9

Creating Stored Procedures and Functions

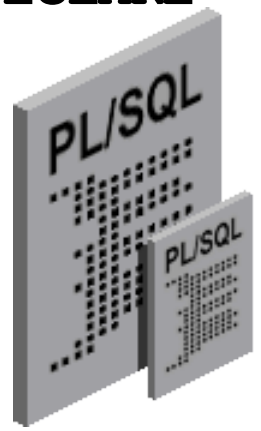
Objectives

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

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



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	They are 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

Procedure: Syntax

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

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;
/
```

Procedure: Example

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

Function: Syntax

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

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;
/
```

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.
```

Passing 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 ...
...
```

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;
/
```

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

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

Practice 9: Overview