



PL/SQL – Procedures And Functions



Contents

Procedure

Procedure Parameter in PL/SQL

Methods for Passing Parameters

Functions

Difference between function and procedure

PROCEDURES

- A **procedure** is a group of **PL/SQL** statements that you can call by name.
- A procedure is a module performing one or more actions , it does not need to return any values.

Creating a Procedure

```
CREATE [OR REPLACE] PROCEDURE procedure_name
[(parameter_name [IN | OUT | IN OUT] type [, ...])]
{IS | AS}
BEGIN
    < procedure_body >
END
    procedure_name;
```

- *procedure-name* specifies the name of the procedure.
- [OR REPLACE] option allows modifying an existing procedure.
- IN represents that value will be passed from outside and OUT represents that this parameter will be used to return a value outside of the procedure.
- *procedure-body* contains the executable part.
- The AS keyword is used instead of the IS keyword for creating a standalone procedure.

Example

The following example creates a simple procedure that displays the string 'Hello World!' on the screen when executed.

```
CREATE OR REPLACE PROCEDURE greetings
AS
    BEGIN
        dbms_output.put_line('Hello World!');
    END;
```

Procedure created

Executing a Standalone Procedure

A standalone procedure can be called in two ways :

- Using the EXECUTE keyword
- Calling the name of the procedure from a PL/SQL block

Deleting a Standalone Procedure

Syntax:

```
DROP PROCEDURE procedure_name
```

Procedure Parameter in PL/SQL

1. IN

It is a read-only parameter. Inside the subprogram, an IN parameter acts like a constant. It cannot be assigned a value . You can pass a constant, expression as an IN parameter . You can also initialize it to a default value . **It is the default mode of parameter passing. Parameters are passed by reference.**

2. OUT

An OUT parameter returns a value to the calling program. Inside the subprogram, an OUT parameter acts like a variable. You can change its value and reference the value after assigning it. **The actual parameter must be variable and it is passed by value.**

3. IN OUT

An IN OUT parameter passes an initial value to a subprogram and returns an updated value to the caller. It can be assigned a value and its value can be read.

The actual parameter corresponding to an IN OUT formal parameter must be a variable, not a constant or an expression. Formal parameter must be assigned a value. **Actual parameter is passed by value.**

IN & OUT Example

Input

```
DECLARE
    a number;
    b number;
    c number;
PROCEDURE findMin(x IN number, y IN number, z OUT number)IS
BEGIN
    IF x < y THEN
        z:= x;
    ELSE
        z:= y;
    END IF;
END;
BEGIN
    a:= 23;
    b:= 45;
    findMin(a, b, c);
    dbms_output.put_line(' Minimum of (23, 45) : ' || c);
END;
```

Output

Minimum of (23, 45) : 23

IN OUT Example

```
DECLARE
    a number;
PROCEDURE squareNum(x IN OUT number) IS
BEGIN
    x := x * x;
END;
BEGIN
    a:= 23;
    squareNum(a);
    dbms_output.put_line(' Square of (23): ' || a);
END;
```

Output

Square of (23): 529

Methods for Passing Parameters

Actual parameters could be passed in three ways:

1. Positional notation
2. Named notation
3. Mixed notation

POSITIONAL NOTATION

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*.

NAMED NOTATION

In named notation, the actual parameter is associated with the formal parameter using the arrow symbol (\Rightarrow). So the procedure call would look like:

```
findMin(x $\Rightarrow$ a, y $\Rightarrow$ b, z $\Rightarrow$ c, m $\Rightarrow$ d);
```

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);
```

But this is not legal:

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

Functions

- Functions are a type of stored code and are very similar to procedures.
- The significant difference is that a function is a PL/SQL block that *returns* a single value.
- Functions can accept one, many, or no parameters, but a function must have a return clause in the executable section of the function.
- The datatype of the return value must be declared in the header of the function.
- A function is not a stand-alone executable in the way that a procedure is: It must be used in some context. You can think of it as a sentence fragment.
- A function has output that needs to be assigned to a variable, or it can be used in a SELECT statement.

Creating a Function

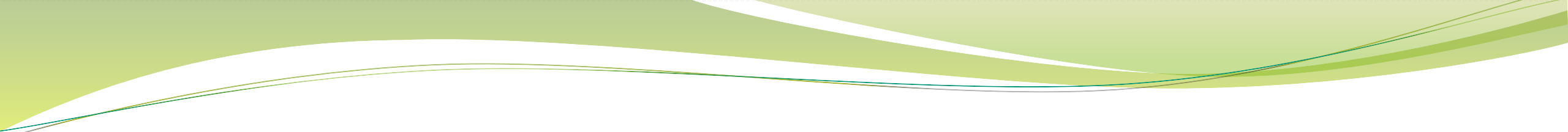
```
CREATE [OR REPLACE] FUNCTION function_name  
[(parameter_name [IN | OUT | IN OUT] type [, ...])]  
RETURN  
    return_datatype {IS | AS} BEGIN < function_body >  
END  
[function_name];
```


- *function-name* specifies the name of the function.
- [OR REPLACE] option allows modifying an existing function.
- The optional parameter list contains name, mode and types of the parameters. IN represents that value will be passed from outside and OUT represents that this parameter will be used to return a value outside of the procedure.
- The function must contain a **return** statement.
- *RETURN* clause specifies that 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 for creating a standalone function.

The following example illustrates creating and calling a standalone function. This function returns the total number of CUSTOMERS in the customers table. We will use the CUSTOMERS table, which we had created in PL/SQL Variables chapter:

```
Select * from customers;
```

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	MP	4500.00



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

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

Function created

Example:

The following is one more example which 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;
```

Maximum of (23,45): 45

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, program control is transferred to the called function.
- A called function performs defined task and when its return statement is executed or when its last end statement is reached, it returns program control back to the main program.
- To call a function you simply need to pass the required parameters along with function name and if function returns a value then you can store returned value. Following program calls the function `totalCustomers` from an anonymous block:

Example

```
DECLARE
```

```
c number(2);
```

```
BEGIN
```

```
    c := totalCustomers();
```

```
    dbms_output.put_line('Total no. of Customers: ' || c);
```

```
END;
```

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:

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

$$= n * (n-1) * (n-2)!$$

...

$$= n * (n-1) * (n-2) * (n-3) \dots 1$$

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 SQL prompt, it produces the following result:

Factorial 6 is 720

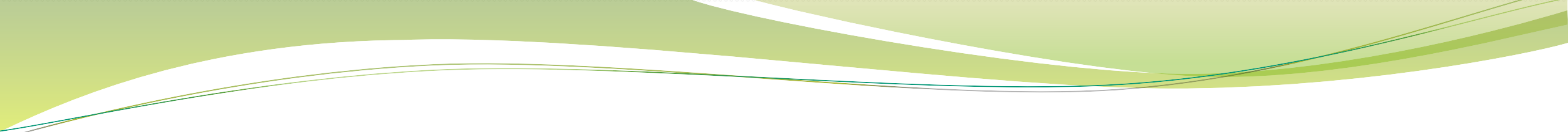
PL/SQL procedure successfully completed.

Difference b/w procedure and function

- A procedure does not have a return value, whereas a function has.

```
CREATE OR REPLACE PROCEDURE my_proc  
(p_name IN VARCHAR2 := 'John') as begin ... End
```

```
CREATE OR REPLACE FUNCTION my_func  
(p_name IN VARCHAR2 := 'John') return varchar2 as begin ... End
```

- 
1. Procedure can perform one or more tasks whereas a function performs a specific task.
 2. Procedure may or may not return a value whereas a function should return one value.
 3. We can call functions in a select statement whereas a procedure we can't.
 4. We can call a function within a procedure but we cannot call a procedure within a function.
 5. A FUNCTION must be part of an executable statement, as it cannot be executed independently whereas a procedure represents an independent executable statement.