

Oracle Basics (PL/SQL)

Lesson 03 Procedures and
Functions

Lesson Objectives

- To understand the following topics:
 - Subprograms in PL/SQL
 - Anonymous blocks versus Stored Subprograms
 - Procedure
 - Subprogram Parameter modes
 - Functions



3.1: Subprograms in PL/SQL

Introduction

- A subprogram is a named block of PL/SQL
- There are two types of subprograms in PL/SQL, namely: Procedures and Functions
- Each subprogram has:
 - A declarative part
 - An executable part or body, and
 - An exception handling part (which is optional)
- A function is used to perform an action and return a single value



Copyright © Capgemini 2015. All Rights Reserved 3

Subprograms in PL/SQL:

- The subprograms are compiled and stored in the Oracle database as “stored programs”, and can be invoked whenever required. As the subprograms are stored in a compiled form, when called they only need to be executed. Hence this arrangement saves time needed for compilation.
- When a client executes a procedure or function, the processing is done in the server. This reduces the network traffic.
- Subprograms provide the following advantages:
 - They allow you to write a PL/SQL program that meets our need.
 - They allow you to break the program into manageable modules.
 - They provide reusability and maintainability for the code.

3.2: Anonymous Blocks & Stored Subprograms

Anonymous Blocks & Stored Subprograms Comparison

Anonymous Blocks	Stored Subprograms/Named Blocks
1. Anonymous Blocks do not have names.	1. Stored subprograms are named PL/SQL blocks.
2. They are interactively executed. The block needs to be compiled every time it is run.	2. They are compiled at the time of creation and stored in the database itself. Source code is also stored in the database.
3. Only the user who created the block can use the block.	3. Necessary privileges are required to execute the block.

3.3: Procedures

Procedures

- A procedure is used to perform an action.
- It is illegal to constrain datatypes.
- Syntax:

```
CREATE PROCEDURE Proc_Name
  (Parameter {IN | OUT | IN OUT} datatype := value,...) AS

  Variable_Declaration ;
  Cursor_Declaration ;
  Exception_Declaration ;
BEGIN
  PL/SQL_Statements ;
EXCEPTION
  Exception_Definition ;
END Proc_Name ;
```



Copyright © Capgemini 2015. All Rights Reserved 5

Procedures:

- A procedure is a subprogram used to perform a specific action.
- A procedure contains two parts:
 - the specification, and
 - the body
- The procedure specification begins with CREATE and ends with procedure name or parameters list. Procedures that do not take parameters are written without a parenthesis.
- The procedure body starts after the keyword IS or AS and ends with keyword END.

contd.

3.3: Procedures

Subprogram Parameter Modes

IN	OUT	IN OUT
The default	Must be specified	Must be specified
Used to pass values to the procedure.	Used to return values to the caller.	Used to pass initial values to the procedure and return updated values to the caller.
Formal parameter acts like a constant.	Formal parameter acts like an uninitialized variable.	Formal parameter acts like an uninitialized variable.
Formal parameter cannot be assigned a value.	Formal parameter cannot be used in an expression, but should be assigned a value.	Formal parameter should be assigned a value.
Actual parameter can be a constant, literal, initialized variable, or expression.	Actual parameter must be a variable.	Actual parameter must be a variable.
Actual parameter is passed by reference (a pointer to the value is passed in).	Actual parameter is passed by value (a copy of the value is passed out) unless NOCOPY is specified.	Actual parameter is passed by value (a copy of the value is passed in and out) unless NOCOPY is specified.



Copyright © Capgemini 2015. All Rights Reserved 5

Subprogram Parameter Modes:

- You use “parameter modes” to define the behavior of “formal parameters”. The three parameter modes are IN (the default), OUT, and INOUT. The characteristics of the three modes are shown in the slide.
- Any parameter mode can be used with any subprogram.
- Avoid using the OUT and INOUT modes with functions.
- To have a function return multiple values is a poor programming practice. Besides functions should be free from side effects, which change the values of variables that are not local to the subprogram.
- Example1:

```

CREATE PROCEDURE split_name
(
    phrase IN VARCHAR2, first OUT VARCHAR2, last
    OUT VARCHAR2
)
IS
    first := SUBSTR(phrase, 1, INSTR(phrase, ' ')-1);
    last := SUBSTR(phrase, INSTR(phrase, ' ')+1);
    IF first = 'John' THEN
        DBMS_OUTPUT.PUT_LINE('That is a common first
name.');

```

Subprogram Parameter Modes (contd.):**Examples:**

Example 2:

```
SQL > SET SERVEROUTPUT ON
SQL > CREATE OR REPLACE PROCEDURE PROC1 AS
2  BEGIN
3      DBMS_OUTPUT.PUT_LINE('Hello from procedure ...');
4  END;
5  /

Procedure created.
SQL > EXECUTE PROC1
Hello from procedure ...
PL/SQL procedure successfully created.
```

Example 3:

```
SQL > CREATE OR REPLACE      PROCEDURE PROC2
2  (N1 IN NUMBER, N2 IN NUMBER, TOT OUT NUMBER) IS
3  BEGIN
4      TOT := N1 + N2;
5  END;
6  /
```

Procedure created.

```
SQL > VARIABLE T NUMBER
SQL > EXEC PROC2(33, 66, :T)
```

PL/SQL procedure successfully completed.

SQL > PRINT T

```
          T
-----
          99
```

3.3: Procedures

Example on Procedures

■ Example 1:

```
CREATE OR REPLACE PROCEDURE Raise_Salary
(s_no IN number, raise_sal IN number) IS
  v_cur_salary number;
  missing_salary exception;
BEGIN
  SELECT staff_sal INTO v_cur_salary FROM staff_master
  WHERE staff_code=s_no;
  IF v_cur_salary IS NULL THEN
    RAISE missing_salary;
  END IF ;
  UPDATE staff_master SET staff_sal = v_cur_salary + raise_sal
  WHERE staff_code = s_no ;
EXCEPTION
  WHEN missing_salary THEN
    INSERT into emp_audit VALUES( sno, 'salary is missing');
END raise_salary;
```



Copyright © Capgemini 2015. All Rights Reserved 5

The procedure example on the slide modifies the salary of staff member. It also handles exceptions appropriately. In addition to the above shown exception you can also handle "NO_DATA_FOUND" exception. The procedure accepts two parameters which is the staff_code and amount that has to be given as raise to the staff member.

3.3: Procedures

Example on Procedures

▪ Example 2:

```
CREATE OR REPLACE PROCEDURE
  Get_Details(s_code IN number,
  s_name OUT varchar2,s_sal OUT number ) IS
BEGIN
  SELECT staff_name, staff_sal INTO s_name, s_sal
  FROM staff_master WHERE staff_code=s_code;
EXCEPTION
  WHEN no_data_found THEN
    INSERT into auditstaff
    VALUES( 'No employee with id ' || s_code);
    s_name := null;
    s_sal := null;
END get_details ;
```



Copyright © Capgemini 2015. All Rights Reserved 9

The procedure on the slide accept three parameters, one is IN mode and other two are OUT mode. The procedure retrieves the name and salary of the staff member based on the staff_code passed to the procedure. The S_NAME and S_SAL are the OUT parameters that will return the values to the calling environment

3.3: Procedures

Executing a Procedure

- Executing the Procedure from SQL*PLUS environment,
 - Create a bind variables salary and name SQLPLUS by using VARIABLE command as follows:

```
variable salary number  
variable name varchar2(20)
```

- Execute the procedure with EXECUTE command

```
EXECUTE Get_Details(100003,:Salary, :Name)
```

- After execution, use SQL*PLUS PRINT command to view results.

```
print salary  
print name
```



Copyright © Capgemini 2015. All Rights Reserved 10

Procedures can be executed through command line as shown on the slide or can be called from other procedures/functions/Anonymous PL/SQL blocks.

On the slide the first snippet declares two variables viz. salary and name. The second snippet calls the procedure and passes the actual parameters. The first is a literal string and the next two parameters are empty variables which will be assigned with values within the procedure.

Calling the procedure from an anonymous PL/SQL block

```
DECLARE  
    s_no number(10):=&sno;  
    sname varchar2(10);  
    sal number(10,2);  
BEGIN  
    Get_Details(s_no,sname,sal);  
    dbms_output.put_line('Name:'||sname||'Salary'||sal);  
END;
```



A bind variable is a variable that you declare in a host environment and then use to pass runtime values.

These values can be character or numeric. You can pass these values either in or out of one or more PL/SQL programs, such as packages, procedures, or functions.

To declare a bind variable in the SQL*Plus environment, you use the command `VARIABLE`.

For example,

```
VARIABLE salary NUMBER
```

Upon declaration, the **bind variables** now become **host** to that environment, and you can now use these variables within your PL/SQL programs, such as packages, procedures, or functions.

To reference host variables, you must add a prefix to the reference with a colon (:) to distinguish the host variables from declared PL/SQL variables.

example

```
EXECUTE Get_Details(100003,:salary, :name)
```

Parameter default values:

- Like variable declarations, the formal parameters to a procedure or function can have default values.
- If a parameter has default values, it does not have to be passed from the calling environment.
 - If it is passed, actual parameter will be used instead of default.
- Only IN parameters can have default values.

```
PROCEDURE Create_Dept( New_Deptno IN NUMBER,  
    New_Dname IN VARCHAR2 DEFAULT 'TEMP') IS  
BEGIN  
    INSERT INTO department_master  
        VALUES ( New_Deptno, New_Dname, New_Loc) ;  
END ;
```

```
BEGIN  
    Create_Dept( 50);  
    -- Actual call will be Create_Dept ( 50, 'TEMP',  
    'TEMP')  
  
    Create_Dept ( 50, 'FINANCE');  
    -- Actual call will be Create_Dept ( 50, 'FINANCE'  
    , 'TEMP')  
  
    Create_Dept( 50, 'FINANCE', 'BOMBAY') ;  
    -- Actual call will be Create_Dept(50, 'FINANCE',  
    'BOMBAY' )  
  
END;
```

Procedures (contd.):**Using Positional, Named, or Mixed Notation for Subprogram Parameters:**

- When calling a subprogram, you can write the actual parameters by using either Positional notation, Named notation, or Mixed notation.
 - **Positional notation:** You specify the same parameters in the same order as they are declared in the procedure. This notation is compact, but if you specify the parameters (especially literals) in the wrong order, the bug can be hard to detect. You must change your code if the procedure's parameter list changes.
 - **Named notation:** You specify the name of each parameter along with its value. An arrow (=>) serves as the “association operator”. The order of the parameters is not significant.
 - **Mixed notation:** You specify the first parameters with “Positional notation”, and then switch to “Named notation” for the last parameters. You can use this notation to call procedures that have some “required parameters”, followed by some “optional parameters”.
- We have already seen a few examples of calling procedures with Positional notation.

```
Create_Dept (New_Deptno=> 50, New_Dname=>'FINANCE');
```

3.3: Procedures

Positional notation: Example

```
CREATE OR REPLACE PROCEDURE Create_Dept(deptno number,dname
varchar2,location varchar2) as
BEGIN
INSERT INTO dept VALUES(deptno,dname,location);
END;
```

- Executing a procedure using positional parameter notation is as follows:

```
SQL>execute Create_Dept(90,'sales','mumbai');
```



Copyright © Capgemini 2015. All Rights Reserved 14

Positional Notation :

You specify the parameters in the same order as they are declared in the procedure.

This notation is compact, but if you specify the parameters (especially literals) in the wrong order, the bug can be hard to detect.

You must change your code if the procedure's parameter list changes

3.3: Procedures

Named notation: Example

```
CREATE OR REPLACE PROCEDURE Create_Dept(deptno number,dname
varchar2,location varchar2) as
BEGIN
INSERT INTO dept VALUES(deptno,dname,location);
END;
```

- Executing a procedure using named parameter notation is as follows:

```
SQL>execute Create_Dept(deptno=>90,dname=>'sales',location=>'mumbai');
```

- Following procedure call is also valid :

```
SQL>execute Create_Dept(location=>'mumbai', deptno=>90,dname=>'sales');
```



Copyright © Capgemini 2015. All Rights Reserved 15

Named notation:

You specify the name of each parameter along with its value. An arrow (=>) serves as the “association operator”. The order of the parameters is not significant.

While executing the procedure, the names of the parameters must be the same as those in the procedure declaration.

3.3: Procedures

Mixed Notation Example:

```
CREATE OR REPLACE PROCEDURE Create_Dept(deptno number,dname
varchar2,location varchar2) as
BEGIN
INSERT INTO dept VALUES(deptno,dname,location);
END;
```

- Executing a procedure using mixed parameter notation is as follows:

```
SQL>execute Create_Dept(90, location=>'mumbai', dname=>'sales');
```



Copyright © Capgemini 2015. All Rights Reserved 16

Mixed notation:

You specify the first parameters with “Positional notation”, and then switch to “Named notation” for the last parameters.

You can use this notation to call procedures that have some “required parameters”, followed by some “optional parameters”.

3.4: Functions

Functions

- A function is similar to a procedure.
- A function is used to compute a value.
 - A function accepts one or more parameters, and returns a single value by using a return value.
 - A function can return multiple values by using OUT parameters.
 - A function is used as part of an expression, and can be called as Lvalue = Function_Name(Param1, Param2,).
 - Functions returning a single value for a row can be used with SQL statements.

3.4: Functions

Functions

▪ Syntax :

```
CREATE FUNCTION Func_Name(Param datatype :=  
value,...) RETURN datatype1 AS  
    Variable_Declaration ;  
    Cursor_Declaration ;  
    Exception_Declaration ;  
BEGIN  
    PL/SQL_Statements ;  
    RETURN Variable_Or_Value_Of_Type_Datatype1 ;  
EXCEPTION  
    Exception_Definition ;  
END Func_Name ;
```

3.4: Functions

Examples on Functions

▪ Example 1:

```
CREATE FUNCTION Crt_Dept(dno number,
  dname varchar2) RETURN number AS
BEGIN
  INSERT into department_master
  VALUES (dno,dname);
  return 1;
EXCEPTION
  WHEN others THEN
    return 0;
END crt_dept;
```



Copyright © Capgemini 2015. All Rights Reserved. 19

Example 2:

- Function to calculate average salary of a department:
 - Function returns average salary of the department
 - Function returns -1, in case no employees are there in the department.
 - Function returns -2, in case of any other error.

```
CREATE OR REPLACE FUNCTION Get_Avg_Sal(p_deptno
in number) RETURN number AS
  V_Sal number;
BEGIN
  SELECT Trunc(Avg(staff_sal)) INTO V_Sal
  FROM staff_master
  WHERE deptno=P_Deptno;
  IF v_sal is null THEN
    v_sal := -1;
  END IF;
  return v_sal;
EXCEPTION
  WHEN others THEN
    return -2; --signifies any other errors
END get_avg_sal;
```

3.4: Functions

Executing a Function

■ Executing functions from SQL*PLUS:

- Create a bind variable Avg salary in SQLPLUS by using VARIABLE command as follows:
 - Execute the Function with EXECUTE command:
 - After execution, use SQL*PLUS PRINT command to view results.

```
variable flag number
```

```
EXECUTE :flag:=Crt_Dept(60,'Production');
```

```
PRINT flag;
```



Copyright © Capgemini 2015. All Rights Reserved. 20

Functions can also be executed through command line as shown on the slide or can be called from other procedures/functions/Anonymous PL/SQL blocks.

The second snippet calls the function and passes the actual parameters. The variable declared earlier is used for collecting the return value from the function
Calling the function from an anonymous PL/SQL block

```
DECLARE
    avgsalary number;
BEGIN
    avgsalary:=Get_Avg_Sal(20);
    dbms_output.put_line('The average salary of Dept 20 is'||
    avgsalary);
END;
```

Calling function using a Select statement

```
SELECT Get_Avg_Sal(30) FROM staff_master;
```

Summary

- In this lesson, you have learnt:
 - Subprograms in PL/SQL are named PL/SQL blocks.
 - There are two types of subprograms, namely: Procedures and Functions
 - Procedure is used to perform an action
 - Procedures have three subprogram parameter modes, namely: IN, OUT, and INOUT
 - Functions are used to compute a value
 - A function accepts one or more parameters, and returns a single value by using a return value
 - A function can return multiple values by using OUT parameters

Review – Questions

- Question 1: Anonymous Blocks do not have names.
 - True / False
- Question 2: A function can return multiple values by using OUT parameters
 - True / False



Review – Questions

- Question 3: An ____ parameter returns a value to the caller of a subprogram.
- Question 4: A procedure contains two parts: _____ and _____.
- Question 5: In ____ notation, the order of the parameters is not significant.

