

# Database Programming with PL/SQL

Creating Functions

ORACLE®

ACADEMY

# Objectives

This lesson covers the following objectives:

- Define a stored function
- Create a PL/SQL block containing a function
- List ways in which a function can be invoked
- Create a PL/SQL block that invokes a function that has parameters
- List the development steps for creating a function
- Describe the differences between procedures and functions

# Purpose

In this lesson, you learn how to create and invoke functions. A function is a subprogram that must return exactly one value.

A procedure is a standalone executable statement, whereas a function can only exist as part of an executable statement.

Functions are an integral part of modular code. Business rules and/or formulas can be placed in functions so that they can be easily reused.

# What Is a Stored Function?

A function is a named PL/SQL block (a subprogram) that can accept optional IN parameters and must return a single output value.

Functions are stored in the database as schema objects for repeated execution.

## What Is a Stored Function? (cont.)

A function can be called as part of a SQL expression or as part of a PL/SQL expression.

- Certain return types (Boolean, for example) prevent a function from being called as part of a `SELECT`.
- In SQL expressions, a function must obey specific rules to control side effects. Side effects to be avoided are:
  - Any kind of DML or DDL
  - `COMMIT` or `ROLLBACK`
  - Altering global variables

## What Is a Stored Function? (cont.)

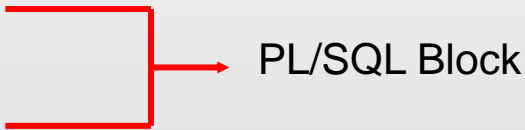
- In PL/SQL expressions, the function identifier acts like a variable whose value depends on the parameters passed to it.

# Syntax for Creating Functions

The header for a function is like a `PROCEDURE` header with two differences:

1. The parameter mode should only be `IN`.
2. The `RETURN` clause is used instead of an `OUT` mode.

```
CREATE [OR REPLACE] FUNCTION function_name
[(parameter1 [mode] datatype1, ...)]
RETURN datatype IS|AS
[local_variable_declarations; ...]
BEGIN
  -- actions;
  RETURN expression;
END [function_name];
```



The PL/SQL block must have at least one `RETURN` statement.

## Syntax for Creating Functions (cont.)

A function is a PL/SQL subprogram that returns a single value. You must provide a `RETURN` statement to return a value with a data type that is consistent with the function declaration type.

You create new functions using the `CREATE [OR REPLACE] FUNCTION` statement which can declare a list of parameters, must return exactly one value, and must define the actions to be performed by the PL/SQL block.



# Stored Function With a Parameter: Example

Create the function:

```
CREATE OR REPLACE FUNCTION get_sal
(p_id employees.employee_id%TYPE)
RETURN NUMBER IS
  v_sal employees.salary%TYPE := 0;
BEGIN
  SELECT salary
  INTO   v_sal
  FROM   employees
  WHERE  employee_id = p_id;
  RETURN v_sal;
END get_sal;
```

Invoke the function as an expression or as a parameter value:

```
... v_salary := get_sal(100);
```

# Using RETURN

You can use RETURN from the executable section and/or from the EXCEPTION section.

Create the function:

```
CREATE OR REPLACE FUNCTION get_sal
(p_id employees.employee_id%TYPE) RETURN NUMBER IS
  v_sal employees.salary%TYPE := 0;
BEGIN
  SELECT salary INTO v_sal
    FROM employees WHERE employee_id = p_id;
  RETURN v_sal;
EXCEPTION
  WHEN NO_DATA_FOUND THEN RETURN NULL;
END get_sal;
```

Invoke the function as an expression with a bad parameter:

```
... v_salary := get_sal(999);
```

# Ways to Invoke (or Execute) Functions With Parameters

Functions can be invoked in the following ways:

- As part of PL/SQL expressions – use a local variable in an anonymous block to hold the returned value from a function.
- As a parameter to another subprogram – pass functions between subprograms.
- As an expression in a SQL statement - invoke a function as any other single-row function in a SQL statement.

# Invoking a Function as Part of a PL/SQL Expression

When invoking a function as part of a PL/SQL expression, you can use a local variable to store the returned result. In this example, `v_sal` is the local variable in an anonymous block that stores the results returned from the `get_sal` function.

```
DECLARE v_sal employees.salary%type;
BEGIN
    v_sal := get_sal(100); ...
END;
```

# Invoking a Function as a Parameter in Another Subprogram

You can also invoke a function as a parameter to another subprogram. In this example, the `get_sal` function with all its arguments is nested in the parameter required by the `DBMS_OUTPUT.PUT_LINE` procedure.

```
... DBMS_OUTPUT.PUT_LINE (get_sal (100)) ;
```

# Invoking a Function as an Expression in a SQL Statement

You can also invoke a function as an expression in a SQL statement. The following example shows how you can use a function as a single-row function in a SQL statement.

```
SELECT job_id, get_sal(employee_id) FROM employees;
```

**Note:** The restrictions that apply to functions when used in a SQL statement are discussed in the next lesson.

If functions are designed thoughtfully, they can be powerful constructs.

# Invoking Functions Without Parameters

Most functions have parameters, but not all. For example, the system functions `USER` and `SYSDATE` have no parameters.

- Invoke as part of a PL/SQL expression, using a local variable to obtain the result

```
DECLARE v_today DATE;  
BEGIN  
    v_today := SYSDATE; ...  
END;
```

- Use as a parameter to another subprogram

```
... DBMS_OUTPUT.PUT_LINE (USER);
```

# Invoking Functions Without Parameters (cont.)

- Use in a SQL statement (subject to restrictions)

```
SELECT job_id, SYSDATE-hiredate FROM employees;
```



# Benefits and Restrictions That Apply to Functions

Benefits	Restrictions
Try things quickly: Functions allow you to temporarily display a value in a new format: a different case, annually vs. monthly (times 12), concatenated, or with substrings.	PL/SQL types do not completely overlap with SQL types. What is fine for PL/SQL (for example, <code>BOOLEAN</code> , <code>RECORD</code> ) might be invalid for a <code>SELECT</code> .
Extend functionality: Add new features, such as spell checking and parsing.	PL/SQL sizes are not the same as SQL sizes. For instance, a PL/SQL <code>VARCHAR2</code> variable can be up to 32 KB, whereas a SQL <code>VARCHAR2</code> column can be only up to 4 KB.

# Syntax Differences Between Procedures and Functions

## Procedures

```
CREATE [OR REPLACE] PROCEDURE name [parameters] IS|AS (Mandatory)
    Variables, cursors, etc. (Optional)
BEGIN (Mandatory)
    SQL and PL/SQL statements;
EXCEPTION (Optional)
    WHEN exception-handling actions;
END [name]; (Mandatory)
```

## Functions

```
CREATE [OR REPLACE] FUNCTION name [parameters] (Mandatory)
    RETURN datatype IS|AS (Mandatory)
    Variables, cursors, etc. (Optional)
BEGIN (Mandatory)
    SQL and PL/SQL statements;
    RETURN ...; (One Mandatory, more optional)
EXCEPTION (Optional)
    WHEN exception-handling actions;
END [name]; (Mandatory)
```

# Differences/Similarities Between Procedures and Functions

Procedures	Functions
Execute as a PL/SQL statement	Invoke as part of an expression
Do not contain RETURN clause in the header	Must contain a RETURN clause in the header
Can return values (if any) in output parameters	Must return a single value
Can contain a RETURN statement without a value	Must contain at least one RETURN statement

Both can have zero or more IN parameters that can be passed from the calling environment. Both have the standard block structure including exception handling.

# Differences Between Procedures and Functions

## Procedures

You create a procedure to store a series of actions for later execution. A procedure does not have to return a value. A procedure can call a function to assist with its actions.

Note: A procedure containing a single `OUT` parameter might be better rewritten as a function returning the value.

# Differences Between Procedures and Functions (cont.)

## Functions

You create a function when you want to compute a value that must be returned to the calling environment. Functions return only a single value, and the value is returned through a `RETURN` statement.

The functions used in SQL statements cannot use `OUT` or `IN OUT` modes. Although a function using `OUT` can be invoked from a PL/SQL procedure or anonymous block, it cannot be used in SQL statements.

# Terminology

Key terms used in this lesson included:

- Stored function

# Summary

In this lesson, you should have learned how to:

- Define a stored function
- Create a PL/SQL block containing a function
- List ways in which a function can be invoked
- Create a PL/SQL block that invokes a function that has parameters
- List the development steps for creating a function
- Describe the differences between procedures and functions