GALGOTIAS UNIVERSITY

**School of Computing Science and Engineering**

Program: BCA - IOP

Course Code: BCAS3031

Course Name: PL/SQL & Cursors and Triggers

Dr. T. Poongodi

Associate Professor

**PL/SQL Stored Procedure & Functions with Examples**

Procedures and Functions are the subprograms which can be created and saved in the database as database objects. They can be called or referred inside the other blocks also.

**Terminologies in PL/SQL Subprograms**

**Parameter:**

The parameter is variable or placeholder of any valid PL/SQL datatype through which the PL/SQL subprogram exchange the values with the main code. This parameter allows to give input to the subprograms and to extract from these subprograms.

- Parameters should be defined along with the subprograms at the time of creation.
- Parameters are included in the calling statement of these subprograms to interact the values with the subprograms.
- Datatype of the parameter in the subprogram and the calling statement should be same.

Based on their purpose parameters are classified as

1. IN Parameter

2. OUT Parameter

3. IN OUT Parameter

IN Parameter:

- This parameter is used for giving input to the subprograms.
- It is a read-only variable inside the subprograms. Their values cannot be changed inside the subprogram.
- In the calling statement, these parameters can be a variable or a literal value or an expression, for example, it could be the arithmetic expression like '5*8' or 'a/b' where 'a' and 'b' are variables.
- By default, the parameters are of IN type.

OUT Parameter:

- This parameter is used for getting output from the subprograms.
- It is a read-write variable inside the subprograms. Their values can be changed inside the subprograms.
- In the calling statement, these parameters should always be a variable to hold the value from the current subprograms.

IN OUT Parameter:

- This parameter is used for both giving input and for getting output from the subprograms.
- It is a read-write variable inside the subprograms. Their values can be changed inside the subprograms.
- In the calling statement, these parameters should always be a variable to hold the value from the subprograms.

These parameter type should be mentioned at the time of creating the subprograms.

# RETURN

- RETURN is the keyword that instructs the compiler to switch the control from the subprogram to the calling statement.
- In subprogram RETURN simply means that the control needs to exit from the subprogram.
- Once the controller finds RETURN keyword in the subprogram, the code after this will be skipped.

- Normally, parent or main block will call the subprograms, and then the control will shift from those parent block to the called subprograms.
- RETURN in the subprogram will return the control back to their parent block.
- In the case of functions RETURN statement also returns the value.
- The datatype of this value is always mentioned at the time of function declaration.
- The datatype can be of any valid PL/SQL data type.

# What is Procedure in PL/SQL?

- A Procedure is a subprogram unit that consists of a group of PL/SQL statements.
- Each procedure in Oracle has its own unique name by which it can be referred.
- This subprogram unit is stored as a database object.

Note: Subprogram is nothing but a procedure, and it needs to be created manually as per the requirement. Once created they will be stored as database objects.

- Procedures are standalone blocks of a program that can be stored in the database.
- Call to these procedures can be made by referring to their name, to execute the PL/SQL statements.
- It is mainly used to execute a process in PL/SQL.
- It can have nested blocks, or it can be defined and nested inside the other blocks or packages.
- It contains declaration part (optional), execution part, exception handling part (optional).
- The values can be passed into the procedure or fetched from the procedure through parameters.
- These parameters should be included in the calling statement.
- Procedure can have a RETURN statement to return the control to the calling block, but it cannot return any values through the RETURN statement.
- Procedures cannot be called directly from SELECT statements. They can be called from another block or through EXEC keyword.

Syntax:

```
CREATE OR REPLACE PROCEDURE
<procedure_name>
        (
        <parameterl IN/OUT <datatype>
        ..
        .
        )
[ IS | AS ]
        <declaration_part>
BEGIN
        <execution part>
EXCEPTION
        <exception handling part>
END;
```

- CREATE PROCEDURE instructs the compiler to create new procedure. Keyword 'OR REPLACE' instructs the compile to replace the existing procedure (if any) with the current one.
- Procedure name should be unique.
- Keyword 'IS' will be used, when the procedure is nested into some other blocks. If the procedure is standalone then 'AS' will be used. Other than this coding standard, both have the same meaning.

# Creating Procedure and calling it using EXEC

In this example, we are going to create a procedure that takes the name as input and prints the welcome message as output.

We are going to use EXEC command to call procedure.

```
CREATE OR REPLACE PROCEDURE welcome_msg
(p_name IN VARCHAR2)
IS
BEGIN
dbms_output.put_line (‘Welcome '|| p_name);
END;
/
EXEC welcome_msg (‘Rahul’);
```

Code Explanation:

- Code line 1: Creating the procedure with name 'welcome_msg' and with one parameter 'p_name' of 'IN' type.
- Code line 4: Printing the welcome message by concatenating the input name.
- Procedure is compiled successfully.
- Code line 7: Calling the procedure using EXEC command with the parameter 'Rahul'. Procedure is executed, and the message is printed out as "Welcome Rahul".

# What is Function?

Functions is a standalone PL/SQL subprogram. Like PL/SQL procedure, functions have a unique name by which it can be referred.

These are stored as PL/SQL database objects.

.Functions are a standalone block that is mainly used for calculation purpose.

- Function use RETURN keyword to return the value, and the datatype of this is defined at the time of creation.
- A Function should either return a value or raise the exception, i.e. return is mandatory in functions.
- Function with no DML statements can be directly called in SELECT query whereas the function with DML operation can only be called from other PL/SQL blocks.
- It can have nested blocks, or it can be defined and nested inside the other blocks or packages.
- It contains declaration part (optional), execution part, exception handling part (optional).
- The values can be passed into the function or fetched from the procedure through the parameters.
- These parameters should be included in the calling statement.
- Function can also return the value through OUT parameters other than using RETURN.
- Since it will always return the value, in calling statement it always accompanies with assignment operator to populate the variables.

Syntax

```
CREATE OR REPLACE FUNCTION
<procedure_name>
(
<parameterl IN/OUT <datatype>
)
RETURN <datatype>
[ IS | AS ]
<declaration_part>
BEGIN
<execution part>
EXCEPTION
<exception handling part>
END;
```

- CREATE FUNCTION instructs the compiler to create a new function. Keyword 'OR REPLACE' instructs the compiler to replace the existing function (if any) with the current one.
- The Function name should be unique.
- RETURN datatype should be mentioned.
- Keyword 'IS' will be used, when the procedure is nested into some other blocks. If the procedure is standalone then 'AS' will be used. Other than this coding standard, both have the same meaning.

```
CREATE OR REPLACE FUNCTION welcome_msg_func( p_name IN
VARCHAR2) RETURN VARCHAR2
IS
BEGIN
RETURN (‘Welcome ‘|| p_name);
END;
/
DECLARE
lv_msg VARCHAR2(250);
BEGIN
lv_msg := welcome_msg_func (‘Rahul’);
dbms_output.put_line(lv_msg);
END;
SELECT welcome_msg_func(‘Rahul:) FROM tablename;
```

Code Explanation:

- Code line 1: Creating the function with name 'welcome_msg_func' and with one parameter 'p_name' of 'IN' type.
- Code line 2: declaring the return type as VARCHAR2
- Code line 5: Returning the concatenated value 'Welcome' and the parameter value.
- Code line 8: Anonymous block to call the above function.
- Code line 9: Declaring the variable with datatype same as the return datatype of the function.
- Code line 11: Calling the function and populating the return value to the variable 'lv_msg'.
- Code line 12: Printing the variable value. The output you will get here is "Welcome Rahul"
- Code line 14: Calling the same function through SELECT statement. The return value is directed to the standard output directly.

# Similarities between Procedure and Function

- Both can be called from other PL/SQL blocks.
- If the exception raised in the subprogram is not handled in the subprogram exception handling section, then it will propagate to the calling block.
- Both can have as many parameters as required.
- Both are treated as database objects in PL/SQL.

| Procedure | Function |
|---|---|
| • Used mainly to execute certain process | • Used mainly to perform some calculation |
| • Cannot call in SELECT statement | • A Function that contains no DML statements can be called in SELECT statement |
| • Use OUT parameter to return the value | • Use RETURN to return the value |
| • It is not mandatory to return the value | • It is mandatory to return the value |
| • RETURN will simply exit the control from subprogram. | • RETURN will exit the control from subprogram and also returns the value |
| • Return datatype will not be specified at the time of creation | • Return datatype is mandatory at the time of creation |

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

```
Procedure Created.
```

Run SQL Command Line

```
SQL*Plus: Release 11.2.0.2.0 Production on Sun Aug 16 09:12:48 2020

Copyright (c) 1982, 2014, Oracle.  All rights reserved.

SQL> connect
Enter user-name: system
Enter password:
Connected.
SQL> set serveroutput on;
SQL> CREATE OR REPLACE PROCEDURE sample
  2  AS
  3  BEGIN
  4     dbms_output.put_line('Hello World');
  5  END;
  6  /

Procedure created.

SQL>
SQL> EXECUTE sample;
Hello World

PL/SQL procedure successfully completed.

SQL> BEGIN
  2     sample;
  3  END;
  4  /
Hello World

PL/SQL procedure successfully completed.

SQL> DROP PROCEDURE sample;

Procedure dropped.

SQL>
```

Run SQL Command Line

```
SQL> DECLARE
  2     a number;
  3     b number;
  4     c number;
  5  PROCEDURE findmin(x IN number,y IN number, z OUT number) IS
  6  BEGIN
  7     IF x<y THEN
  8             z:=x;
  9     ELSE
 10             z:=y;
 11     END IF;
 12  END;
 13  BEGIN
 14     a:=25;
 15     b:=43;
 16     findmin(a,b,c);
 17     dbms_output.put_line('The minimum value:'||c);
 18  END;
 19  /
The minimum value:25

PL/SQL procedure successfully completed.

SQL>
```

Run SQL Command Line

```
SQL> DECLARE
  2     a number;
  3  PROCEDURE squarenum(p IN OUT number) IS
  4  BEGIN
  5     p:=p*p;
  6  END;
  7  BEGIN
  8     a:=8;
  9     squarenum(a);
 10     dbms_output.put_line('Square of 8 is' ||a);
 11  END;
 12  /
Square of 8 is64

PL/SQL procedure successfully completed.

SQL>
```

To create and call a standalone function

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;
/
Function created.
```

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

Output:

```
Total no. of Customers: 6

PL/SQL procedure successfully completed.
```

Run SQL Command Line

```
SQL> DECLARE
  2     a number;
  3     b number;
  4     c number;
  5  FUNCTION findMax(x IN number, y IN number)
  6  RETURN number
  7  IS
  8      z number;
  9  BEGIN
 10     IF x > y THEN
 11        z:= x;
 12     ELSE
 13        Z:= y;
 14     END IF;
 15     RETURN z;
 16  END;
 17  BEGIN
 18     a:= 23;
 19     b:= 45;
 20     c := findMax(a, b);
 21     dbms_output.put_line(' Maximum of (23,45): ' || c);
 22  END;
 23  /
Maximum of (23,45): 45

PL/SQL procedure successfully completed.

SQL>
```

Run SQL Command Line

```
PL/SQL procedure successfully completed.

SQL> DECLARE
  2     num number;
  3     factorial number;
  4
  5  FUNCTION fact(x number)
  6  RETURN number
  7  IS
  8     f number;
  9  BEGIN
 10     IF x=0 THEN
 11        f := 1;
 12     ELSE
 13        f := x * fact(x-1);
 14     END IF;
 15  RETURN f;
 16  END;
 17
 18  BEGIN
 19     num:= 6;
 20     factorial := fact(num);
 21     dbms_output.put_line(' Factorial '|| num || ' is ' || factorial);
 22  END;
 23  /
Factorial 6 is 720

PL/SQL procedure successfully completed.

SQL>
```


Thank You