

Database Programming with PL/SQL

Creating Packages

ORACLE®

ACADEMY

Objectives

This lesson covers the following objectives:

- Describe the reasons for using a package
- Describe the two components of a package: specification and body
- Create packages containing related variables, cursors, constants, exceptions, procedures, and functions
- Create a PL/SQL block that invokes a package construct

Purpose

You have already learned how to create and use stored procedures and functions.

Suppose you want to create several procedures and/or functions that are related to each other. An application can use either all of them or none of them.

Wouldn't it be easier to create and manage all the subprograms as a single database object: a package?

Purpose (cont.)

In this lesson, you learn what a package is and what its components are. You also begin to learn how to create and use packages.

What Are PL/SQL Packages?

PL/SQL packages are containers that enable you to group together related PL/SQL subprograms, variables, cursors, and exceptions.

For example, a Human Resources package can contain hiring and firing procedures, commission and bonus functions, and tax-exemption variables.

Components of a PL/SQL Package

A package consists of two parts stored separately in the database:

- **Package specification:** The interface to your applications. It must be created first. It declares the constructs (procedures, functions, variables, and so on) that are visible to the calling environment.
- **Package body:** This contains the executable code of the subprograms that were declared in the package specification. It can also contain its own variable declarations.

**Package
specification**



**Package
body**

Components of a PL/SQL Package (cont.)

The detailed package body code is invisible to the calling environment, which can see only the specification.

If changes to the code are needed, the body can be edited and recompiled without having to edit or recompile the specification.

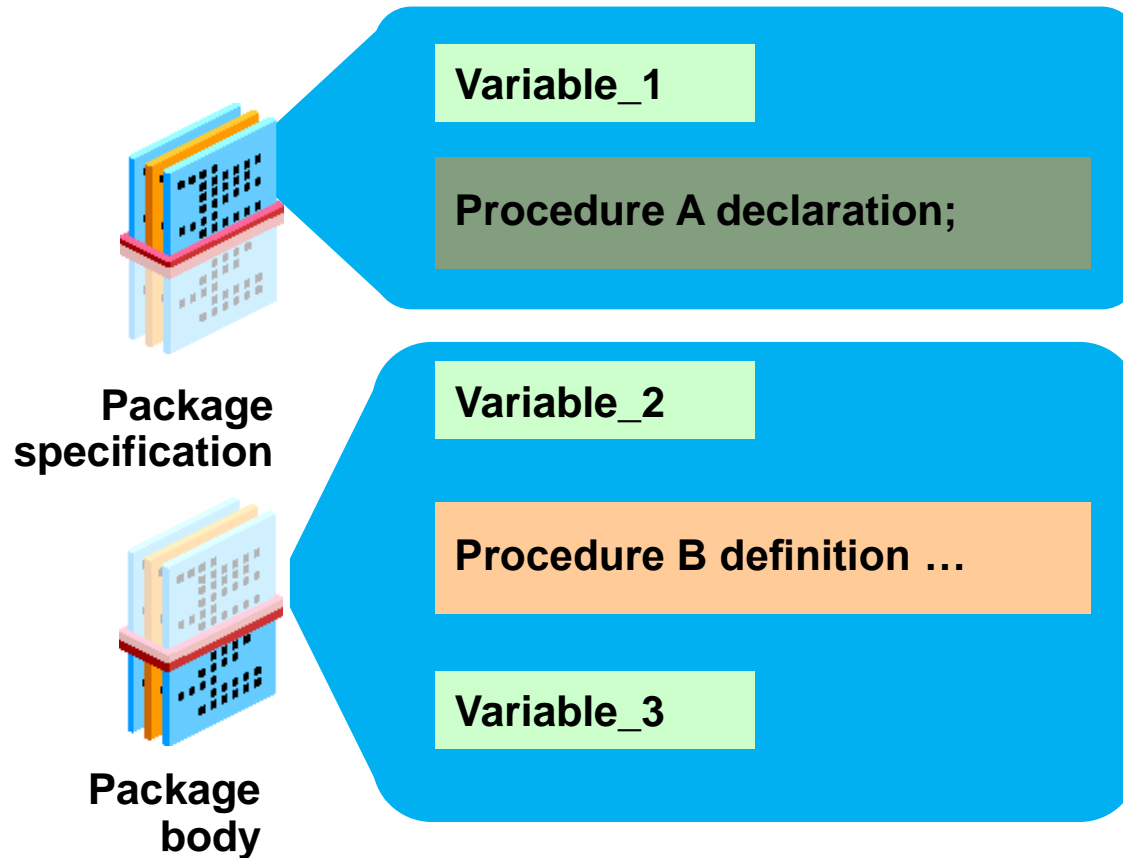
This two-part structure is an example of a modular programming principle called encapsulation.

**Package
specification**



**Package
body**

Components of a PL/SQL Package (cont.)



Syntax for Creating the Package Specification

To create packages, you declare all public constructs within the package specification.

- The `OR REPLACE` option drops and re-creates the package specification.
- Variables declared in the package specification are initialized to `NULL` by default.
- All the constructs declared in a package specification are visible to users who are granted `EXECUTE` privilege on the package.

```
CREATE [OR REPLACE] PACKAGE package_name
IS|AS
    public type and variable declarations
    public subprogram specifications
END [package_name];
```

Syntax for Creating the Package Specification (cont.)

- *package_name*: Specifies a name for the package that must be unique among objects within the owning schema. Including the package name after the `END` keyword is optional.
- *public type and variable declarations*: Declares public variables, constants, cursors, exceptions, user-defined types, and subtypes.
- *public subprogram specifications*: Declares the public procedures and/or functions in the package.

Creating the Package Specification

“Public” means that the package construct (variable, procedure, function, and so on) can be seen and executed from outside the package. All constructs declared in the package specification are automatically public constructs.

The package specification should contain procedure and function headings terminated by a semicolon, without the `IS` (or `AS`) keyword and its PL/SQL block.

Creating the Package Specification (cont.)

The implementation (writing the detailed code) of a procedure or function that is declared in a package specification is done in the package body. The next two slides show code examples.

Example of Package Specification: `check_emp_pkg`

- `G_MAX_LENGTH_OF_SERVICE` is a constant declared and initialized in the specification.
- `CHK_HIREDATE` and `CHK_DEPT_MGR` are two public procedures declared in the specification. Their detailed code is written in the package body.

```
CREATE OR REPLACE PACKAGE check_emp_pkg
IS
    g_max_length_of_service CONSTANT NUMBER := 100;
    PROCEDURE chk_hiredate
        (p_date      IN    employees.hire_date%TYPE);
    PROCEDURE chk_dept_mgr
        (p_empid     IN    employees.employee_id%TYPE,
         p_mgr       IN    employees.manager_id%TYPE);
END check_emp_pkg;
```

Package Specification: A Second Example

Remember that a cursor is a type of variable.

```
CREATE OR REPLACE PACKAGE manage_jobs_pkg
IS
    g_todays_date      DATE := SYSDATE;
    CURSOR jobs_curs IS
        SELECT employee_id, job_id FROM employees
            ORDER BY employee_id;
    PROCEDURE update_job
        (p_emp_id IN employees.employee_id%TYPE);
    PROCEDURE fetch_emps
        (p_job_id IN employees.job_id%TYPE,
         p_emp_id OUT employees.employee_id%TYPE);
END manage_jobs_pkg;
```

Syntax for Creating the Package Body

Create a package body to contain the detailed code for all the subprograms declared in the specification.

```
CREATE [OR REPLACE] PACKAGE BODY package_name IS|AS
    private type and variable declarations
    subprogram bodies
    [BEGIN initialization statements]
END [package_name];
```

Syntax for Creating the Package Body (cont.)

- The `OR REPLACE` option drops and re-creates the package body.
- “Subprogram bodies” must contain the code of all the subprograms declared in the package specification.
- Private types and variables, and `BEGIN` initialization statements are discussed in later lessons.

```
CREATE [OR REPLACE] PACKAGE BODY package_name IS|AS
    private type and variable declarations
    subprogram bodies
    [BEGIN initialization statements]
END [package_name];
```


Syntax for Creating the Package Body (cont.)

- *package_name* specifies a name for the package that must be the same as its package specification. Using the package name after the END keyword is optional.
- *subprogram bodies* specifies the full implementation (the detailed PL/SQL code) of all private and/or public procedures or functions.

```
CREATE [OR REPLACE] PACKAGE BODY package_name IS|AS
    private type and variable declarations
    subprogram bodies
    [BEGIN initialization statements]
END [package_name];
```

Creating the Package Body

When creating a package body, do the following:

- Specify the `OR REPLACE` option to overwrite an existing package body.
- Define the subprograms in an appropriate order. The basic principle is that you must declare a variable or subprogram before it can be referenced by other components in the same package body.
- Every subprogram declared in the package specification must also be included in the package body.

Example of Package Body: check_emp_pkg

```
CREATE OR REPLACE PACKAGE BODY check_emp_pkg IS
  PROCEDURE chk_hiredate
    (p_date      IN      employees.hire_date%TYPE)
  IS BEGIN
    IF MONTHS_BETWEEN(SYSDATE, p_date) >
      g_max_length_of_service * 12 THEN
      RAISE_APPLICATION_ERROR(-20200, 'Invalid Hiredate');
    END IF;
  END chk_hiredate;
  PROCEDURE chk_dept_mgr
    (p_empid     IN      employees.employee_id%TYPE,
     p_mgr       IN      employees.manager_id%TYPE)
  IS BEGIN ...
  END chk_dept_mgr;
END check_emp_pkg;
```

Changing the Package Body Code

Suppose now you want to make a change to the `CHK_HIREDATE` procedure, for example, to raise a different error message.

You must edit and recompile the package body, but you do not need to recompile the specification. Remember, the specification can exist without the body (but the body cannot exist without the specification).

Changing the Package Body Code (cont.)

Because the specification is not recompiled, you do not need to recompile any applications (or other PL/SQL subprograms) that are already invoking the package procedures.

Recompiling the Package Body: check_emp_pkg

```
CREATE OR REPLACE PACKAGE BODY check_emp_pkg IS
  PROCEDURE chk_hiredate
    (p_date      IN      employees.hire_date%TYPE)
  IS BEGIN
    IF MONTHS_BETWEEN(SYSDATE, p_date) >
      g_max_length_of_service * 12 THEN
      RAISE_APPLICATION_ERROR(-20201, 'Hiredate Too Old');
    END IF;
  END chk_hiredate;
  PROCEDURE chk_dept_mgr
    (p_empid     IN      employees.employee_id%TYPE,
     p_mgr       IN      employees.manager_id%TYPE)
  IS BEGIN ...
  END chk_dept_mgr;
END check_emp_pkg;
```

Describing a Package

You can `DESCRIBE` a package in the same way as you can `DESCRIBE` a table or view:

```
DESCRIBE check_emp_pkg
```

Object Type **PACKAGE** Object **CHECK_EMP_PKG**

Package Name	Procedure	Argument	In Out	Datatype
CHECK_EMP_PKG	CHK_DEPT_MGR	P_EMPID	IN	NUMBER
		P_MGR	IN	NUMBER
	CHK_HIREDATE	P_DATE	IN	DATE

You cannot `DESCRIBE` individual packaged subprograms, only the whole package.

Reasons for Using Packages

- **Modularity:** Related programs and variables can be grouped together.
- **Hiding information:** Only the declarations in the package specification are visible to invokers. Application developers do not need to know the details of the package body code.
- **Easier maintenance:** You can change and recompile the package body code without having to recompile the specification. Therefore, applications that already use the package do not need to be recompiled.

Terminology

Key terms used in this lesson included:

- Encapsulation
- `OR REPLACE`
- Package body
- Package specification
- PL/SQL packages

Summary

In this lesson, you should have learned how to:

- Describe the reasons for using a package
- Describe the two components of a package: specification and body
- Create packages containing related variables, cursors, constants, exceptions, procedures, and functions
- Create a PL/SQL block that invokes a package construct