

Managing Package Concepts

What Will I Learn?

In this lesson, you will learn to:

- Explain the difference between public and private package constructs
- Designate a package construct as either public or private
- Specify the appropriate syntax to drop packages
- Identify views in the Data Dictionary that manage packages
- Identify guidelines for using packages

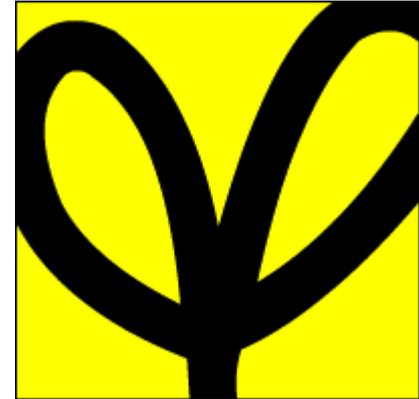


Why Learn It?

How would you create a procedure or function that cannot be invoked directly from an application (maybe for security reasons), but can be invoked only from other PL/SQL subprograms?

You would create a private subprogram within a package.

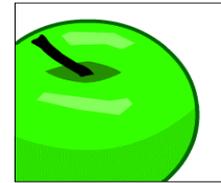
In this lesson, you learn how to create private subprograms. You also learn how to drop packages, and how to view them in the Data Dictionary. You also learn about the additional benefits of packages.



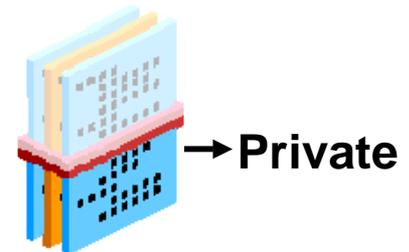
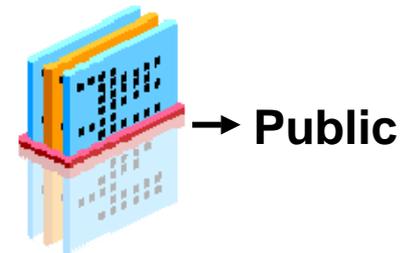
Tell Me / Show Me

Components of a PL/SQL Package

- Public components are declared in the package specification. You can invoke public components from any calling environment, provided the user has been granted `EXECUTE` privilege on the package.
- Private components are declared only in the package body and can be referenced only by other (public or private) constructs within the same package body. Private components can reference the package's public components.



**Package
specification**



**Package
body**

Tell Me / Show Me

Visibility of Package Components

The *visibility* of a component describes whether that component can be seen, that is, referenced and used by other components or objects. Visibility of components depends on where they are declared. You can declare components in three places within a package:

- Globally in the specification: These components are visible throughout the package body, and by the calling environment
- Locally in the package body, but outside any subprogram: These components are visible throughout the package body, but not by the calling environment
- Locally in the package body, within a specific subprogram: These components are visible only within that subprogram.

Tell Me / Show Me

Global/Local Compared to Public/Private:

Remember that public components declared in the specification are visible to the calling environment, while private components declared only within the body are not. Therefore all public components are global, while all private components are local.

So what's the difference between public and global, and between private and local?

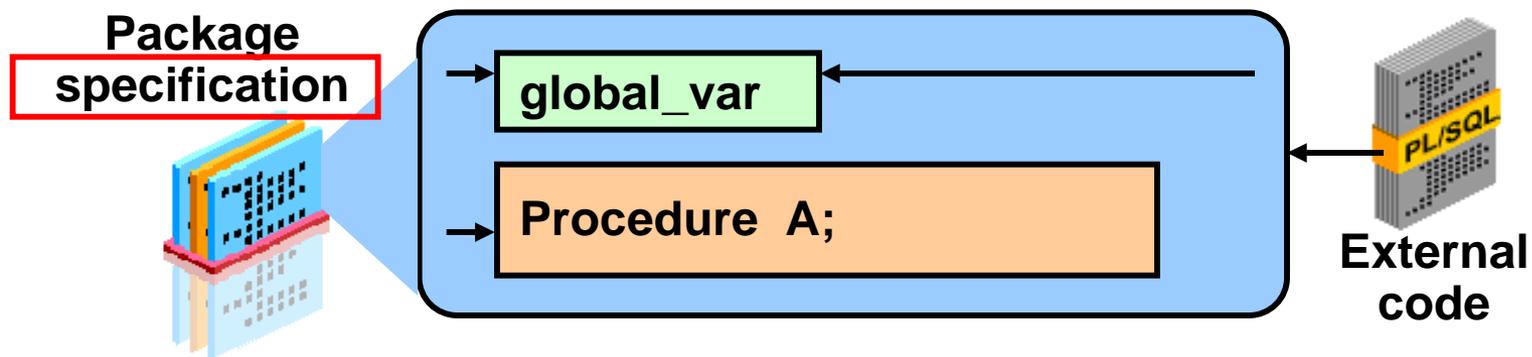
The answer is none, really. They are the same thing! But you use public/private when describing procedures and functions, and global/local when describing other components, such as variables, constants, and cursors.

Tell Me / Show Me

Visibility of Global (Public) Components

Globally declared components are visible internally and externally to the package, such as:

- A global variable declared in a package specification can be referenced and changed outside the package (for example, `global_var` can be referenced externally).
- A public subprogram declared in the specification can be called from external code sources (for example, `Procedure A` can be called from an environment external to the package).

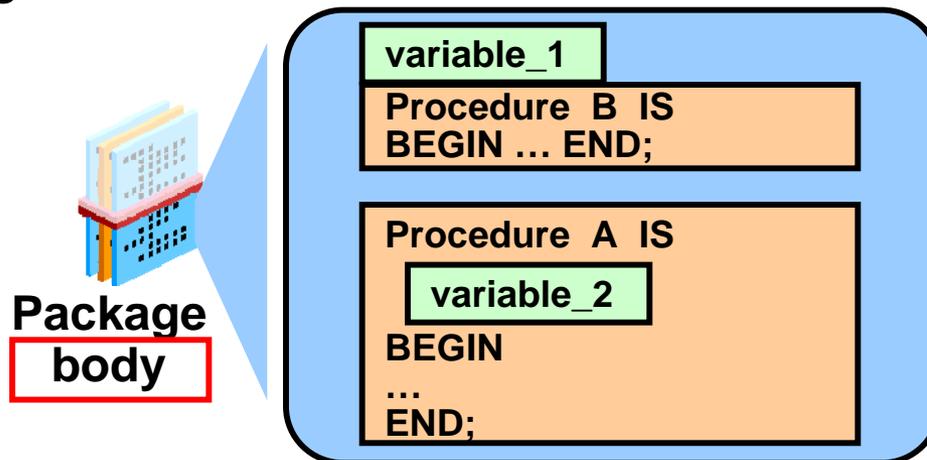


Tell Me / Show Me

Visibility of Local (Private) Components

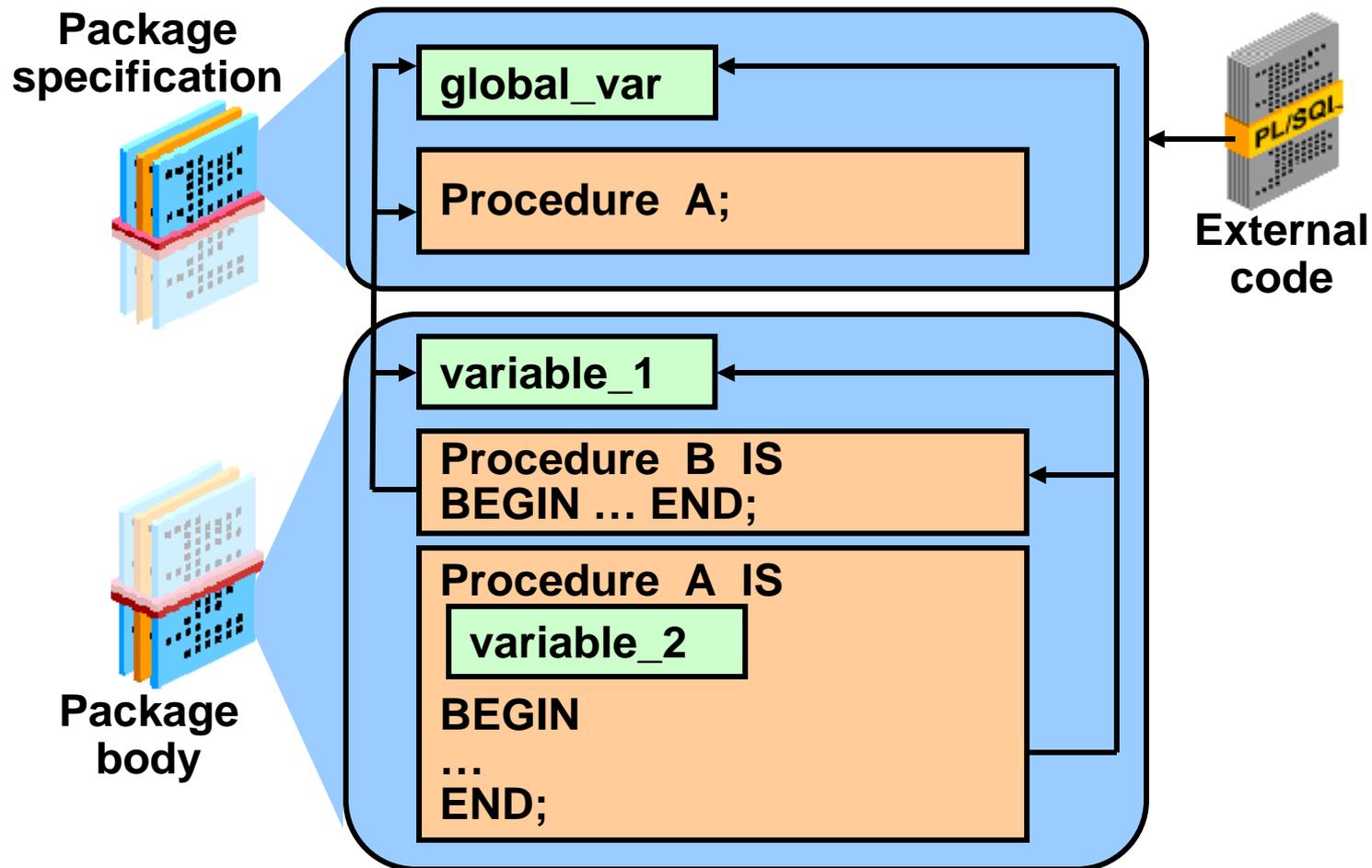
Local components are visible only within the structure in which they are declared, such as the following:

- Local variables defined within a specific subprogram can be referenced only within that subprogram, and are not visible to external components.
- Local variables that are declared in a package body can be referenced by other components in the same package body. They are not visible to any subprograms or objects that are outside the package.



Tell Me / Show Me

Note: Private subprograms, such as Procedure B, can be invoked only with public subprograms, such as Procedure A, or other private package constructs.





Tell Me / Show Me

Example of Package Specification: salary_pkg:

You have a business rule that no employee's salary can be increased by more than 20 percent at one time.

```
CREATE OR REPLACE PACKAGE salary_pkg
IS
  g_max_sal_raise CONSTANT NUMBER := 0.20;
  PROCEDURE update_sal
    (p_employee_id    employees.employee_id%TYPE,
     p_new_salary     employees.salary%TYPE);
END salary_pkg;
```

- g_max_sal_raise is a global constant initialized to 0.20.
- update_sal is a public procedure that updates an employee's salary.



Tell Me / Show Me

Example of Package Body: salary_pkg:

```
CREATE OR REPLACE PACKAGE BODY salary_pkg IS
  FUNCTION validate_raise -- private function
    (p_old_salary employees.salary%TYPE,
     p_new_salary employees.salary%TYPE)
  RETURN BOOLEAN IS
  BEGIN
    IF p_new_salary >
      (p_old_salary * (1 + g_max_sal_raise)) THEN
      RETURN FALSE;
    ELSE
      RETURN TRUE;
    END IF;
  END validate_raise;

  -- next slide shows the public procedure
```



Tell Me / Show Me

```
...  
PROCEDURE update_sal -- public procedure  
  (p_employee_id employees.employee_id%TYPE,  
   p_new_salary employees.salary%TYPE)  
IS v_old_salary employees.salary%TYPE; -- local variable  
BEGIN  
  SELECT salary INTO v_old_salary FROM employees  
    WHERE employee_id = p_employee_id;  
  IF validate_raise(v_old_salary, p_new_salary) THEN  
    UPDATE employees SET salary = p_new_salary  
      WHERE employee_id = p_employee_id;  
  ELSE  
    RAISE_APPLICATION_ERROR(-20210, 'Raise too high');  
  END IF;  
END update_sal;  
END salary_pkg;
```

Tell Me / Show Me

Invoking Package Subprograms

After the package is stored in the database, you can invoke subprograms stored within the same package or stored in another package.

Within the same package	Specify the subprogram name <pre>Subprogram;</pre> <p>You can fully qualify a subprogram within the same package, but this is optional</p> <pre>package_name.subprogram;</pre>
External to the package	Fully qualify the (public) subprogram with its package name <pre>package_name.subprogram;</pre>

 **Tell Me / Show Me**

Which of the following invocations from outside the `salary_pkg` are valid (assuming the caller either owns or has `EXECUTE` privilege on the package)?

```
DECLARE
    v_bool      BOOLEAN;
    v_number    NUMBER;
BEGIN
    salary_pkg.update_sal(100,25000);           -- 1
    update_sal(100,25000);                       -- 2
    v_bool := salary_pkg.validate_raise(24000,25000); -- 3
    v_number := salary_pkg.g_max_sal_raise;      -- 4
    v_number := salary_pkg.v_old_salary;        -- 5
END;
```



Tell Me / Show Me

Removing Packages

- To remove the **entire** package, specification, and body, use the following syntax:

```
DROP PACKAGE package_name;
```

- To remove **only** the package body, use the following syntax:

```
DROP PACKAGE BODY package_name;
```

- You cannot remove the package specification on its own.

Tell Me / Show Me

Viewing Packages in the Data Dictionary

The source code for PL/SQL packages is maintained and is viewable through the `USER_SOURCE` and `ALL_SOURCE` tables in the Data Dictionary.

- To view the package specification, use:

```
SELECT text
FROM   user_source
WHERE  name = 'SALARY_PKG' AND type = 'PACKAGE'
ORDER BY line;
```

- To view the package body, use:

```
SELECT text
FROM   user_source
WHERE  name = 'SALARY_PKG' AND type = 'PACKAGE BODY'
ORDER BY line;
```

Tell Me / Show Me

Guidelines for Writing Packages

- Construct packages for general use.
- Create the package specification before the body.
- The package specification should contain only those constructs that you want to be public/global.
- Only recompile the package body, if possible, because changes to the package specification require recompilation of all programs that call the package.
- The package specification should contain as few constructs as possible.

Tell Me / Show Me

Advantages of Using Packages

- Modularity: Encapsulating related constructs
- Easier maintenance: Keeping logically related functionality together
- Easier application design: Coding and compiling the specification and body separately
- Hiding information:
 - Only the declarations in the package specification are visible and accessible to applications.
 - Private constructs in the package body are hidden and inaccessible.
 - All coding is hidden in the package body.

Tell Me / Show Me

Advantages of Using Packages

- Added functionality: Persistency of variables and cursors
- Better performance:
 - The entire package is loaded into memory when the package is first referenced.
 - There is only one copy in memory for all users.
 - The dependency hierarchy is simplified.
- Overloading: Multiple subprograms having the same name.

Persistency and Overloading are covered in later lessons in this section.

Dependencies are covered in a later section of this course.

Tell Me / Show Me

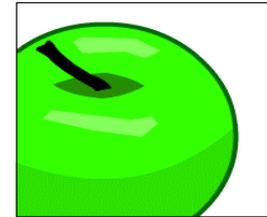
Terminology

Key terms used in this lesson include:

Public components

Private components

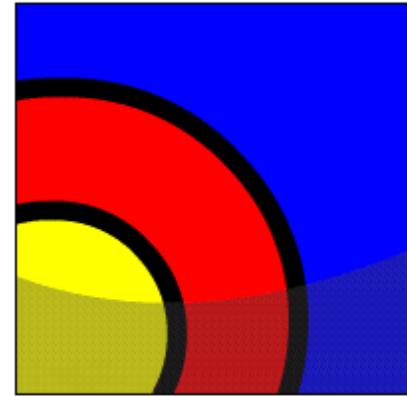
Visibility



Summary

In this lesson, you learned to:

- Explain the difference between public and private package constructs
- Designate a package construct as either public or private
- Specify the appropriate syntax to drop packages
- Identify views in the data dictionary that manage packages
- Identify guidelines for using packages



Try It / Solve It

The exercises in this lesson cover the following topics:

- Designating a package construct as either public or private
- Invoking a package construct
- Identifying views in the data dictionary to manage packages
- Dropping packages
- Identifying guidelines and benefits of packages

