|  |  |  |
| --- | --- | --- |
| **Ex.No. 12** | **PL/SQL FUNCTIONS** | **Date :** |

Creating a Function

A standalone function is created using the **CREATE FUNCTION** statement. The simplified syntax for the **CREATE OR REPLACE PROCEDURE** statement is as follows −

CREATE [OR REPLACE] FUNCTION function\_name

[(parameter\_name [IN | OUT | IN OUT] type [, ...])]

RETURN return\_datatype

{IS | AS}

BEGIN

< function\_body >

END [function\_name];

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 for 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.

We will use the CUSTOMERS table, which we had created in the [PL/SQL Variables](https://www.tutorialspoint.com/plsql/plsql_variable_types.htm) 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 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 procedure successfully completed.

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

PL/SQL procedure successfully completed.

## PL/SQL Drop Function Syntax

To delete an existing procedure, use the following syntax:

### Syntax

DROP FUNCTION function\_name;

Questions

1. Write a pl/sql function recursive code for finding factorial of a number.
2. Write a pl/sql function for finding a number is a prime number.
3. Write a pl/sql function to retrieve the count of students from ‘CSE’ department from the table Student (Sno, sname, dept, cgpa).
4. Write a pl/sql function to retrieve the maximum CGPA of the student from the table Student (Sno, sname, dept, cgpa).
5. Write a simple PL/SQL Function that computes and returns the average of two numbers.