Release Date: August, 2016

Updates:

Database Programming with PL/SQL

10-1
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.
- It might be helpful to group them together or in some way identify their relationship to each other.
- Oracle provides a way to do just that.

Purpose

- You can create and manage all the related subprograms as a single database object called a package.
- In this lesson, you learn what a package is and what its components are.
- You will also learn 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.

A package is a schema object that groups logically related subprograms and associated elements.

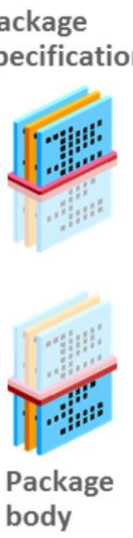
Packages have a long history in software engineering, offering important features for reliable, maintainable, reusable code, often in team development efforts for large systems.

Packages let you encapsulate logically related types, items, and subprograms in a named PL/SQL module. Each package is easy to understand, and the interfaces between packages are simple, clear, and well defined. This aids application development.

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.



The package specification is sometimes referred to as the Package Spec or Package Header.

It is possible to create a package specification with no corresponding package body. This is called a bodiless package, and contains no executable code. Bodiless packages are outside the scope of this course.

When designing an application, all you need initially is the interface information in the package specifications. You can code and compile a specification without its body. Then, stored subprograms that reference the package can be compiled as well. You need not define the package body fully until you are ready to complete the application.

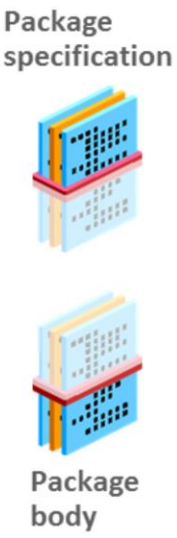
The principle of encapsulation states that, to invoke a subprogram, an application developer needs to know only how to call it: the name of the subprogram and the variables which must be passed to it. The application developer does not need to know how the package code works internally.

The specification (spec) is the interface for your applications; it declares the types, variables, constants, exceptions, cursors, and subprograms available for use. The body fully codes the cursors and subprograms, and so implements the spec.

The spec holds public declarations, which are visible to your application. You must declare subprograms at the end of the spec after all other items. The body holds implementation details and private declarations, which are hidden from your application.

Components of a PL/SQL Package

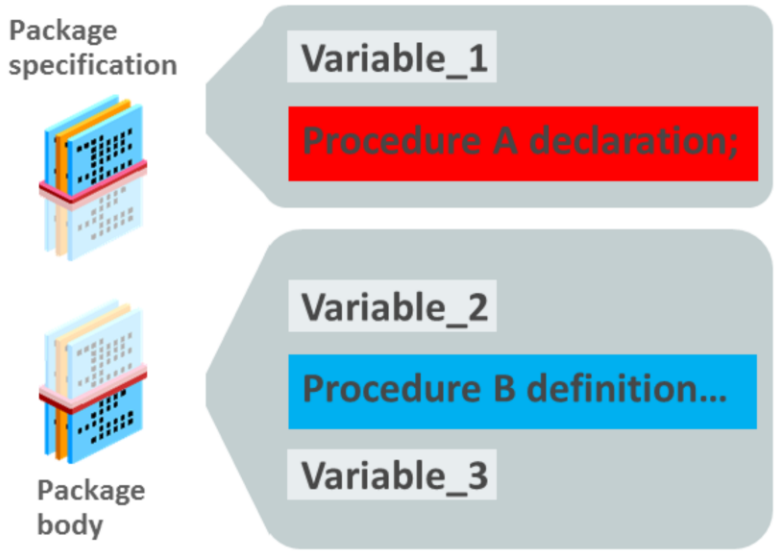
- The detailed package body code is invisible to the calling environment, which can see only the specification.
- If changes to the code are needed, the body can be edited and recompiled without having to edit or recompile the specification.
- This two-part structure is an example of a modular programming principle called encapsulation.



It may help to think of a container truck being driven along a road. The truck has two parts: the cab and the trailer. The cab can be driven without the trailer, but the trailer cannot go anywhere without the cab.

The principle of encapsulation states that, to invoke a subprogram, an application developer needs to know only how to call it: the name of the subprogram and the variables which must be passed to it. The application developer does not need to know how the package code works internally.

Components of a PL/SQL Package



Note the Procedure B in the slide.

It was not declared in the specification and therefore is invisible to (cannot be invoked from) a calling environment. This is an example of a private procedure, which will be explained in the next lesson.

Syntax for Creating the Package Specification

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

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

- The OR REPLACE option drops and re-creates the package specification.

Syntax for Creating the Package Specification

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

- *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.

Syntax for Creating the Package Specification

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

- *public type and variable declarations*: Declares public variables, constants, cursors, exceptions, user-defined types, and subtypes.
- Variables declared in the package specification are initialized to NULL by default.
- *public subprogram specifications*: Declares the public procedures and/or functions in the package.

Constructs declared in a package specification are visible to users granted EXECUTE privilege on the package.

Private types and variables, and BEGIN initialization statements, will be discussed in later lessons.

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.
- For all public procedures and functions, the package specification should contain the subprogram name and associated parameters terminated by a semicolon (not the actual code of the subprogram).

So the Spec contains a reference to all constructs you want to be public (i.e., accessible from outside the package). The Body contains all the actual code of the subprograms.

Creating the Package Specification

- The implementation (i.e., 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;
```


Although g_max_length_of_service is a constant, we have named it beginning with g_ (not c_) to show that it is a global (not local) variable.

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];
```

- *package_name* specifies a name for the package body that must be the same as its package specification.
- Using the package name after the END keyword is optional.

As with the Spec, the OR REPLACE option drops and re-creates the package body.

Syntax for Creating the Package Body

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

- Private types and variables, and `BEGIN` initialization statements, are discussed in later lessons.
- *`subprogram bodies`* must contain the code of all the subprograms declared in the package specification (i.e., the public subprograms) and the code for all private subprograms.

Private subprograms may only be called by other subprograms within the package.

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.

Create the package specification with the CREATE PACKAGE statement. Create the package body with the CREATE PACKAGE BODY statement.

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;
```


The slide shows the package body for CHECK_EMP_PKG. The detailed code for CHK_DEPT_MGR has been omitted to save space on the slide. The CHK_HIREDATE procedure is a validation procedure which checks that a date passed as an IN argument is not more than 100 years ago.

Notice how this procedure references the constant declared in the package specification in slide #13.

Also point out that the procedure END statements include the procedure-name: END chk_hiredate; We don't have to include it, but it makes the code more readable.

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).
- 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.

We need to edit and recompile the specification only if the procedure's name or parameters are changed.

Keeping the body of a package separate from the specification part of the package often makes life as a developer easier, so we tend to keep them in two separate files. This is because changes to the Body are far more frequent than changes to the Spec.

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
```

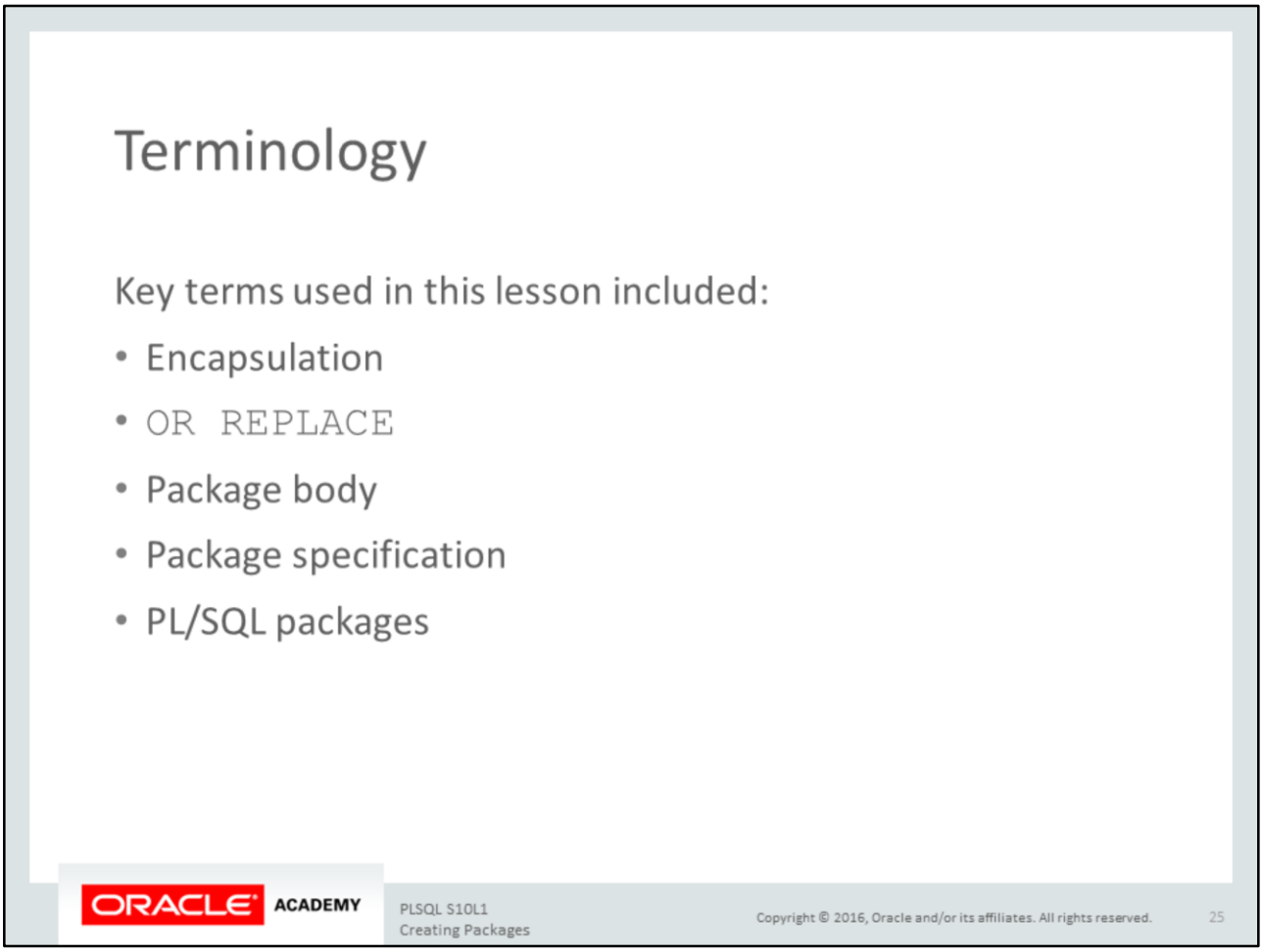
Object Type PACKAGE Object CHECK_EMP_PKG

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.



Encapsulation – A two-part structure in which the detailed package body code is invisible to the calling environment, which can only see the specification. If changes to the code are needed, the body can be edited and recompiled without having to edit or recompile the specification.

OR REPLACE – An option that drops and re-creates the package body.

Package body – This contains the executable code of the subprograms which were declared in the package specification. It may also contain its own variable declarations.

Package specification – The interface to your applications that declares the constructs (procedures, functions, variables and so on) which are visible to the calling environment.

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

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


ORACLE
ACADEMY