
/
```

- **PL/SQL Drop Procedure**

Syntax: Drop Procedure <procedure_name>;
Example: Drop Procedure pro1;

- **PL/SQL Display information of Procedure**

Example: Desc user_procedures;
Select object_name from user_procedures;

Methods for Passing Parameters

Actual parameters can be passed in three ways –

- Positional notation
- Named notation
- Mixed notation

Positional Notation

In positional notation, you can call the procedure as –

findMin(a, b, c, d);

In positional notation, the first actual parameter is substituted for the first formal parameter; the second actual parameter is substituted for the second formal parameter, and so on. So, **a** is substituted for **x, b** is substituted for **y, c** is substituted for **z** and **d** is substituted for **m**.

Example sql> exec proc_dept(1, 'Sales', 'Pune');

Named Notation

In named notation, the actual parameter is associated with the formal parameter using the **arrow symbol (=>)**. The procedure call will be like the following –

findMin(x => a, y => b, z => c, m => d);

Example exec proc_dept(vdno=>2, vdname=>'Admin', vloc=>'Mumbai');

Mixed Notation

In mixed notation, you can mix both notations in procedure call; however, the positional notation should precede the named notation.
The following call is legal –

findMin(a, b, c, m => d);

However, this is not legal:

findMin(x => a, b, c, d);

Example sql> exec proc_dept(3, vdname=>'HR', vloc=>'Delhi');