

9

Creating Stored Procedures and Functions

ORACLE

Copyright © 2006, Oracle. All rights reserved.

PATITAPABAN PARIDA (pppparida9@gmail.com) has a non-transferable license to use this Student Guide.

Objectives

After completing this lesson, you should be able to do the following:

- **Differentiate between anonymous blocks and subprograms**
- **Create a simple procedure and invoke it from an anonymous block**
- **Create a simple function**
- **Create a simple function that accepts a parameter**
- **Differentiate between procedures and functions**

ORACLE

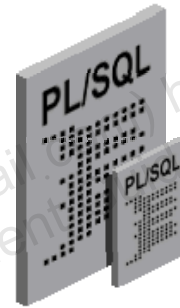
Copyright © 2006, Oracle. All rights reserved.

Lesson Aim

You have learned about anonymous blocks. This lesson introduces you to named blocks, which are also called *subprograms*. Procedures and functions are PL/SQL subprograms. In the lesson, you learn to differentiate between anonymous blocks and subprograms.

Procedures and Functions

- **Are named PL/SQL blocks**
- **Are called PL/SQL subprograms**
- **Have block structures similar to anonymous blocks:**
 - **Optional declarative section (without DECLARE keyword)**
 - **Mandatory executable section**
 - **Optional section to handle exceptions**



ORACLE

Copyright © 2006, Oracle. All rights reserved.

Procedures and Functions

Until this point, anonymous blocks are the only examples of PL/SQL code covered in this course. As the name indicates, *anonymous* blocks are unnamed executable PL/SQL blocks. Because they are unnamed, they can be neither reused nor stored for later use.

Procedures and functions are named PL/SQL blocks. They are also known as subprograms. These subprograms are compiled and stored in the database. The block structure of the subprograms is similar to the structure of anonymous blocks. Subprograms can be declared not only at the schema level but also within any other PL/SQL block. A subprogram contains the following sections:

Declarative section: Subprograms can have an optional declarative section. However, unlike anonymous blocks, the declarative section of a subprogram does not start with the keyword DECLARE. The optional declarative section follows the keyword IS or AS in the subprogram declaration.

Executable section: This is the mandatory section of the subprogram, which contains the implementation of the business logic. Looking at the code in this section, you can easily determine the business functionality of the subprogram. This section begins and ends with the keywords BEGIN and END, respectively.

Exception section: This is an optional section that is included to handle exceptions.

Differences Between Anonymous Blocks and Subprograms

Anonymous Blocks	Subprograms
Unnamed PL/SQL blocks	Named PL/SQL blocks
Compiled every time	Compiled only once
Not stored in the database	Stored in the database
Cannot be invoked by other applications	Named and therefore can be invoked by other applications
Do not return values	Subprograms called functions must return values.
Cannot take parameters	Can take parameters

ORACLE

Copyright © 2006, Oracle. All rights reserved.

Differences Between Anonymous Blocks and Subprograms

The table in the slide not only shows the differences between anonymous blocks and subprograms, but also highlights the general benefits of subprograms.

Anonymous blocks are not persistent database objects. They are compiled and executed only once. They are not stored in the database for reuse. If you want to reuse, you must rerun the script that creates the anonymous block, which causes recompilation and execution. Procedures and functions are compiled and stored in the database in a compiled form. They are recompiled only when they are modified. Because they are stored in the database, any application can make use of these subprograms based on appropriate permissions. The calling application can pass parameters to the procedures if the procedure is designed to accept parameters. Similarly, a calling application can retrieve a value if it invokes a function or a procedure.

Procedure: Syntax

```
CREATE [OR REPLACE] PROCEDURE procedure_name
  [(argument1 [mode1] datatype1,
    argument2 [mode2] datatype2,
    . . .)]
IS|AS
procedure_body;
```

ORACLE

Copyright © 2006, Oracle. All rights reserved.

Procedure: Syntax

The slide shows the syntax for creating procedures. In the syntax:

<i>procedure_name</i>	Is the name of the procedure to be created
<i>argument</i>	Is the name given to the procedure parameter. Every argument is associated with a mode and data type. You can have any number of arguments separated by commas.
<i>mode</i>	Mode of argument: IN (default) OUT IN OUT
<i>datatype</i>	Is the data type of the associated parameter. The data type of parameters cannot have explicit size; instead, use %TYPE.
<i>Procedure_body</i>	Is the PL/SQL block that makes up the code

The argument list is optional in a procedure declaration. You learn about procedures in detail in the course titled *Oracle Database 10g: Develop PL/SQL Program Units*.

Procedure: Example

```

...
CREATE TABLE dept AS SELECT * FROM departments;
CREATE PROCEDURE add_dept IS
  dept_id dept.department_id%TYPE;
  dept_name dept.department_name%TYPE;
BEGIN
  dept_id:=280;
  dept_name:='ST-Curriculum';
  INSERT INTO dept(department_id,department_name)
  VALUES(dept_id,dept_name);
  DBMS_OUTPUT.PUT_LINE(' Inserted ' ||
    SQL%ROWCOUNT || ' row ');
END;
/

```

ORACLE

Copyright © 2006, Oracle. All rights reserved.

Procedure: Example

Examine the code in the slide. The `add_dept` procedure inserts a new department with department ID 280 and department name ST-Curriculum. The procedure declares two variables, `dept_id` and `dept_name`, in the declarative section. The declarative section of a procedure starts immediately after the procedure declaration and does not begin with the keyword `DECLARE`. The procedure uses the implicit cursor attribute or the `SQL%ROWCOUNT` SQL attribute to verify whether the row was successfully inserted. `SQL%ROWCOUNT` should return 1 in this case.

Note: When you create any object (such as a table, procedure, function, and so on), the entries are made to the `user_objects` table. When the code in the slide is executed successfully, you can check the `user_objects` table by issuing the following command:

```
SELECT object_name,object_type FROM user_objects;
```

ADD_DEPT	PROCEDURE
DEPT	TABLE

Procedure: Example (continued)

The source of the procedure is stored in the `user_source` table. You can check the source for the procedure by issuing the following command:

```
SELECT * FROM user_source WHERE name='ADD_DEPT';
```

NAME	TYPE	LINE	TEXT
ADD_DEPT	PROCEDURE	1	PROCEDURE add_dept IS
ADD_DEPT	PROCEDURE	2	dept_id dept.department_id%TYPE;
ADD_DEPT	PROCEDURE	3	dept_name dept.department_name%TYPE;
ADD_DEPT	PROCEDURE	4	BEGIN
ADD_DEPT	PROCEDURE	5	dept_id:=280;
ADD_DEPT	PROCEDURE	6	dept_name:='ST-Curriculum';
ADD_DEPT	PROCEDURE	7	INSERT INTO dept(department_id,department_name)
ADD_DEPT	PROCEDURE	8	VALUES(dept_id,dept_name);
ADD_DEPT	PROCEDURE	9	DBMS_OUTPUT.PUT_LINE(' Inserted ' SQL%ROWCOUNT ' row ');
ADD_DEPT	PROCEDURE	10	END;
ADD_DEPT	PROCEDURE	11	

Invoking the Procedure

```
BEGIN
  add_dept;
END;
/
SELECT department_id, department_name FROM
dept WHERE department_id=280;
```

Inserted 1 row
PL/SQL procedure successfully completed.

DEPARTMENT_ID	DEPARTMENT_NAME
280	ST-Curriculum

ORACLE

Copyright © 2006, Oracle. All rights reserved.

Invoking the Procedure

The slide shows how to invoke a procedure from an anonymous block. You have to include the call to the procedure in the executable section of the anonymous block. Similarly, you can invoke the procedure from any application, such as a forms application, Java application and so on. The select statement in the code checks to see if the row was successfully inserted.

You can also invoke a procedure with the SQL statement `CALL <procedure_name>`.

Function: Syntax

```
CREATE [OR REPLACE] FUNCTION function_name
  [(argument1 [mode1] datatype1,
    argument2 [mode2] datatype2,
    . . .)]
RETURN datatype
IS|AS
function_body;
```

ORACLE

Copyright © 2006, Oracle. All rights reserved.

Function: Syntax

The slide shows the syntax for creating a function. In the syntax:

<i>function_name</i>	Is the name of the function to be created
<i>argument</i>	Is the name given to the function parameter (Every argument is associated with a mode and data type. You can have any number or arguments separated by a comma. You pass the argument when you invoke the function.)
<i>mode</i>	Is the type of parameter (Only IN parameters should be declared.)
<i>datatype</i>	Is the data type of the associated parameter
RETURN <i>datatype</i>	Is the data type of the value returned by the function
<i>function_body</i>	Is the PL/SQL block that makes up the function code

The argument list is optional in function declaration. The difference between a procedure and a function is that a function must return a value to the calling program. Therefore, the syntax contains *return_type*, which specifies the data type of the value that the function returns. A procedure may return a value via an OUT or IN OUT parameter.

Function: Example

```

CREATE FUNCTION check_sal RETURN Boolean IS
  dept_id employees.department_id%TYPE;
  empno    employees.employee_id%TYPE;
  sal      employees.salary%TYPE;
  avg_sal  employees.salary%TYPE;
BEGIN
  empno:=205;
  SELECT salary,department_id INTO sal,dept_id
  FROM employees WHERE employee_id= empno;
  SELECT avg(salary) INTO avg_sal FROM employees
  WHERE department_id=dept_id;
  IF sal > avg_sal THEN
    RETURN TRUE;
  ELSE
    RETURN FALSE;
  END IF;
EXCEPTION
  WHEN NO_DATA_FOUND THEN
    RETURN NULL;
END;
/

```

ORACLE

Copyright © 2006, Oracle. All rights reserved.

Function: Example

The `check_sal` function is written to determine whether the salary of a particular employee is greater than or less than the average salary of all employees working in the same department. The function returns `TRUE` if the salary of the employee is greater than the average salary of employees in the department; if not, it returns `FALSE`. The function returns `NULL` if a `NO_DATA_FOUND` exception is thrown.

Note that the function checks for the employee with the employee ID 205. The function is hard-coded to check for this employee ID only. If you want to check for any other employees, you must modify the function itself. You can solve this problem by declaring the function so that it accepts an argument. You can then pass the employee ID as parameter.

Invoking the Function

```

SET SERVEROUTPUT ON
BEGIN
  IF (check_sal IS NULL) THEN
    DBMS_OUTPUT.PUT_LINE('The function returned
      NULL due to exception');
  ELSIF (check_sal) THEN
    DBMS_OUTPUT.PUT_LINE('Salary > average');
  ELSE
    DBMS_OUTPUT.PUT_LINE('Salary < average');
  END IF;
END;
/

```

Salary > average
PL/SQL procedure successfully completed.

ORACLE

Copyright © 2006, Oracle. All rights reserved.

Invoking the Function

You include the call to the function in the executable section of the anonymous block. The function is invoked as a part of a statement. Remember that the `check_sal` function returns `Boolean` or `NULL`. Thus the call to the function is included as the conditional expression for the `IF` block.

Note: You can use the `DESCRIBE` command to check the arguments and return type of the function, as in the following example:

```
DESCRIBE check_sal;
```

Passing a Parameter to the Function

```

DROP FUNCTION check_sal;
CREATE FUNCTION check_sal(empno employees.employee_id%TYPE)
RETURN Boolean IS
    dept_id employees.department_id%TYPE;
    sal      employees.salary%TYPE;
    avg_sal  employees.salary%TYPE;
BEGIN
    SELECT salary,department_id INTO sal,dept_id
    FROM employees WHERE employee_id=empno;
    SELECT avg(salary) INTO avg_sal FROM employees
    WHERE department_id=dept_id;
    IF sal > avg_sal THEN
        RETURN TRUE;
    ELSE
        RETURN FALSE;
    END IF;
EXCEPTION ...
...

```

ORACLE

Copyright © 2006, Oracle. All rights reserved.

Passing a Parameter to the Function

Remember that the function was hard-coded to check the salary of the employee with the employee ID 205. The code shown in the slide removes that constraint because it is re-written to accept the employee number as a parameter. You can now pass different employee numbers and check for the employee's salary.

You learn more about functions in the course titled *Oracle Database 10g: Develop PL/SQL Program Units*.

Invoking the Function with a Parameter

```
BEGIN
DBMS_OUTPUT.PUT_LINE('Checking for employee with id 205');
IF (check_sal(205) IS NULL) THEN
DBMS_OUTPUT.PUT_LINE('The function returned
  NULL due to exception');
ELSIF (check_sal(205)) THEN
DBMS_OUTPUT.PUT_LINE('Salary > average');
ELSE
DBMS_OUTPUT.PUT_LINE('Salary < average');
END IF;
DBMS_OUTPUT.PUT_LINE('Checking for employee with id 70');
IF (check_sal(70) IS NULL) THEN
DBMS_OUTPUT.PUT_LINE('The function returned
  NULL due to exception');
ELSIF (check_sal(70)) THEN
...
END IF;
END;
/
```

ORACLE

Copyright © 2006, Oracle. All rights reserved.

Invoking the Function with a Parameter

The code in the slide invokes the function twice by passing parameters. The output of the code is as follows:

```
Checking for employee with id 205
Salary > average
Checking for employee with id 70
The function returned NULL due to exception
PL/SQL procedure successfully completed.
```

Summary

In this lesson, you should have learned how to:

- **Create a simple procedure**
- **Invoke the procedure from an anonymous block**
- **Create a simple function**
- **Create a simple function that accepts parameters**
- **Invoke the function from an anonymous block**

ORACLE

Copyright © 2006, Oracle. All rights reserved.

Summary

You can use anonymous blocks to design any functionality in PL/SQL. However, the major constraint with anonymous blocks is that they are not stored and therefore cannot be reused. Instead of creating anonymous blocks, you can create PL/SQL subprograms. Procedures and functions are called subprograms, which are named PL/SQL blocks. Subprograms express reusable logic by virtue of parameterization. The structure of a procedure or a function is similar to the structure of an anonymous block. These subprograms are stored in the database and are therefore reusable.

Practice 9: Overview

This practice covers the following topics:

- **Converting an existing anonymous block to a procedure**
- **Modifying the procedure to accept a parameter**
- **Writing an anonymous block to invoke the procedure**

The Oracle logo, consisting of the word "ORACLE" in a bold, sans-serif font, is positioned on the right side of a red horizontal bar.

Copyright © 2006, Oracle. All rights reserved.

Practice 9

1. In *iSQL*Plus*, load the script `lab_02_04_soln.sql` that you created for question 4 of practice 2.
 - a. Modify the script to convert the anonymous block to a procedure called `greet`.
 - b. Execute the script to create the procedure.
 - c. Save your script as `lab_09_01_soln.sql`.
 - d. Click the Clear button to clear the workspace.
 - e. Create and execute an anonymous block to invoke the procedure `greet`. Sample output is shown below.

```

Hello World
TODAY IS : 20-JAN-04
TOMORROW IS : 21-JAN-04
PL/SQL procedure successfully completed.

```

2. Load the script `lab_09_01_soln.sql`.
 - a. Drop the procedure `greet` by issuing the following command:
`DROP PROCEDURE greet`
 - b. Modify the procedure to accept an argument of type `VARCHAR2`. Call the argument `name`.
 - c. Print Hello *<name>* instead of printing Hello World.
 - d. Save your script as `lab_09_02_soln.sql`.
 - e. Execute the script to create the procedure.
 - f. Create and execute an anonymous block to invoke the procedure `greet` with a parameter. Sample output is shown below.

```

Hello Neema
TODAY IS : 20-JAN-04
TOMORROW IS : 21-JAN-04
PL/SQL procedure successfully completed.

```