SILVER OAK UNIVERSITY
EDUCATION TO INNOVATION

Silver Oak Group of Institutes
Silver Oak College of Computer Application

UNIT 2: Function and Stored Procedure

TOPICS:

❖ Application Development using Procedural

PL/SQL procedures are database objects that contain PL/SQL procedural logic and SQL statements that can be invoked in contexts where the CALL statement or procedure references are valid. PL/SQL procedures are created by executing the PL/SQL CREATE PROCEDURE statement.

❖ Overview of Function and Procedure

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. Below are some of the characteristics of functions.

- 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.
- A PLSQL 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 to create a function:

```
CREATE [OR REPLACE] FUNCTION function_name [parameters]
[(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.

PL/SQL Function Example

Let's see a simple example to **create a function**.

```
create or replace function adder(n1 in number, n2 in number)
return number
is
n3 number(8);
begin
n3 :=n1+n2;
return n3;
end;
```

Now write another program to **call the function**.

```
DECLARE
  n3 number(2);
BEGIN
  n3 := adder(11,22);
  dbms_output.put_line('Addition is: ' || n3);
END;
```

Another PL/SQL Function Example

Let's take an example to demonstrate Declaring, Defining and Invoking a simple PL/SQL function which will compute and return 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;
```

Output:

```
Maximum of (23,45): 45
Statement processed.
```

PL/SQL function example using table

Let's take a customer table. This example illustrates creating and calling a standalone function. This function will return the total number of CUSTOMERS in the customers table.

Customers			
Id	**Name**	**Department**	**Salary**
1	Alex	web developer	35000
2	Ricky	program developer	45000
3	Mohan	web designer	35000
4	Dilshad	database manager	44000

Create Function:

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

Calling PL/SQL Function:

While creating a function, you have to 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. Once the function is called, the program control is transferred to the called function.

After the successful completion of the defined task, the call function returns program control back to the main program.

To call a function you have 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 total Customers from an anonymous block:

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

PL/SQL Drop Function

Syntax for removing your created function:

```
DROP FUNCTION function_name;
```

PL/SQL Procedure

The PL/SQL stored procedure or simply a procedure is a PL/SQL block which performs one or more specific tasks. It is just like procedures in other programming languages.

The procedure contains a header and a body.

- **Header:** The header contains the name of the procedure and the parameters or variables passed to the procedure.
- **Body:** The body contains a declaration section, execution section and exception section similar to a general PL/SQL block.

How to pass parameters in procedure:

When you want to create a procedure or function, you have to define parameters. There is three ways to pass parameters in procedure:

1. **IN parameters:** The IN parameter can be referenced by the procedure or function. The value of the parameter cannot be overwritten by the procedure or the function.
2. **OUT parameters:** The OUT parameter cannot be referenced by the procedure or function, but the value of the parameter can be overwritten by the procedure or function.
3. **INOUT parameters:** The INOUT parameter can be referenced by the procedure or function and the value of the parameter can be overwritten by the procedure or function.

A procedure may or may not return any value.

PL/SQL Create Procedure

Syntax for creating procedure:

```
CREATE [OR REPLACE] PROCEDURE procedure_name
    [ (parameter [,parameter]) ]
IS
    [declaration_section]
BEGIN
    executable_section
[EXCEPTION
    exception_section]
END [procedure_name];
```

Create procedure example

In this example, we are going to insert record in user table. So you need to create user table first.

Table creation:

```
create table user(id number(10) primary key,name varchar2(100));
```

Now write the procedure code to insert record in user table.

Procedure Code:

```
create or replace procedure "INSERTUSER"
(id IN NUMBER,
name IN VARCHAR2)
is
begin
insert into user values(id,name);
end;
1.   /
```

Output:

```
Procedure created.
```

PL/SQL program to call procedure

Let's see the code to call above created procedure.

```
BEGIN
  insertuser(101,'xyz');
  dbms_output.put_line('record inserted successfully');
END;
1.   DROP PROCEDURE procedure_name;
```

Example of drop procedure

```
DROP PROCEDURE pro1;
```

❖ Stored Procedure

A stored procedure in PL/SQL is nothing but a series of declarative SQL statements which can be stored in the database catalogue. A procedure can be thought of as a function or a method. They can be invoked through triggers, other procedures, or applications on Java, PHP etc. All the statements of a block are passed to Oracle engine all at once which increases processing speed and decreases the traffic.

Advantages:

- They result in performance improvement of the application. If a procedure is being called frequently in an application in a single connection, then the compiled version of the procedure is delivered.
- They reduce the traffic between the database and the application, since the lengthy statements are already fed into the database and need not be sent again and again via the application.
- They add to code reusability, similar to how functions and methods work in other languages such as C/C++ and Java.

Disadvantages:

- Stored procedures can cause a lot of memory usage. The database administrator should decide an upper bound as to how many stored procedures are feasible for a particular application.
- MySQL does not provide the functionality of debugging the stored procedures.

Creating a Procedure

A procedure is created with the **CREATE OR REPLACE PROCEDURE** statement. The simplified syntax for the CREATE OR REPLACE PROCEDURE statement is as follows −

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

Where,

- *procedure-name* specifies the name of the procedure.
- [OR REPLACE] option allows the modification of an existing procedure.
- 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.
- *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;
```

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

```
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

The above procedure named **'greetings'** can be called with the EXECUTE keyword as −

```
EXECUTE greetings;
```

The above call will display −

```
Hello World

PL/SQL procedure successfully completed.
```

The procedure can also be called from another PL/SQL block −

```
BEGIN
   greetings;
END;
```

The above call will display −

```
Hello World
```

Basic difference between Procedure and Function

S.No	PROCEDURE	FUNCTION
1	Used mainly to execute certain business logic with DML and DRL statements	Used mainly to perform some computational process and returning the result of that process.
2	Procedure can return zero or more values as output.	Function can return only single value as output.
3	Procedure cannot call with select statement, but can call from a block or from a procedure.	Function can call with select statement, if function does not contain any DML statements and DDL statements. function with DML and DDL statements can call with select statement with some special cases (using Pragma autonomous transaction)
4	OUT keyword is used to return a value from procedure.	RETURN keyword is used to return a value from a function.
5	It is not mandatory to return the value.	It is mandatory to return the value.
6	RETURN will simply exit the control from subprogram.	RETURN will exit the control from subprogram and also returns the value.
7	Return datatype will not be specified at the time of creation.	Return datatype is mandatory at the time of creation.