CHAPTER – 3

STORED FUNCTION

Let's make coding fun!

STORED FUNCTION

A stored **function** (also called a user **function** or user-defined **function**) is a set of **PL/SQL** statements you can call by name. Stored **functions** are very similar to procedures, except that a **function** returns a value to the environment in which it is called. User **functions** can be used as part of a **SQL** expression. A standalone function is created using the **CREATE FUNCTION** statement.

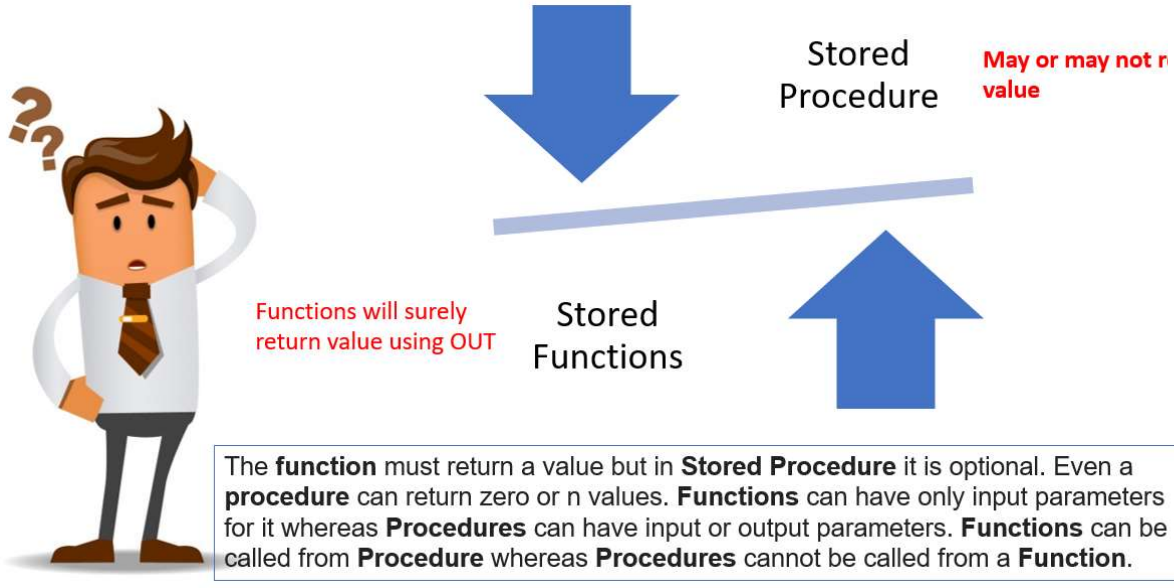
Stored Functions Syntax

```
CREATE OR REPLACE FUNCTIONS
<function_name>
(Argument {IN, OUT, IN/OUT }*
<Data type> )Return <var> IS,AS
<variable> declaration;
<constant> declarations;
BEGIN
<PL/SQL subprogram body>;
EXCEPTION PL/SQL BLOCK>;
Return <variable>
END <function_name>;
```

Output parameter is a **parameter** whose value **is** passed **out** of the **stored** procedure / **function** module, back to the calling PL/SQL block. An **OUT parameter** must be a variable, not a constant. It **can** be found only on the left-hand side of an assignment in the module. **PL/SQL functions can**'t able to **return multiple values**,it **can return** only **one** (single)**values** using OUT parameters in their arguments.

*
IN – parameter will accept a value from user
OUT – Parameter will return value to user

Where, *function-name* specifies the name of the function. [OR REPLACE] option allows the modification of an existing function. The optional parameter list contains name, mode and types of the parameters. IN represents the value that will be passed from outside and OUT represents the parameter that will be used to return a value outside of the procedure. The function must contain a **return** statement. The *RETURN* clause specifies the data type you are going to return from the function. *function-body* contains the executable part. The AS keyword is used instead of the IS keyword creating a standalone function.



EXAMPLE

The following example illustrates how to create and call a standalone function. This function returns the total number of CUSTOMERS in the customers table.

Select * from customers;

ID	NAME	AGE	ADDRESS	SALARY
I1	Mark	32	Sydney	2000
I2	Tom	25	Melbourne	1500
I3	Shane	23	Amsterdam	2050

```
CREATE OR REPLACE FUNCTION totalCustomers
RETURN number IS
   total number(2) := 0;
BEGIN
   SELECT count(*) into total
   FROM customers;
   RETURN total;
END;
```

/ When the above code is executed using the SQL prompt, it will produce the following result –

Function created.

Calling a Function

While creating a function, you give a definition of what the function has to do. To use a function, you will have to call that function to perform the defined task. When a program calls a function, the program control is transferred to the called function. A called function performs the defined task and when its return statement is executed or when the **last end statement** is reached, it returns the program control back to the main program. To call a function, you simply need to pass the required parameters along with the function name and if the function returns a value, then you can store the returned value. Following program calls the function **totalCustomers** from an anonymous block −

```
DECLARE
   c number(2);
BEGIN
   c := totalCustomers();
   dbms_output.put_line('Total no. of Customers: ' || c);
END;
/
```

When the above code is executed at the SQL prompt, it produces the following result −

```
Total no. of Customers: 6

PL/SQL procedure successfully completed.
```

Example

The following example demonstrates Declaring, Defining, and Invoking a Simple PL/SQL Function that computes and returns the maximum of two values.

```
DECLARE
   a number;
   b number;
   c number;
FUNCTION findMax(x IN number, y IN number)
RETURN number
```

```
IS
    z number;
BEGIN
   IF x > y THEN
      z:= x;
   ELSE
      Z:= y;
   END IF;
   RETURN z;
END;
BEGIN
   a:= 23;
   b:= 45;
   c := findMax(a, b);
   dbms_output.put_line(' Maximum of (23,45): ' || c);
END;  /
```

When the above code is executed at the SQL prompt, it produces the following result −Maximum of (23,45): 45

PL/SQL Recursive Functions

We have seen that a program or subprogram may call another subprogram. When a subprogram calls itself, it is referred to as a recursive call and the process is known as **recursion**. To illustrate the concept, let us calculate the factorial of a number. Factorial of a number n is defined as –

```
n! = n*(n-1)!

= n*(n-1)*(n-2)!
   ...
= n*(n-1)*(n-2)*(n-3)... 1
```

The following program calculates the factorial of a given number by calling itself recursively −

```
DECLARE
   num number;
   factorial number;
FUNCTION fact(x number)
```

```
RETURN number
IS
   f number;
BEGIN
   IF x=0 THEN
      f := 1;
   ELSE
      f := x * fact(x-1);
   END IF;
RETURN f;
END;
BEGIN
   num:= 6;
   factorial := fact(num);
   dbms_output.put_line(' Factorial '|| num || ' is ' || factorial);
END;  /
```

When the above code is executed at the SQL prompt, it produces the following result − Factorial 6 is 720

Stored Function Demonstration

```
create or replace FUNCTION FUNCTION2(
      col_a IN tablea.cola%TYPE)
  RETURN NUMBER
IS
  col_b TABLEA.colb%TYPE := 0;
BEGIN
  SELECT colb
  INTO col_b
  FROM tablea where cola ='ROW11';
  dbms_output.put_line('Function return --> '||col_b);

  RETURN col_b;
END FUNCTION2;
```

Table A

COLA	COLB
ROW11	20
ROW21	12

CONCLUSION

In this chapter, we have explained about Stored Functions syntax, example and demonstration of implementation of algorithms.